2

USB Hosts for Embedded Systems

Developers have many choices for hardware and programming for USB embedded host systems. This chapter will help you choose a platform that has the USB host capabilities your project needs.

Embedded Hosts are Different

Because embedded systems typically support a limited number of peripheral types, most USB hosts in embedded systems don't need the full capabilities of a conventional USB host. At the same time, some USB hosts in embedded systems need capabilities that conventional hosts don't have, such as the option to turn off bus power when the bus is idle.

Dedicated Functions

A conventional PC's function and attached peripherals vary with the applications that users install and run. A PC in a science lab might connect to a variety of lab instruments, while a home PC might need to support the latest game controller. USB hosts in conventional PCs must support the wide variety of devices that users might attach.

To do so, the host supports multiple bus speeds and external hubs. Each host port can provide 500 mA (900 mA for SuperSpeed) to an attached device. The operating system (OS) provides drivers for popular USB device classes, and users can load drivers for additional devices as needed.

In contrast, embedded systems have defined functions. The firmware programmed into the system determines the system's function and the number and types of supported USB devices. These devices in turn determine what speeds the host must support, whether the host needs to support external hubs, and how much current the host port(s) must provide. Adding support for new devices typically requires a firmware update.

Some embedded systems provide both USB host and USB device functions. These systems may have a dedicated port for each function or a single dual-role port that can serve as both a host and device port, swapping roles as needed.

The Targeted Peripheral List

To reduce user confusion and frustration, a USB embedded host system can provide a Targeted Peripheral List that names devices that are known to work with the system. For example, a vendor might list manufacturers and model numbers of tested printers. Other printer models may also work, but the list enables users to rely on known good peripherals.

The USB-IF's *On-The-Go and Embedded Host Supplement to the USB Revision 2.0 Specification* defines requirements for USB host systems that provide a Targeted Peripheral List. The specification calls these systems Targeted Hosts.

Two types of Targeted Hosts are Embedded Host and On-The-Go (OTG) systems (Figure 2-1).

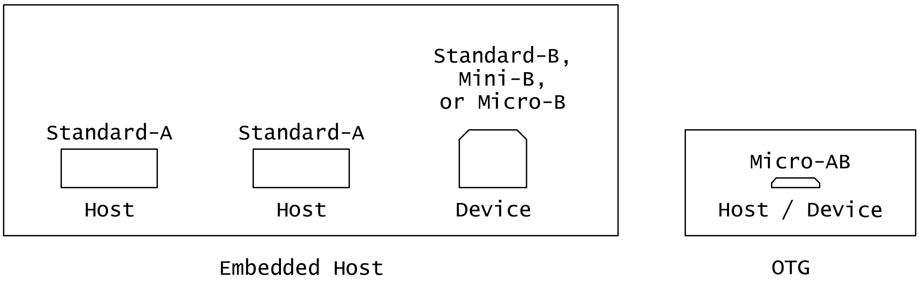

Figure 2-1. An Embedded Host system can have multiple USB ports, while an OTG system has a single dual-role port.

An Embedded Host system has one or more host ports and may also have a device port. An OTG system has a single port that can function as both a host and device port. Some requirements are relaxed for Targeted Host systems, and OTG systems have added responsibilities for managing the dual-role port.

On attachment of an unsupported peripheral, including a hub on a system that doesn't support hubs, a Targeted Host system shouldn't fail silently but should provide a message or other indicator to inform the user that the host doesn't support the device.

This chapter focuses on Embedded Hosts and the host capabilities of OTG systems. Chapter 11 has more about using the unique capabilities of OTG ports.

Requirements

The ports in an Embedded Host system function much like ports in conventional PCs but without the need to support the bus speeds and bus currents that the targeted peripherals don't use. Table 2-1 compares the requirements for Embedded Host ports and conventional host ports.

Capability or Feature	Conventional Host	USB 2.0 Embedded Host
Communicate at high speed	Yes	Must support all devices on the Targeted Peripheral List. May support high, full, and low speeds; high and full speeds; full and low speeds; full speed only; or low speed only.
Communicate at full speed	Yes	
Communicate at low speed	Yes	
Support external hubs	Yes	Optional
Provide Targeted Peripheral List	No	Yes
Minimum available bus current per port	500 mA (100 mA if battery-powered)	8 mA or the amount needed by targeted peripherals, whichever is greater
OK to turn off VBUS when unneeded?	No	Yes
Connector	1 or more Standard-A receptacles	1 or more Standard-A receptacles

Table 2-1: USB 2.0 embedded hosts have different requirements compared to conventional USB 2.0 hosts.

