Universal Serial Bus and the Multimedia PC

Kosar A. Jaff
Intel Architecture Labs
Intel Corporation
Hillsboro, OR
Kosar_A_Jaff@ccm.jf.intel.com

Introduction

The personal computer has evolved into a powerful multimedia appliance over the last several years. Spurred by such advances as powerful Intel microprocessors, advanced graphics subsystems, and highly capable software, the mainstream portion of the personal computer market segment has made impressive advances in multimedia capabilities. However, the PC has also continued to be plagued by an Achilles' heel--its unfriendly I/O subsystems. Users have continued to struggle with cryptic elements of the PC like IRQ, DMA, and I/O Addresses. The Universal Serial Bus (USB) should go a long way towards solving many of these problems and offers powerful new multimedia capabilities to help make the PC **the** ubiquitous multimedia appliance.

PC Peripherals -- Inside or Outside?

PC peripherals have generally come in two flavors--those that live inside the PC and those that live outside the PC. Examples of peripherals that live inside the PC include IDE disk controllers, SCSI bus host adapters, hard drives, internal CD-ROMs, internal fax/modems, and video adapters. Peripherals that live outside the PC include monitors, mice, keyboards, external modems, scanners, and external CD-ROMs.

Inside-the-box upgrades (or changes to the PC configuration) tend to be more complex and more intimidating. This stems from two major reasons: first, opening the PC box itself can be an intimidating and complex proposition, and second, buses that accommodate upgrades inside the PC (for example, the PC-AT ISA bus) have typically been hard to configure when a new adapter is introduced. Cables and assemblies inside the PC can often look like a rat's nest, confusing even the most sophisticated user. In addition, users installing new adapter cards in the PC have often been required to change arcane DIP switches on the add-in board or modify system BIOS settings.

Connecting outside-the-box peripherals has a key advantage over inside-the-box peripherals because the user isn't required to open the PC. Printers, modems, and the mouse are examples of outside-the-box peripherals that plug into ports on the PC. However, even a simple operation such as plugging a modem into a serial port can often lead to problems. The serial and parallel ports on a PC are based on dated technology that doesn't support the detection of devices as they are attached. Rather, configuration software makes "guesses" about what type of device is connected. Such detection isn't always successful, nor is it always correct in identifying the peripheral. In addition, serial and parallel ports have another major drawback; they can support only one device on the port at a time.

The difficulties in adding peripherals to the PC outlined above have led to a requirement for an easy-to-use, outside-the-box peripheral attachment point for the PC. Based on this and other requirements, a group of PC industry leaders defined and are currently implementing a new PC standard known as the Universal Serial Bus (USB). USB provides solutions to many of the issues outlined above, and allows PC users to focus on using the PC as an information appliance by spending less time on installation and configuration. USB provides a Plug and Play attachment point outside-the-box on the PC so users can simply attach USB peripherals and use them right away.

USB Architecture in a Nutshell

USB is a 12 megabit per second serial channel that can be used for a wide variety of peripherals. USB transfers signal and power over a four-wire cable, shown inFigure 1. USB can be used for keyboards, mice, modems, printers, scanners, CD-ROMs, audio devices (for example, microphones or digital speakers), digital cameras, and other multimedia-oriented peripherals. The USB architecture places no limitation on the type of device that is attached to the bus, but does provide guidelines and upper bounds on the amount of bus bandwidth a device can consume. Device designers need only be aware of the bandwidth their device requires before deciding if such a device can be designed for USB.

