

# Designing a USB hub: basic options and issues

HARRY INIA, NATIONAL SEMICONDUCTOR, AND DOUG DYMENT, COMPUTER ACCESS TECHNOLOGY CORP

The Universal Serial Bus offers considerable flexibility for connecting peripherals to computers. For example, peripherals on a USB can be self- or bus-powered, as can the hubs that collectively make up a USB. Also, USB has both low- and medium-speed signal modes for simultaneously passing data from slow devices, such as mice, and faster devices, such as audio and video sources. Further, a USB hub can exist independently or as part of a peripheral unit, such as a monitor. In short, you have to consider numerous options when designing a USB hub. In addition, you need to understand basic USB-design issues before you get into deeper technical details.

First, consider some of USB's broader goals. USB can support virtually unlimited PC expansion "outside the box." USB provides a "one-size-fits-all" connector that enables a PC user to install and remove peripherals without opening a computer's case or juggling interrupt-request conflicts. In addition, USB's hot-insertion-and-removal feature allows installation or disconnection of peripherals while a PC stays up and running. However, because USB is essentially a user-configurable, outside-the-box peripheral bus, you need to design USB hubs as robust, easy-to-connect, user-proof devices.

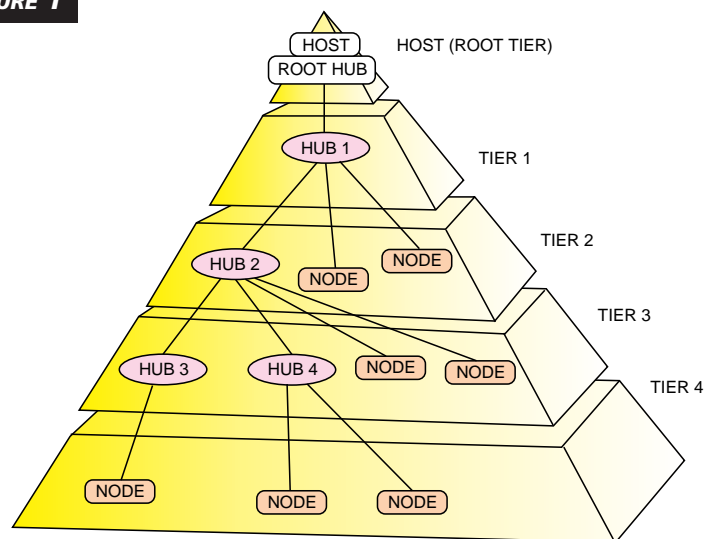
Another main goal of USB is to reduce the load on a PC's CPU and thus improve overall system performance. Consequently, management of all USB

All peripherals connect to the Universal Serial Bus through hubs. How you design a USB hub depends on the purpose you want it to serve.

peripherals occurs in a USB host controller and in subsidiary hub controllers. The host controller mounts on a PC's main board or on a PCI add-in card. USB system software in the PC's operating system manages the host controller.

USB physical interconnections form a tiered-star topology with a hub at the center of each star (Figure 1). Only one USB host controller (also referred to simply as a host) exists in any USB system. Associated with the host is a root hub, to

FIGURE 1



USB physical interconnections form a tiered-star topology. A hub is at the center of each star.

## USB HUBS

which other hubs attach. Each wire segment in the topology either connects the host's root hub to a hub or a peripheral or connects a hub to a peripheral or another hub.

Data on the USB flows through a bidirectional pipe regulated by the host controller and by its subsidiary hub controllers. Bus mastering allows allocation of USB's 12-Mbps bandwidth in two ways. USB can dynamically allocate bandwidth portions to various peripherals as needed, and it can reserve bandwidth for isochronous data transfers by specific peripherals. You can connect as many as 127 peripherals to one another by plugging USB's simple rectangular cable connectors into multiport hub controllers. The maximum cable-segment length for USB is 5m.