An Embedded Host system can support just about any combination of speeds needed for the targeted peripherals. If all of the targeted peripherals use low speed or all use full speed, the system needs to support only one speed. A system that supports high speed must also support full speed. All host ports should support the same speeds and devices. The *On-The-Go and Embedded Host* supplement applies to the USB 2.0 specification and thus offers no specific guidance for USB embedded hosts on SuperSpeed systems.

Switching Off Bus Power

To lengthen battery life, embedded systems that use battery power typically conserve power when possible. Unlike conventional USB hosts, Embedded Host systems have the option to turn off Vbus to save power when the bus is idle.

When Vbus is off, the host needs a way to detect device attachment, and already attached devices need a way to signal that they want to communicate on the bus. Two protocols, the Attach Detection Protocol and the Session Request Protocol, meet these needs.

Attach Detection Protocol

Conventional hosts detect device attachment by monitoring for a voltage change on the D+ or D- data line. But the USB 2.0 specification forbids devices from powering the pull-up resistor on D+ or D- when Vbus is absent except to do data-line pulsing for the Session Request Protocol as described below. The Attach Detection Protocol (ADP) provides a way for a host to detect device attachment when Vbus is absent.

An Embedded Host or OTG system performs ADP probing by discharging the Vbus line, then measuring the time required for a known current to charge the line to a known voltage. If the line doesn't charge within the expected time, no device is present. The probing repeats about every 1.75 s. Host support for ADP is optional. Hubs don't support ADP probing, so if a hub lies between the host and device, the host can't use ADP probing.

Session Request Protocol

If the host has turned off Vbus, a device can use the Session Request Protocol (SRP) to request restoring Vbus.

A device requests the host to restore Vbus by performing data-line pulsing, which consists of switching in the pull-up on D+ (for full and high speed) or D- (for low speed) for 5–10 ms. The host detects the voltage. Hubs don't recognize SRP signaling, so if a hub lies between the host and device, the device can't use SRP.

An Embedded Host or OTG system that ever turns off Vbus with a series-A plug inserted must support SRP.

Functioning as a USB Device

An embedded system with USB host support can also provide a device port and function as a USB device. For example, a data logger might have a host port that connects to a drive for saving data and a device port that connects to a PC for uploading data. Unlike OTG systems, which can perform only one function at a time, a system with conventional host and device ports can function as a host and device at the same time.

Necessary Hardware

To function as a USB host, an embedded system must have a system processor, a USB host controller, a root hub, one or more host receptacles, and a power source (Figure 2-2).

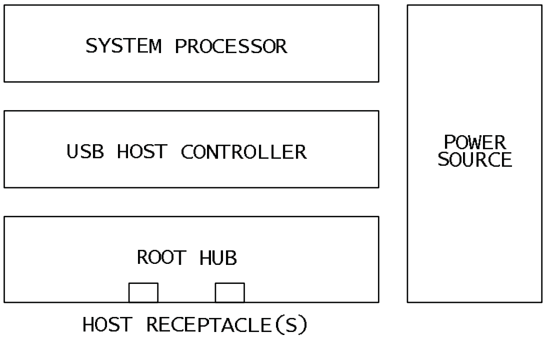

Figure 2-2. A USB host consists of a processor, USB host controller, root hub, one or more USB host receptacles, and a power source.

System Processor

The system processor is a microcontroller or other processor chip that executes the system's firmware. As explained later in this chapter, the choice of processor depends in part on the needed performance for USB communications.

USB Host Controller

The host-controller hardware includes electrical interfaces for one or more host ports and logic to implement low-level host protocols. The hardware can be in the system-processor chip or on a dedicated chip that interfaces to the system processor.

The electrical interface is one or more transceivers (for USB 2.0) or transmitters and receivers (for SuperSpeed) that interface to connectors.