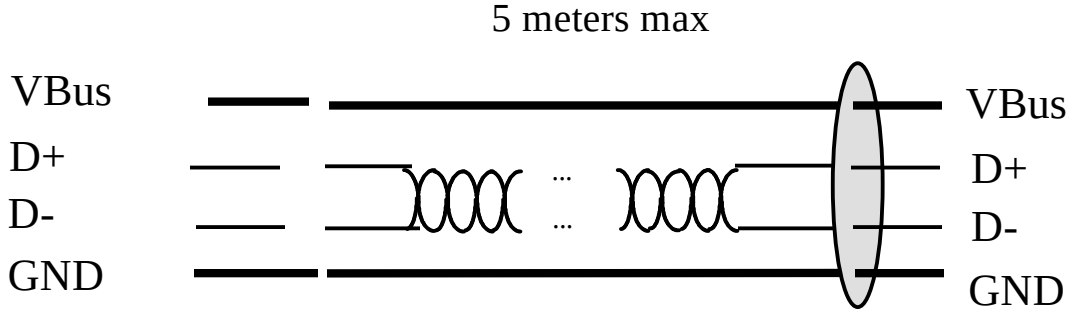

Figure 1: USB Cable

USB is a token-based bus, similar to other token-based buses such as the token ring network or FDDI. The USB Host Controller broadcasts tokens on the bus and a device that detects a match on the address in the token responds by either accepting or sending data to the host. The host also manages USB bus power by supporting suspend/resume operations.

USB uses a tiered star topology to allow simultaneous attachment of up to 127 devices on the bus at a time. At the root of the tiers is the USB Host Controller, which controls all traffic on the bus. The topology allows multiple devices to connect to a single logical bus without introducing delay to devices further downstream. Unlike other bus architectures, USB is not a store-and-forward bus, so there is no delay in sending a packet to a lower tier on the bus.

The USB hub device is a special component of the USB architecture. The hub is central to USB's architecture since it provides support for such key features as bus fanout (that is, providing additional attach points to the bus), detection of device attach and detach, and device power management. USB hubs are considered a "class" of USB device (USB device classes are explained in more detail below), and their operation is defined in detail in the USB specification. (SeeFigure 2.)