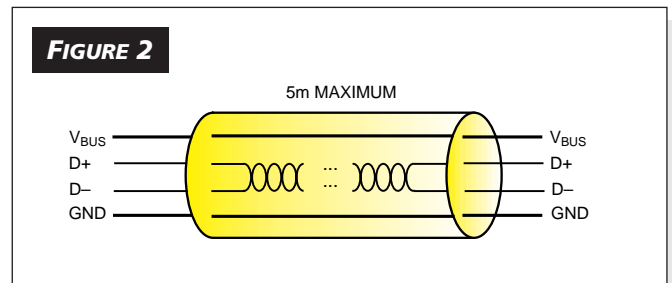
USB provides a peripheral interface for just about any low-to medium-speed device. Low-speed connections typically serve fairly slow interactive devices, such as keyboards and mice, that need data-transfer speeds of 10 to 100 kbps. A medium-speed isochronous connection, on the other hand, serves applications that need guaranteed latency and bandwidth at speeds of 500 kbps to 7 Mbps. These faster applications might handle audio and compressed video, for example. Devices that need still more bandwidth are probably best served by another bus, such as IEEE 1394.

In addition to offering plug and play with hot-swapping capabilities, USB is source-terminated, so you don't have to worry about device termination when you add or remove peripherals. When you connect a new device, the associated USB hub controller queries the device for a vendor ID, and the operating system chooses an appropriate device driver. USB further simplifies peripheral handling by using a single IRQ, port-address, and DMA channel.

USB transfers signals and power over a four-wire cable, with two wires used for a differential signal and the other two wires for power and ground (Figure 2). The signal rate for USB's full-speed mode is 12 Mbps; the limited-capability, low-speed mode operates at 1.5 Mbps. A USB system can simultaneously support both modes, switching between transfers in a manner that is transparent to the peripherals that are affected.

USB transmits clock data along with the differential data, using nonreturn-to-zero-invert (NRZI) data encoding with bit stuffing to ensure adequate transitions. A Sync field precedes each packet to allow the receivers to synchronize their bit-recovery clocks.

A USB cable's power and ground wires, denoted  $V_{BUS}$  and GND, respectively, can deliver power to an attached peripheral.  $V_{BUS}$  is nominally 5V at the source, and you can ensure an appropriate input voltage for an attached device by proper cable selection. Cables can be several meters long if you choose an appropriate conductor gauge to match a specified IR drop and other attributes, such as cable flexibility and the attached device's power budget. To provide guaranteed input-voltage levels and proper termination impedance, you need to use biased terminations at each end of the cable. These terminations also permit the detection of attachments and detachments at each port; they also differentiate between full-speed and low-speed devices.



**USB's four-wire cables carry differential signals as far as 5m.**

The USB specification allows USB hubs and peripherals to be either self- or bus-powered. Bus-powered peripherals can draw their power from USB as long as they don't need too much of it.

Self-powered hubs can accommodate power-hungry peripherals, such as floppy drives, but they require an extra ac socket on a user's power strip or uninterruptible power supply. Conversely, bus-powered hubs don't need their own power sources, but they must meet rigid overall power-distribution criteria, and they limit the types of peripherals that can attach to them. An alternative to a stand-alone, self-powered hub is a self-powered hub that is integrated with an existing device, such as a monitor, that already has power. This alternative provides the benefits of a stand-alone, self-powered hub without requiring a separate power connection.

The USB specification defines hubs' and peripherals' power-sourcing and sinking requirements in terms of "unit loads" of 100 mA. The specification establishes limitations for the following device (hub and peripheral) classes:

**Bus-powered hubs** draw all power for any internal functions and downstream ports from the USB-connector power pins. A bus-powered hub may draw as much as one load (100 mA) for itself and as much as four loads (400 mA) for external ports, for a total of five loads. External ports in a bus-powered hub can supply only one load per port regardless of the current drawn on the other ports of that hub. The hub must be able to supply this port current when the hub is in the active or suspended state.

**Self-powered hubs** (either stand-alone or device-integrated) draw no power for their internal functions and downstream ports from USB. However, the hub's USB interface may draw as much as one load from its upstream connection so the interface can function when the remainder of the hub is powered down. The hub must be able to supply as much as five unit loads on each of its external downstream ports, even when the hub is in the suspended state.

**Low-power, bus-powered peripherals** draw all their power from the USB connector. They may draw no more than one unit load at any time.

**High-power, bus-powered peripherals** draw all their power from the USB connector. They must draw no more than one unit load on power-up and may draw as much as five unit loads after being configured.