The host controller's internal logic handles functions such as generating transactions to send data provided by the system onto the bus, making data received in transactions available to the system, error checking, and other tasks detailed in Chapter 8 of the USB 2.0 or USB 3.0 specification.

Root Hub

The root hub provides an interface between the host controller and its port(s). A root hub performs the same functions as external USB hubs, but the interface to the host controller is specific to the host hardware.

Host Connectors

Ports that function only as hosts ports use the same Standard-A receptacles that PCs use (Figure 2-3).

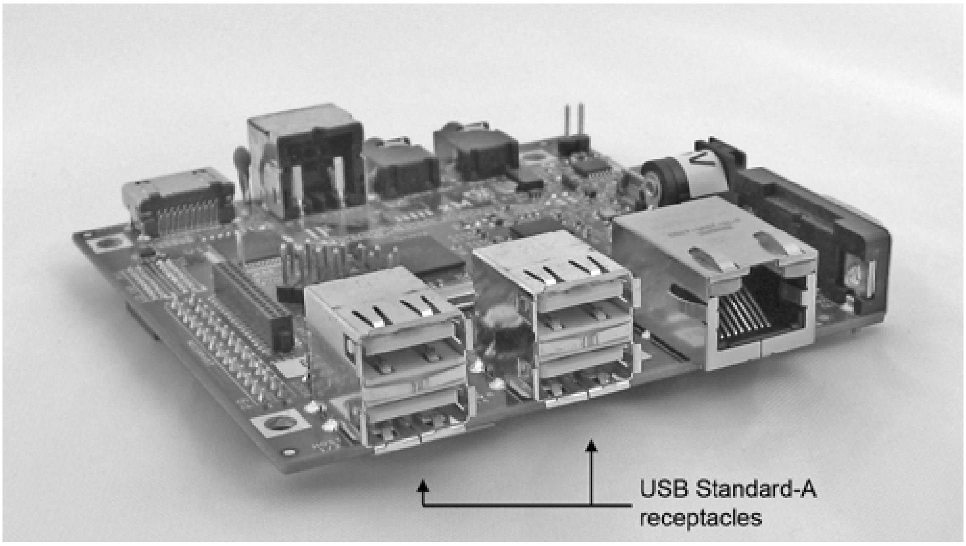

Figure 2-3. The BeagleBoard-xM has four Standard-A host receptacles.

A system can have one or more host ports.

Designers of products that have both Standard-A (host) and type-B (device) receptacles should use product design, labeling, and product literature to inform users of the product's function. In particular, the product's design and labeling should make it clear that the product isn't a hub.

Source of Bus Current

The USB host provides a nominal +5V to all devices that attach directly to the host ports. Embedded Host ports must be capable of providing 8 mA or the amount of bus current the supported devices need, whichever is greater. Devices that attach to an external hub receive power from the hub. A standard hub can supply 100 mA per port if bus powered (150 mA for SuperSpeed) or 500 mA per port if self powered (900 mA for SuperSpeed).

What the Host Does

The motivation for developing the USB interface was to make it easy to use peripherals of all kinds on conventional PCs. To keep the cost of devices low, the host is responsible for managing the bus, including scheduling traffic and providing and managing power. Devices just need to respond to communications and other events initiated by the host and upstream hubs.

On some USB embedded host platforms, the OS or a host module handles many of the USB host functions. On other platforms, the developer must provide firmware for these tasks.

Detecting and Enumerating Devices

A host or hub detects an attached device by monitoring the voltage on the data lines and optionally by using the Attach Detection Protocol as described above. Hubs use control and interrupt transfers to inform the host of newly attached devices. On detecting a device, the host attempts to enumerate the device and assign a driver.

Supporting External Hubs

A USB embedded host can support external hubs or require all devices to attach directly to a host port. A host that supports hubs can support the hub class, including providing 500 mA to bus-powered hubs and supporting five tiers of hubs (Figure 2-4), or the host can support specific hub models.

Managing Traffic

The host schedules traffic on the bus, reserving time for endpoints that have guaranteed bandwidth and scheduling other traffic in the time that remains.