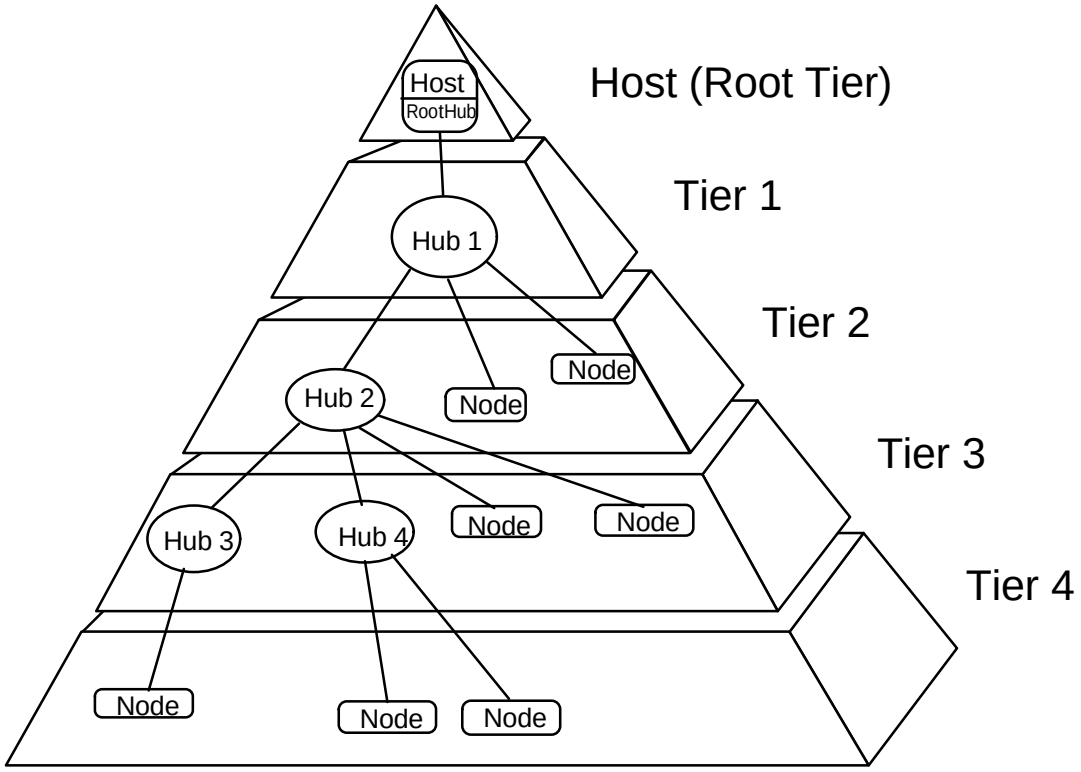

Figure 2: USB Bus Topology

Hot Attach and Detach

One of the major benefits of USB is its support for "hot" attachment and detachment of devices. This means that users can plug in and use a new USB device without having to shut off the system. When a new device has been detected by a USB hub, the host system is notified, and system software interrogates the device, determines its capabilities, and configures the device. In addition, system software loads the appropriate device driver so that the user's applications can begin using the new device immediately. Some operating systems refer to this process as "bus enumeration". Figure 3 outlines the steps in an example enumeration sequence on USB.

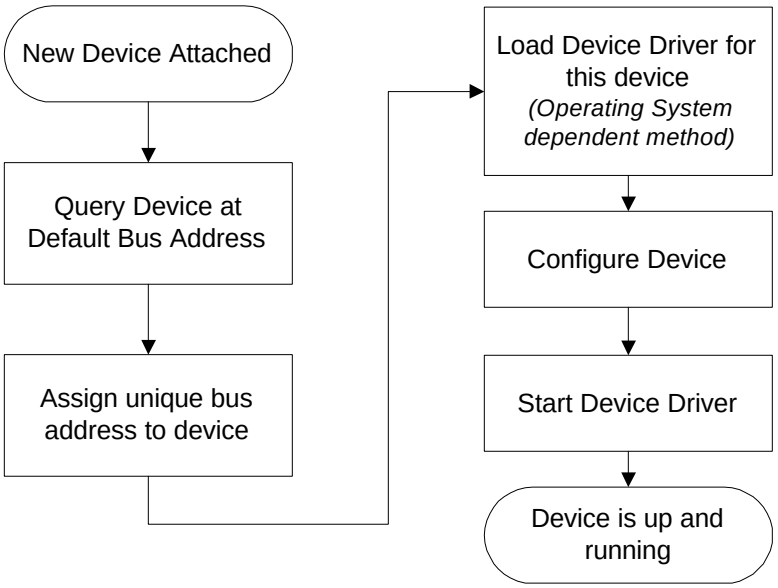

Figure 3: Example Enumeration Sequence

USB Device Classes

The USB specification defines a standard set of device operations that all USB devices must support[1]. These standard operations ensure that there is some consistency in basic device behavior when connecting to USB. In addition, USB device implementers have defined USB device classes that standardize the behavior of similar USB devices. For example, a class of USB device has been defined to encompass standard functions in USB Human Input Devices (HID). USB HID-class-compliant devices benefit from a base level of support built into most operating systems. For the user, this means that USB-aware operating systems configure and support devices that conform to the HID device class. When the user plugs in the device, standard class device drivers will be able to communicate with the device, determine its capabilities, and either handle the device at the base class level, or hand the device off to a device-specific driver. In this fashion, it's still possible for manufacturers to differentiate their products by providing additional functionality to that specified in the device class. However, supporting standard USB device class behavior will ensure universal acceptance of the conforming device.