## USB HUBS

**Self-powered peripherals** may draw as much as one unit load from an upstream connection so that the interface can function when the remainder of the hub is powered down. All other power comes from an external (to USB) source.

An additional power-related concern that hub designers must address is the UL overcurrent-prevention requirement that no hub ever pass more than 5A onto any downstream port. The hub designer has a choice of implementing this current-limiting function either through gang switching, in which all ports power down if any port experiences an overcurrent condition, or through “per-port” switching, which powers down only the port that exceeds the 5A limit. Because per-port switching requires separate sensing circuitry for each port, it is more costly and thus preferable only for high-end, feature-rich hubs. Gang switching satisfies the UL safety requirement in low-end, cost-driven designs.

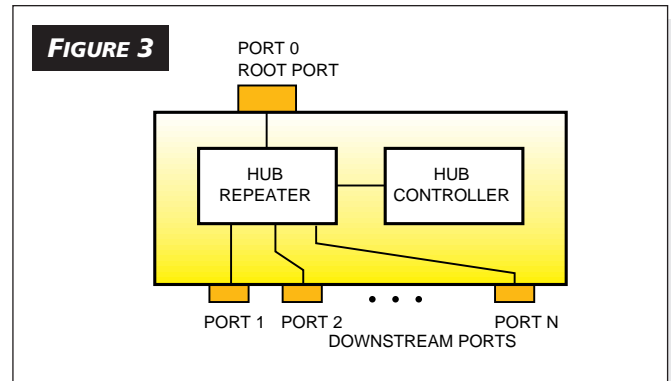
### Stand-alone, bus-powered hubs

A USB hub basically consists of a hub repeater and a hub controller (**Figure 3**). The repeater is a protocol-controlled switch between the upstream port and downstream ports. It also has hardware for handling reset and suspend/resume signaling. The controller provides the interface registers to allow communication to and from the host. Essentially, the repeater is responsible for managing connectivity on a per-packet basis, whereas the hub controller provides status and control and permits host access to the hub.

There are two key objectives in designing a stand-alone, bus-powered hub. One is to provide bus integrity and robust data transmission; the other is to live within the tight power constraints defined in the USB specification. Because bus-powered hubs must draw all their power from the bus for both internal functions and downstream ports, power efficiency is paramount. To further complicate matters, the bus-powered hub must support “hot” plugging and unplugging of devices without causing any attached device to lose its existing state and without causing any disruption in USB signaling.

The USB specification requires that the power draw of a bus-powered hub not exceed five unit loads (500 mA), that the hub’s circuitry receives a minimum dc voltage level of 4.75V from the host connector, and that the bus delivers at least 4.40V to all downstream connector ports. In addition to avoiding voltage-drop conditions for newly plugged devices, the USB spec also defines a condition called “voltage droop,” which refers to the impact on other ports when a new device plugs into the hub. Many designers focus their efforts on addressing the voltage drop for new devices but fail to pay adequate attention to the requirement for avoiding voltage droop in other ports. You can usually address the issue simply by adding a separate capacitor for each port to independently maintain an adequate current level.

Although a bus-powered hub does not have to deal with some of the overcurrent-limiting issues inherent in host and self-powered hubs, it does have to incorporate highly efficient internal power switching to avoid voltage loss for downstream ports. This issue can be especially significant



**FIGURE 3** A USB hub contains a hub repeater and a hub controller.

during a hot-plug event, when, for a fraction of a microsecond, the bus must supply current to the peripheral to charge the peripheral’s bulk capacitance to the nominal  $V_{BUS}$  level. In addition, to avoid overdrawing upstream current, the bus-powered hub’s switch must include inrush current limiting so that the hub’s downstream bulk capacitance charges slowly when it first powers up.

The net impact of these current-distribution requirements is that a bus-powered hub can have no more than four downstream ports, because each port consumes one unit load of 100 mA, and a fifth unit-load equivalent is then necessary to power the hub’s internal circuitry. Conversely, the basic core cost of a hub’s design makes it commercially impractical, at least for consumer markets, to design bus-powered hubs with fewer than four ports.

