Designing USB Into Embedded Systems

Issues and Answers

Overview

The interoperability, performance, and ease-of-use of USB has led to its overwhelming acceptance in the marketplace. It is the interface of choice for all but the highest performance and lowest cost applications. This near-ubiquity has led many companies to investigate adding USB to existing embedded designs, thus providing compatibility with a wide variety of devices with a minimum of development effort.

However, the ease-of-use of USB from a consumer standpoint belies the serious complexity of the software and hardware design. Making poor decisions before beginning a USB design can lead to long development times, poor system performance, and non-existent long-term maintainability.

This paper attempts to detail some of the thoughts that should go into adding USB to a design. The topics covered, in order, are

- USB background
- Is USB the right solution?
- Hardware issues
- Software issues
- OTG
- Support
- An overview of the Oxford controller line

Note that most of the issues raised in this document are related to adding USB host functionality to a design, but the concepts may apply to a peripheral-only design as well.

A Quick USB Background

Below is a bullet list of the pertinant aspects of USB:

- Three data rates are supported – low speed (1.5Mbit/sec), full speed (12Mbit/sec), and high speed (480 Mbit/sec). All three can coexist in one network, as the high-speed protocol is backward-compatible. Some hardware controllers support low- and full-speed only, and others support all three.
- One host, many peripherals. Only one USB host can exist in a USB network; on the desktop, this is typically a PC. Multiple peripherals, such as keyboards, mice, cameras, MP3 players, and Bluetooth adapters, can be connected directly to the host, or chained via USB hubs.
- The host controls all USB operations, sending data to and requesting data and status from each peripheral as needed.
- The USB cabling can carry power as well as data, and low-power peripherals can draw their power from the bus itself. Devices with higher power needs (as well as the host device) will need to be powered externally.
- Large amounts of both bulk (asynchronous) and isochronous (video, audio) data can be accomodated on the bus at once.
- Hot pluggability, no pre-configuration. Devices can be added to and removed from the bus at any time, and the host controller will automatically detect their presence and configure them such that they can communicate with host driver software.
- The core USB data format is used for all device types. Specific types of devices (e.g. mice, cameras, GPS units) and the software drivers that support them are all layered on top of the same basic USB framework.
- The USB OTG (On-The-Go) extension provides for ports that can attach either host or peripheral devices, and can exchange host and peripheral roles without resetting.

As mentioned above, this flexibility and ease-of-use comes at a price, as the software and hardware needed to support a USB host implementation are quite complex.

Is USB The Right Solution?

This may seem like an odd question for an Oxford Semiconductor white paper, as we are in the business of selling USB controllers. However, there is nothing to be gained by encouraging companies to add USB when it isn't the right solution.

There are several things to consider before you assume that USB is the best solution:

- Given that there is a high cost of entry for USB, as far as hardware and software knowledge, is USB overkill for your connectivity needs?
- How do your intended uses map to the class driver model that exists for USB? Devices typically adhere to one of a set of classes, such as printer, mass storage, human-interface device, etc. In order to be truly interoperable with other vendors' products, your host or peripheral application needs to adhere to one of these class standards.

- How do your intended uses map to the software that is available for your operating system? Depending on the OS and USB software you intend to use, you may have to change how your product functions to meet constraints imposed by the software.
- What requirements do you have for performance, cost, time to market, PCB footprint, etc.? Can a USB solution realistically meet those needs?

If the answers to those questions support an embedded USB solution, read on. Hardware and software concerns are next.

Can Your Hardware Support USB?

The performance capabilities of USB can tax a medium-powered embedded platform. If your system's performance is already close to its limit, adding USB without adequate planning may make the overall system unacceptably slow. There are a number of hardware issues that should be considered before USB is added:

- Bus Access: Assuming you are not using a PCI bus (and no Oxford Semiconductor USB controllers have a PCI interface) you will need 8, 16, or 32 bits of address and data to connect a USB host controller. In order to move data efficiently, this memory space will need access times on the order of 50-90 nanoseconds.
- System Throughput: The maximum full speed data rate is 12 Mbit/second; realistically, a host port may be asked to move at most 8 Mbit/second. Multiple host ports on one host controller will increase that load. All of that data will be moving to or from system memory. Embedded host controllers frequently have DMA, but not PCI (or other bus-mastering capabilities). Requirements for high speed USB applications could be anywhere from 10 Mbit/second to 150+ Mbit/second, depending on the devices you intend to support.
- Interrupt Load: Full-speed and low-speed USB have frame intervals of 1 millisecond (ms). In the worst case scenario, your CPU will be taking an interrupt every frame, and will have to copy data to and from the USB controller on each of those interrupts. The amount of data is likely to be small (probably no more than 64 bytes per USB device) but if the interrupt latency is a problem, this will have a serious performance impact. High-speed USB uses a microframe interval of 125 microseconds, but high-speed controllers can typically be configured to prevent interrupting at that great a rate.
- DMA: Many USB controller vendors trumpet DMA as a performance boost, but in reality, the gains here can be marginal. In non-busmastering USB controllers, data is typically moved in blocks that correspond to the USB Max Packet Size (64 or 512 bytes). For each such transfer, the setup and completion-interrupt times for DMA are a significant portion of the time. Also, it is not a given that the CPU can accomplish real work while a short DMA burst is happening.
- Memory: In these days of gigabit memory parts in cell phones, this is becoming less important. If the hardware cost is paramount, however, the memory needs of a USB stack and its data buffers will matter. A robust USB stack may take 30-100 KB of code storage, and 10-100 KB of runtime memory.
- PCB Footprint: Embedded USB host controllers are typically in 80-128 pin packages, depending on package type and bus width. In addition, the PCB layout requirements are strict, especially for high-speed applications.

USB Software Needs

As mentioned above, the software that enables USB is not trivial. We encounter prospective customers who envision their software teams writing their own USB stack and class drivers, and that is not realistic for the vast majority of cases. You may be able to cobble together something that works most of the time with a few specific devices, but if you value robustness, interoperatibility and scalablility, a USB stack is not a do-it-yourself project.

There are three major situations when it comes to USB software stacks – RTOS environments that include a native USB stack, RTOSs that do not, and designs with no RTOS at all.

Operating systems that have a native USB stack.

Linux and Microsoft's WinCE are the two most common RTOS environments that include a full USB stack. The USB capabilities depend on the version of OS, however:

- WinCE 4.2 and Linux 2.4.x have native host stacks, but not native peripheral stacks.
- WinCE 5.0 and Linux 2.6.x include native host and peripheral stacks.

If USB silicon vendors provide driver support for WinCE and Linux, they will typically do so one of two ways:

- The interface to their hardware adheres to a common standard, such as OHCI or EHCI. In this case, the drivers that ship with CE or Linux can be used unchanged.
- The vendor provides a small software interface layer that maps the OS function requests into the format needed for the controller.

Oxford Semiconductor chose the latter route, since adhering to the OHCI or EHCI interfaces was too restrictive for our needs. If you use an Oxford USB controller, we provide a Linux or WinCE "tie-in" layer that connects between the USB stack of the OS and our OS-independent controller driver code. Oxford Semiconductor's tie-in software and low-level drivers are free of charge for designs that use Oxford Semiconductor USB controllers.

Operating systems that do not have a native USB stack.

Many commercial RTOSs do not have a native USB stack available as an option. Third party vendors may provide stacks for some USB controllers, but combining three different companies (for the RTOS, USB stack, and USB controller) may give you pause. Ideally, the vendor of your USB controller can provide a USB stack for your software environment as well. They may provide this free of charge (which is rare) or the licensing costs may be significant.

Oxford Semiconductor's USBLink software stack is full-featured, RTOS-neutral, and has been ported to more than a dozen different commercial RTOS environments. There is a license fee associated with USBLink, and the fee will vary based on controller purchases, the class drivers desired, and the level of support required to implement the customer's solution. The fee may or may not include a per-product royalty, depending on the engagement.

What if there is no operating system at all?

USB peripherals, since they only respond to commands from the host, can typically run with little or no underlying OS support.

The USB host protocol, on the other hand, lends itself well to a multi-threaded environment. The process of enumerating a device involves handling asynchronous events and generating command packets independent of code that wants to use the device. The infrastructure provided by an RTOS (memory allocation, semaphores, message passing) is very useful for making sure data movement operations do not collide with device insertion and extraction. It is possible to implement a basic USB host stack with no RTOS, but if multiple devices are to be supported simultaneously, it is not recommended.