Natural Data Types on USB

USB devices can take advantage of special bus features to deliver additional functionality to users. USB supports natural data type delivery for continuous data streams such as audio data and telephony data. Using USB's support for isochronous data transfers, devices can transmit and receive data in a guaranteed and predictable fashion. USB also allows non-isochronous devices to peacefully coexist with their data-driven brethren. For example, USB easily supports simultaneous traffic to a pair of digital audio speakers while using a joystick to play a game and spooling a document to a USB printer in the background. The bus architecture allows the audio data stream to proceed at the highest priority (as an isochronous

[1] USB Specification. *Chapter 9: USB Device Framework*; Revision 1.0, January, 1996.

device) while still allowing bus time for the joystick device. The printer consumes any bus time that is left over. USB is designed to provide such a balanced bus architecture while hiding the complexity from the devices connected to the bus. Bus bandwidth management and control is handled by the USB host controller and system software, and devices are left to focus on their specific functionality.

Built-in Robustness

USB incorporates several architectural features that contribute to overall bus robustness and reliability. USB devices enjoy a well-defined electrical and bus protocol. The electrical interface has been augmented with a set of standard test requirements that USB device manufacturers can use to ensure their electrical compliance. The bus protocol is relatively simple and based on other similar token-based bus protocols.

USB's electrical interface can be implemented in commodity CMOS components, which will allow rapid adoption and proliferation of USB building blocks. This in turn can lead to more peripherals with standard USB bus interfaces. USB electrical components should become readily available in 1996, with mass production slated to begin in late 1996 and early 1997.

To ensure homogeneity in device implementations, industry workshops are being conducted to allow device manufacturers to test their peripherals (using compliance tools) prior to taking products to market. Similar workshops have been conducted successfully in other industry initiatives such as PCI.

USB Software Architecture

USB's software architecture has been carefully designed to fit into modern operating systems' designs. For example, the USB software architecture is designed to allow migration and adoption in the Microsoft Windows* operating systems such as Windows 95 and Windows NT. USB takes advantage of existing system programming interfaces to accommodate device and bus management, device driver loading, and power management. USB bus-specific details are hidden by "plug-in" components in the operating system.

For example, a USB bus enumerator detects and configures USB devices in a bus-independent fashion on behalf of the Windows 95 Configuration Manager. In this fashion, USB software support can be added to existing operating systems with minimal impact on the operating system and the user.

In addition, modern operating systems such as Windows 95 and Windows NT isolate the hardware-specific software components to allow rapid adoption of new hardware technologies such as USB peripheral devices. For example, the Microsoft DirectSound* environment defines interfaces that allows the hardware-specific component to be isolated to a device driver. The hardware-specific driver is adapted to the specifics of the audio hardware. This simplifies the USB device manufacturer's job since the bulk of the system code has already been written and tested. Only one new piece (the hardware-specific driver) is added to the software "stack". See Figure 4.