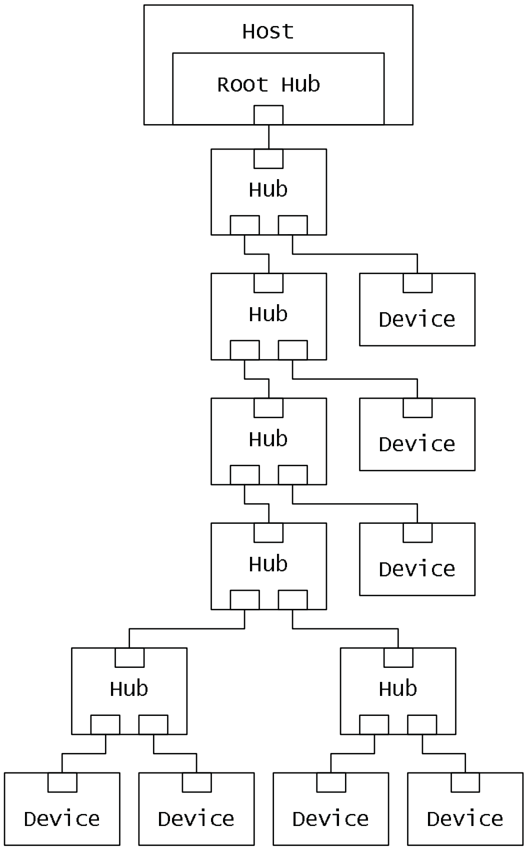

Figure 2-4. With support for the hub class, a host can have five tiers of external hubs.

Managing Power

During enumeration, a device's configuration descriptor requests bus current from the host. If the requested current isn't available, the host refuses to configure the device. To conserve power when the bus has no traffic, a host can use bus-signaling protocols to request devices to enter the Suspend state and reduce their use of bus current. When in the Suspend state, devices can use the remote wakeup protocol to request communications with the host. Embedded Host systems that are supplying bus power can remove power from the bus when the bus is idle.

Communicating with Devices

The ultimate purpose of a USB host is to exchange data with devices. The devices may belong to USB classes or use vendor-defined protocols.

Choosing a Development Platform

Because of the host's many responsibilities, adding USB host capability to a small embedded system may seem like a daunting task. Fortunately, a variety of hardware and programming platforms can ease the way.

Comparing Options

Host hardware and software are available for just about any need. Systems that need capabilities comparable to conventional PCs can use high-end processors targeted to embedded applications. Cost-sensitive systems that need good performance can use mid-range microcontrollers with USB host support either on-chip or in an external controller. Where high performance isn't essential, even 8-bit microcontrollers can access USB devices by interfacing to host modules that manage USB protocols.

The amount and type of programming the developer needs to provide host communications varies widely depending on the host hardware and the amount and type of firmware support for host communications. If you need to write USB host code from the ground up, the Linux source code for USB host communications can provide a model.

Table 2-2 compares options for implementing an Embedded Host port in an embedded system.

System Type	Sources	Host Communications Support
Embedded PC with host controller	beagleboard.org, Digi International Inc., EMAC, Inc.	Linux or Windows API, other protocols supported by the OS and programming environment
General-purpose microcontroller with on-chip host controller	Cypress Semiconductor, Freescale Semiconductor Inc., Microchip Technology, Texas Instruments	Libraries from chip provider

Table 2-2: Many hardware options are available for implementing hosts in embedded systems. (Part 1 of 2)

System Type	Sources	Host Communications Support
External host interface chip plus general-purpose microcontroller	PLX Technology, Maxim Integrated Products, Inc., ST-Ericsson	Libraries from chip provider
Processor with on-chip host module	FTDI (Vinculum II)	Vendor-specific API
Host module with interface to external processor	FTDI (Vinculum II in Vincil mode)	Vendor-specific command set
Processor with USB host and support for .NET Micro Framework and USB host communications	GHI Electronics	.NET Micro Framework classes

Table 2-2: Many hardware options are available for implementing hosts in embedded systems. (Part 2 of 2)

Embedded PC

At its heart, USB is an interface for PCs, and PC OSes such as Linux and Windows have rich support for USB host communications. An embedded PC can take advantage of this built-in support by using a distribution or edition of a PC OS targeted to small systems.

In an embedded PC, applications can access devices in much the same way that applications access devices on conventional PCs. The OS manages enumeration and other low-level protocols and provides drivers for popular USB device classes.