Does the software solution meet your needs?

Regardless of which OS route you take, you will, in general, be limited to the class drivers provided with the USB stack. It is rare for third party companies to provide robust class drivers that are layered on top of someone else's native USB stack. For example, WinCE includes a Mass-Storage host class driver, but if it does not support the latest version of the DVD drive that you are using, there is little recourse. When you look at your intended software solution, make sure it both serves your needs now, and can meet your expected plans in the future.

Is the USB solution forward-looking?

It is rare that a company will add USB to only one product - there will be other products in the same family, or new products with different functionality. It is a major savings if you can leverage your USB experience from one project onto multiple future projects. There are four aspects to consider here:

- Can your USB hardware vendor provide a future roadmap? You shouldn't have to change vendors in order to move to high speed or OTG.
- Does the software work on a wide variety of hardware platforms? If your new product uses a different CPU, how much has to change to move to that new architecture? Ideally, the OS and USB stack software are written to be portable, and bus- and endian-independent, so little has to change to move to new hardware.
- Do you have to re-invent your software if you change USB controllers? Ideally, changing to a different USB controller is as simple as swapping only the low level driver – all stack and application interface software is unchanged.
- If you want to add class drivers to your host solution, how difficult is it? Do the class drivers have a consistent programming interface, so there is no learning curve when new functionality is added?

A Note About OTG

The term "OTG" is used in a variety of places when describing USB devices, and not all of the uses are technically correct. In order to conform to the OTG specification, an OTG device must be a USB 2.0-compliant peripheral, have limited host capabilities, and support Session Request Protocol and Host Negotiation Protocol. SRP allows a suspended peripheral to request a host session, while HNP allows a peripheral to request a role exchange with an attached host device.

SRP and HNP, however, are complex to implement, and are frequently overkill for what consumers really need. In practice, there are few devices on the market that implement and use full OTG. Most use cases that claim to need full OTG can actually be satisfied by what is referred to as "ID pin detect." In this case, a device uses a Mini-AB receptacle, and either a host or peripheral can be attached. At attachment time, the device checks the state of the ID pin and

determines whether to act as a host or peripheral. It maintains that role until the attached device is disconnected. The hardware and software to implement this are much simpler.

All Oxford Semiconductor USB controllers that support OTG can be used in ID-pin-detect mode.

Support, the Forgotten Child

If your organization has no history with USB, it can be a large effort to get up to speed on the protocol. It is one thing to read the specification, but quite another to be able to analyze problems or improve stack performance. When you choose your intended USB solution, ask yourself how much effort you want to put into educating your staff. RTOS solutions that include a native USB stack may offer no help when that stack doesn't work in your environment. The cost benefit of licensing a USB stack and obtaining dedicated, experienced support might easily outweigh weeks of development time spent on debugging. A support agreement should also ensure quick delivery of bug fixes (and feature upgrades) that have been thoroughly tested.

Oxford Semiconductor's USB Controller Family

Below is an overview of the Oxford Semiconductor embedded USB controller family. If you have further questions or wish to speak with sales or engineering contacts, please see our web site at http://www.oxsemi.com.

Controller	Ports	High Speed	OTG	Bus	Pins/Package
UHC124	4 host	N	N	8 bit	64 LQFP
TD122	2 host	N	N	8 bit	64 LQFP
TD242	H/H or H/P or H/OTG	N	Y	16 bit	64 LQFP or 64 BGA
TD1120	H/H/P or H/OTG	Y, peripheral	Y	16 bit	100 LQFP or 84 BGA
OXU121HP	H/H/P or H/OTG	Y, peripheral	Y	16 bit	100 LQFP or 84 BGA
OXU140CM	H/H/P or H/OTG (*)	Y, peripheral	Y	16 bit	128 LQFP or 100 BGA
OXU200	1 peripheral	Y	N	16 bit	100 LQFP or 64 BGA
OXU210HP	H/H or H/P or H/OTG	Y, host and peripheral	Y	16 bit (BGA) or 16/32 bit (LQFP)	128 LQFP or 84 BGA

(*) The OXU140CM adds a built-in memory interface for direct access to SD, MMC, and CE-ATA devices.