### Integrated, self-powered hubs

The most obvious limitation of a bus-powered hub is that it cannot support high-power-consumption peripherals. For instance, a bus-powered hub typically cannot support a high-powered, downstream floppy drive. Consequently, many PC manufacturers have established a default configuration that requires the first USB hub to be self-powered. Also, many monitor manufacturers are integrating this default hub into their monitors to provide a ready source for connecting peripherals. Integrating a USB hub controller into a monitor or other existing device also allows cost savings by eliminating some redundancies—separate monitor and hub  $\mu$ Cs, for example.

You can integrate a stand-alone USB hub built around a National LM1050 hub controller and a COP820  $\mu$ C. In a monitor-integrated design, the monitor’s existing  $\mu$ C can replace the COP820. The monitor’s  $\mu$ C can also use the hub controller’s programmable clock, thus eliminating the need for a separate crystal. The hub-controller chip can also convert 5V to 3.3V, which monitors don’t normally have, thus eliminating the need for a separate 3.3V power source. The chip’s 3.3V regulator gives the additional benefit of being able to disable this voltage, thus informing a host controller that the hub isn’t ready for communication.

Because USB performs the complex task of dynamically

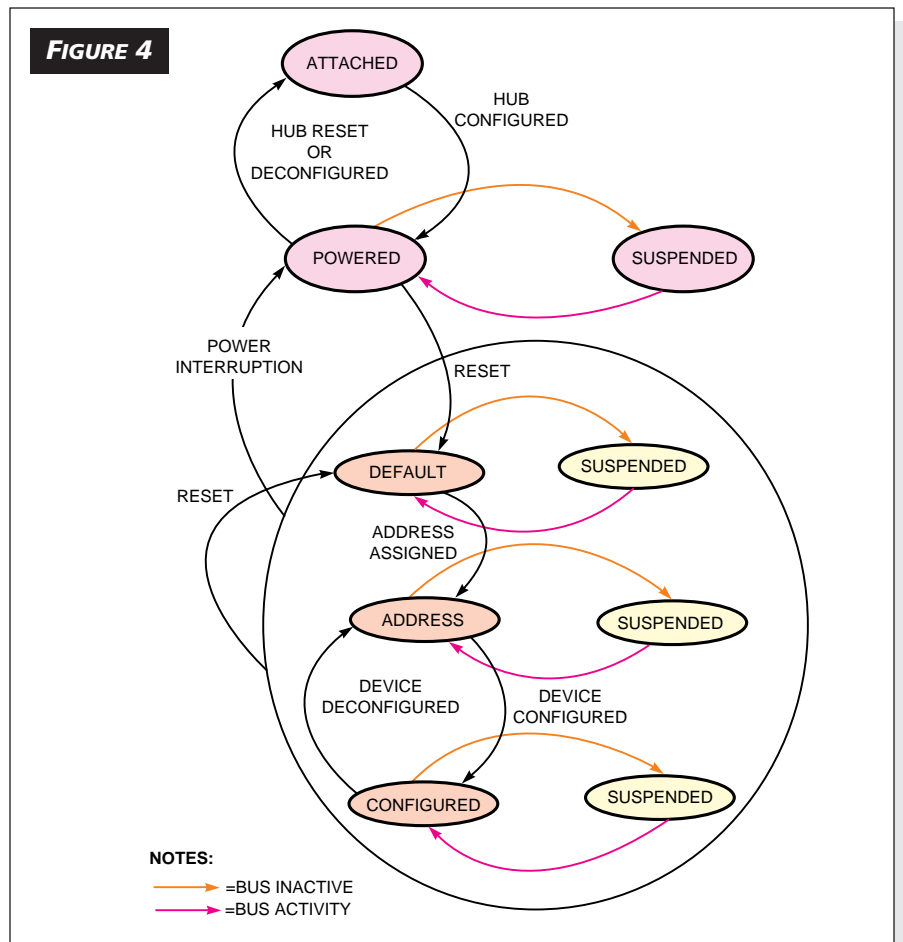
## USB HUBS

initializing, controlling, and communicating with as many as 127 diverse peripherals, extensive simulation of varied conditions and events is necessary to verify a USB design's correctness. In fact, because the number of possible conditions and events is so large, it's advisable to simulate conditions even beyond the operating boundaries of the USB specification. USB protocol analyzers and programmable traffic generators are vital pieces of equipment for this effort.