With an embedded PC, you can use many of the same development tools you use when developing mainstream PC applications. You can buy boards with Linux or Windows installed, or you can install an OS on suitable hardware.

Sources for embedded PCs with Linux or Windows installed include Digi International Inc. and EMAC, Inc. Chapter 3 has more about a Linux USB Embedded Host system that uses the BeagleBoard-xM open development board.

General-purpose Processor

A general-purpose microcontroller or other processor with an on-chip host controller allows full control of the firmware with low per-unit cost. The down side is the effort needed to program host communications. Firmware typically manages device detecting and enumeration, communications down to the transaction level, and bus power.

Chip vendors often provide firmware libraries that implement basic host communications and provide a foundation for application programming.

Sources for microcontrollers and processors with on-chip host controllers include Cypress Semiconductor, Freescale Semiconductor Inc., Microchip Technology, and Texas Instruments.

A microcontroller or other processor that doesn't have an on-chip USB host controller can use an external host interface chip. For example, ST-Ericsson's ISP1763A host controller can use an 8- or 16-bit bus interface to a system processor. Other sources for host interface chips are Maxim Integrated Products, Inc. and PLX Technology.

Host Module

For projects that don't have the firmware resources to support USB protocols, a USB host module can be a solution. The module manages enumeration and low-level communications and supports commands or an API for accessing popular device types. FTDI's Vinculum II is a host module with built-in support for accessing drives, keyboards, and other devices.

The Vinculum II has an on-chip processor core that supports an API for accessing USB devices. FTDI provides a C compiler for the processor. Supported USB device classes includes mass storage, hub, HID, still image, and audio. The module can also communicate with FTDI's FT232x USB UART devices.

The Vinculum II also supports an alternate mode that can use an asynchronous serial (UART), SPI, or parallel interface to an external processor. The processor uses defined commands to exchange data with USB devices. The Vinculum II handles the USB protocols and communications. This mode emulates the first-generation Vinculum.

For more on how to use the Vinculum II, download the free ebook *Embedded USB Design By Example* by John Hyde from *usb-by-example.com*.

Processor with .NET Micro Framework

Microsoft's .NET Framework is a popular programming framework for Windows applications written in .NET languages such as Visual C#. If you want to program in Visual C# but don't need the capabilities of even an embedded edition of Windows, GHI Electronics offers a series of boards with a system processor, a USB host controller, and built-in support for the .NET Micro Framework with .NET classes that support USB host communications. You can program the boards using the same Visual Studio software used to develop applications for PCs.

For more about USB host programming with the .NET Micro Framework, see the *Beginners Guide to NETMF* and other documentation for the GHI Electronics NETMF library at *ghielectronics.com*.

A Word about Protocol Analyzers

When developing a USB host system, no matter what hardware and programming platform you use, a USB protocol analyzer will save you time and trouble. A hardware-based protocol analyzer captures traffic and events on a bus segment and sends the information to a PC for decoding and displaying.

With this type of analyzer, you don't need to guess, wonder, or set up other debugging tools to try to find out what is happening. You can see the contents of every packet the host and device send. You know if the host sent what you expected and how the device responded. Monitoring traffic on a conventional host can be a rich source of clues for developing and debugging host communications for embedded systems.

Hardware-based protocol analyzers are available from multiple sources in a range of prices.

Software-only analyzers, which run on the host system, can be useful though they can show only what the host's drivers see, not transaction-level data on the bus. Chapter 3 shows how to use the `usbmon` software analyzer on Linux systems.

Non-USB Alternatives

Perhaps it's heresy to say this in a book about USB hosts, but not every embedded system needs USB host capability. Microcontrollers and application processors typically have a variety of I/O interfaces available. Before you decide to add a USB host port, it makes sense to decide if your system really needs one.

Many microcontrollers have serial interfaces such as a UART, SPI, and I^{2}C, which can handle many tasks that require low to moderate throughput. For accessing a network, on-board Ethernet or wireless-network ports are alternatives. A system that needs to exchange data with PCs can use a USB device port with an appropriate driver for the desired task.

With that said, USB hosts are powerful and flexible system components, and peripherals with USB ports dominate the market. If your system needs to access peripherals, chances are good that a USB host is the way to go.