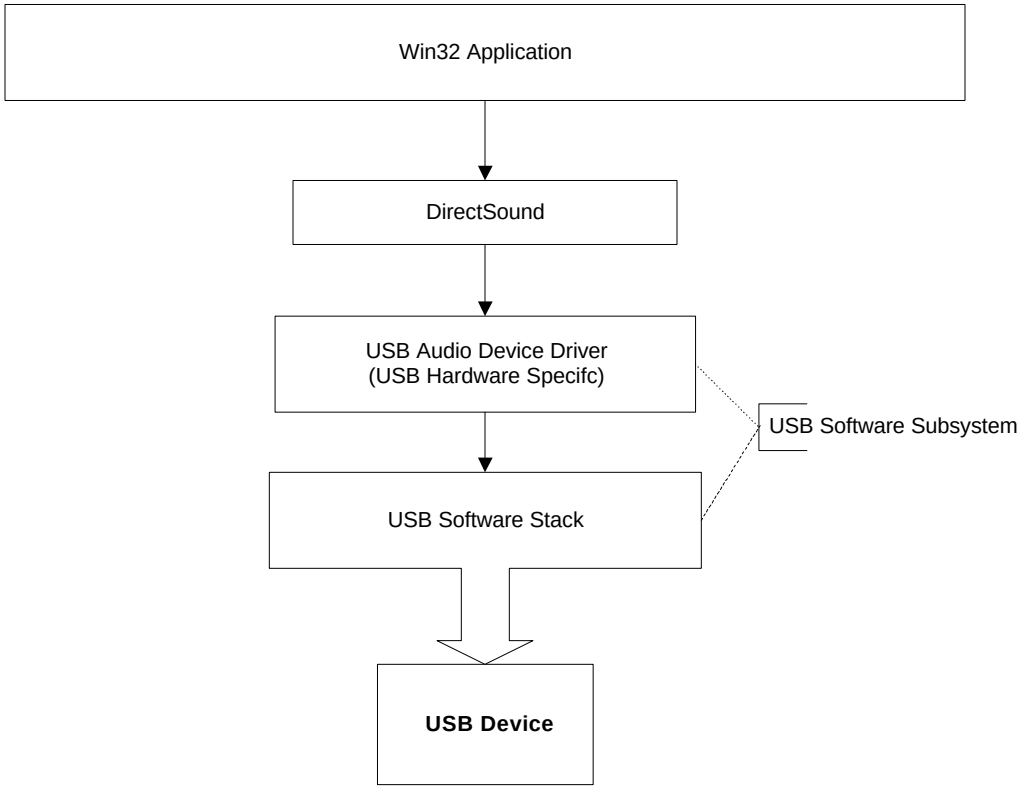

Figure 4: DirectSound* USB Example

A Modular Approach

The USB software architecture is based on a modular, object-oriented approach. The components of the USB system software stack are broken into three major modules. At the lowest level of the software architecture, the USB Host Controller is managed and controlled by a Host Controller Driver. USB Host Controller implementers define a standard hardware interface to provide a uniform host controller programming interface. For example, Intel's Host Controller, the 82430 PCISet USB Host Controller, implements the Universal Host Controller Interface (UHCI). UHCI is a hardware interface that has an associated device driver--the UHCI driver (UHCD). UHCD implements the underlying details of communicating with and controlling the Intel USB host controller. UHCD hides these details

from other components of the system software stack. Higher level components of the USB system software architecture are "layers" on top of the UHCD component and use UHCD's software interface to communicate with the host controller in a host-controller independent fashion. See Figure 5.

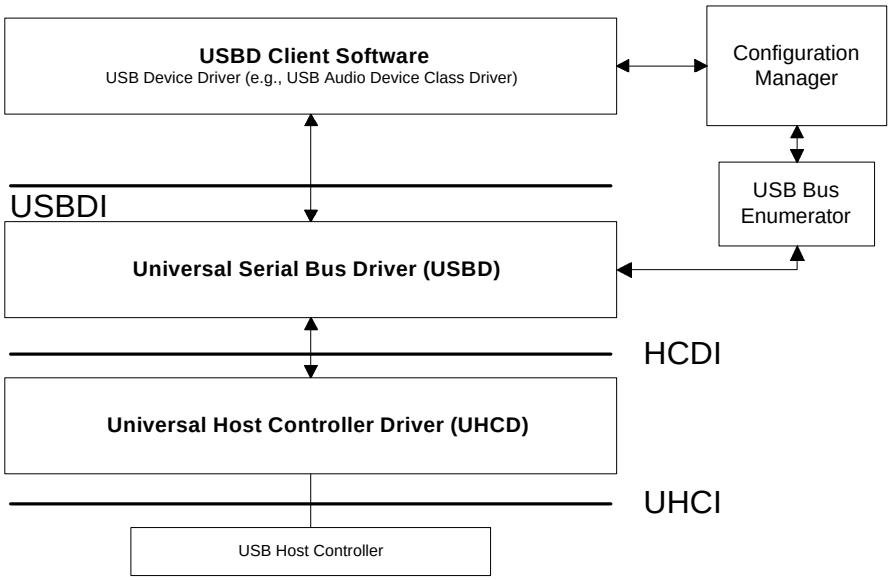

Figure 5: USB Software Architecture

The second major component of the USB software architecture is the Universal Serial Bus Driver (USBD). USBD provides a device driver-level interface designed to meet the requirements of existing device driver designs. In defining the USBD interface, the USB software architects studied how hardware-specific device drivers interacted with their devices and with other system services such as the Virtual DMA Device (VDMAD) in Windows 3.x and Windows 95. These case studies were important so that USB could design a device-driver interface that closely matched what the device drivers used and expected from the underlying system components.