Consider just how complex USB activity is. Because devices can attach to or detach from the USB at any time, device enumeration for the bus is constant. It includes detection and processing of both additions and removals. Hubs indicate the attachment or removal of a USB device by reporting per-port status to the host and by identifying the port used to attach the device. The host enables the port and addresses the USB device with a control pipe, using the USB default address. The host then determines if the newly attached USB device is a hub or a function (a device that can transmit or receive data or control information over the bus) and assigns a unique USB address to the USB device. Next, the host establishes a control "pipe" (USB terminology for a data-transfer mechanism from a source or destination on a host to an endpoint on a device). This control pipe for the USB device uses the assigned USB address and endpoint zero. If the attached USB device is a hub and if USB devices are attached to its ports, then the above procedure repeats for each attached USB device. If the attached USB device is a function, then USB software dispatches attachment notifications to interested host software.

The USB specification also defines states that can exist for attached downstream hub ports (**Figure 4**). These states include:

- Powered off—a power-switched port that is in the off condition. All its downstream ports are also off, and the port supplies no power downstream.
- Disconnected—a port that can detect a connect event but cannot propagate any signaling. A connect event drives the port to the disabled state.
- Disabled—a port that can propagate downstream-directed signaling but not upstream.
- Enabled—a port that propagates all downstream and upstream signaling, at both full and low speed.
- Suspend—state when the hub selectively suspends all devices downstream of the port. Ongoing transactions



**The many device states that can exist in USB hubs necessitate thorough testing.**

complete before entering the suspend state; any downstream or upstream device can awaken a suspended port. To ensure that a USB device will operate within this highly dynamic environment, many designers use dedicated USB protocol analyzers and programmable traffic generators for testing. An effective test system should give you dynamic access to USB's raw data traffic and, at the same time, allow for dynamic input of a variety of bus configurations and traffic levels.

### Test gear


A programmable traffic generator is vital to any USB test strategy, because you need to inject many "illegal" conditions to push the hub design beyond specification limits. You can't typically generate these illegal, beyond-specification conditions simply by attaching hubs to a standard USB system.

USB protocol-analyzer test systems typically use a standard 386/486 Pentium-class PC to create a full USB test station, adding a nonintrusive USB probe and a software-analysis program. By monitoring the two USB data wires (D+ and D-), the test system re-creates the raw NRZI bit stream and

*continued on page 129*

## USB HUBS

stores it on disk for analysis. The software then scans the sampled data and lets you view it as continuous data or as complete data transactions. In a continuous mode, the analyzer fills the whole display line with data; in a data-transaction mode, one bus transaction—such as token, data, or handshake—goes on each line. The analysis program lets you look at specific events on the bus and search for and highlight error conditions, such as bad CRC, bad stuffing bits, and missing frames. You can also get a statistical analysis of overall bus use, and you can print or store analysis data for subsequent comparisons.

Another important step in preparing any USB device for release is putting it through a series of practical, real-world tests in a USB Compliance Workshop, held regularly by the USB Implementers Forum (**Reference 1**). These sessions, popularly known as “plug fests,” let you exercise a device with a variety of other USB systems, hubs, and devices. In addition to testing final, production-ready versions at these sessions, many designers also bring early prototype designs to help guide overall design development. 

---

## Reference

1. Universal Serial Bus Specification, Revision 1.0, USB Implementers Forum, Hillsboro, OR. 1-503-264-0590, fax 1-503-693-7975, [www.usb.org](http://www.usb.org).

---

## Authors' biographies



*Harry Inia is the marketing manager for CRT monitor products at National Semiconductor (Santa Clara, CA), where he has worked for 12 years. He holds a BscEE from the University of the Netherlands (Leeuwarden) and an MBA from San Jose State University (San Jose, CA).*

*Doug Dyment, PhD, is the director of technical marketing at Computer Access Technology Corp (Santa Clara, CA). Formerly, he was a professor and associate chairman of the department of computer science at the University of Waterloo (ON, Canada).*