Using the data from these case studies, the USB software architects defined the requirements of the USB Driver interface (USBDI) in the USB specification. The general requirements outlined in the specification reflect what services and operations USBD must perform. The exact details of USBD implementations differ in different operating system environments such as Windows 95 and Windows NT. However, the type of operations that USBD performs are similar in different operating system environments since the USB specification outlines the requirements of that software layer.

The third major component of the USB software architecture is the USBD client software. Usually USBD's clients are device drivers responsible for handling a specific USB device or class of device. In Windows 95, these device drivers use Plug and Play system features to handle USB device attach/detach and configuration events. Plug and Play USB device drivers allow dynamic configuration of a newly added device to insulate the users and applications from device configuration issues. In Windows 95, USB hubs are handled by a bus enumerator, which is nothing more than a USB hub device driver that uses Windows 95

Configuration Manager services to notify the system of device attach and detach events on USB. See Figure 6.

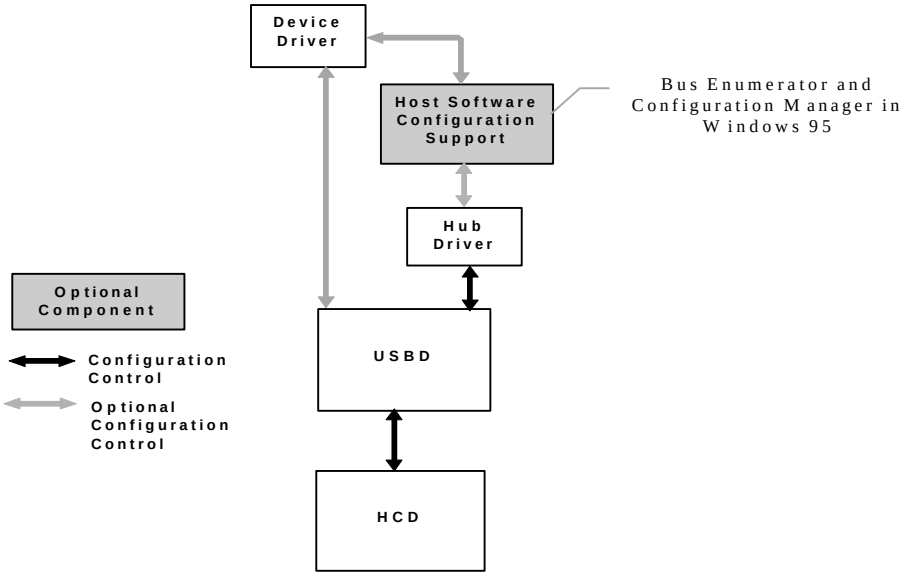

Figure 6: Configuration Interactions

Conclusion

The personal computer has undergone an evolution from a hobbyist-type device into a general purpose information appliance in the modern household. Novice PC users are much less tolerant of problems with installation or configuration , so PC peripheral manufacturers are always looking for ways to make the PC easier to use. To this end, the PC has needed an easy-to-use connection point to which they can connect a multitude of peripheral devices. USB addresses many of these issues and will provide an outside-the-box Plug and Play attach point that will accommodate a wide range of devices. USB will enable a variety of applications on the PC to enhance ease-of-use for existing peripherals while allowing room for device designers to dream up new and exciting peripherals to continually improve the PC usage experience.

For More Information

The USB Specification and other USB-related documents are available on the World Wide Web:

http://www.teleport.com/~usb

To join the USB Implementers Forum, contact:

USB IMPLEMENTERS FORUM
JF2-51, 2111 NE 25th Avenue,
Hillsboro, OR 97124