USB Connectivity in a Battery Powered World

By Alf Petter Syvertsen, MCU 32-bit Product Manager, Silicon Labs

Introduction

Over the last decade, the universal serial bus (USB) has been adopted by designers of industrial and consumer devices as their interface of choice for enabling connectivity to other applications due to its ease-of-use, plug-and-play functionality and robustness. USB has achieved its primary goal of simplifying the way consumers control peripherals and transfer data. With more than three billion USB-enabled devices shipped into the market, USB is not only the fastest growing interface in consumer applications but has also achieved significant growth in industrial markets.

However, USB's ease-of-use, plug-and-play functionality and robustness do not come free for embedded solutions designers. For small, portable devices, adding USB as a communication interface at least doubles application current consumption and leads to devices that require much larger batteries than originally anticipated.

Upgrading from a traditional serial interface for communication to the popular USB interface often puts unfeasible restrictions on an energy budget. Often, a developer will have to choose between doubling the battery size and increasing device cost, which makes it less appealing, or cutting back on much-needed differentiating features. In this article, we discuss how the USB standard has evolved from the dream of standardizing all PC connections to a state-of-the-art technology that allows even small battery-powered devices to communicate with anything.

A Short History of USB

If you ever looked at the back of a PC in the late 90s, you would instantly recognize the proliferation of standards for connecting different types of hardware to your computer. There was the 5-pin DIN, PS/2, serial port, parallel port, maybe an SCSI port or two and, if you were a gamer, you would also have a game port on your soundcard.

The original developers of the USB recognized this and, in 1995, started to create one common machine-to-machine (M2M) standard that would supersede all others. In the late 90's, USB was being adopted, and, at first, it just added another connector to the mix. However, during the 2000s, it really started to proliferate, and, after a series of updates, it is now one of the most widely adopted M2M interfaces. The success of the USB standard can be seen by looking at your laptop or phone. Your phone has one connector, and it is USB. If your laptop was bought after 2010, it probably only has USB connectors in addition to the display and network connectors.

The USB standard separates the topology into devices and hosts. The host is the machine that initiates the communication and provides the power, and, on your desk, this is generally your computer. The device is the downstream thing that is connected to the host and simply replies to whatever the host asks for. On your desk, the mouse and keyboard are examples of USB devices.

The neat thing about a USB connector is that it also supplies power to the attached device, so there is no need for an external power supply to your mouse or external hard-drive. The USB standard specifies that the host deliver at least 100 mA of current to the device, and, if the device is lucky, it will have 500 mA available. This comes from the origin of the USB: PCs were always the host, and they were always connected to the wall. This requirement of the USB standard effectively stopped development of USBs for low power, as abundant power has always been available.

But what happens when this M2M interface finds itself in a battery-powered world? What is the impact when the host is also a portable device?

Impacts for Today's USB Hardware

In today's portable devices, a much-used term is 'power budget'. The power budget dictates how much energy the device can consume and is based on battery size and the required battery-life. For example, for an application that has a 250 mAh battery and needs a battery-life of two days (48 hours), the power budget will be approximately 5 mA. This power budget needs to be distributed across everything a developer wants the device to do, from sensor acquisition and processing to communication and display driving.

For the last two or three decades, as MCUs got smaller and batteries got better, we saw an explosion of handheld electronic devices ranging from handheld wind meters and oscilloscopes to digital breathalyzers and remote controls. However, with the introduction of Gigahertz, quad-core handheld smart-phones, we now see more of these devices being introduced as additions to the smart-phone since manufacturers no longer have to worry about processing power or user interfaces. This allows for the proliferation of cheap add-ons. Examples include the Kickstarter-backed Vaavud wind meter for smart phones and a breathalyzer that plugs into your iPhone. Both these applications use the HiJack interface, an ad-hoc interface that works on low-end devices but is far from optimal.

If you wanted to make something like this truly universal and user-friendly, you would have to move to a more suitable M2M-interface like USB. Choosing USB also allows the gadgets you design to be host-agnostic, meaning that it no longer matters if you are connecting to a Mac, a Windows phone or an Android tablet. Therefore, when we want to connect all of these extra gadgets via USB to our battery-powered everyday companions, what was never a concern in the original USB specs, power, suddenly becomes a top priority when choosing a USB-solution. You wouldn't want to design a simple addition to your phone that drained its battery!

So, by choosing the right hardware, you will be able to develop your device with a much smaller energy footprint since a universal M2M interface allows you to exclude almost all external components.

USB Technology for the Battery-Powered World

In order to understand how USB technology can be improved in terms of power consumption while retaining its ease of use and plug-and-play functionality, we first need to take a quick look at how USB communication works. In general, only the host can initiate transfers. Even if there is no communication, the host sends keep-alive messages to the device every millisecond. If the device has data available, it will reply with it. In this active mode, the device has up to 100 mA of power, and the host expects the device to provide an immediate response to any request. When the host stops sending these keep-alive messages for 3 ms, the device should enter a suspend state and immediately reduce its current draw below 3 mA.

In the suspend state, most of the device can be switched off, and, usually, we can switch off the most power-hungry parts of the PHY. Even though a 3 mA suspend current should be easily achievable by any modern microcontroller, there is no reason to keep it that high, and good microcontrollers with well-thought-out energy modes should be able to achieve less than 3 uA in this mode, including the current draw of the PHY.

However, in active mode, when inspecting the USB communication of a regular keyboard device, active is not very active; most of the time, the device is just waiting for the host to send data. However, whenever the host requests a response from the device, the response must be immediate; that is why most implementations keep the USB peripheral running at 48 MHz at all times to allow sufficient response time. In this particular example, the lines are idle for 97% of the time, even though we are enumerated and active.

A USB implementation designed for the battery-powered world takes this into account and determines exactly when the clock is needed, for how long it is needed, and what other parts of the USB can be shut off. Silicon Labs® now has two patents pending for such designs in which we have taken feedback from manufacturers and customers and made the USB interface truly usable in the battery-powered world. The same communication mentioned in the paragraph above is also

created with the USB-oscillators and the power-hungry part of the USB disabled between packets. When you use crystal-less USB oscillators, the power-hungry part of the USB is disabled between packets. This greatly reduces power consumption and creates a truly universal M2M interface, which also leads in terms of power-consumption.

Of course, this should implemented in such a way that it is invisible to both developer and end user, except for the significantly reduced power consumption, and when this technology is combined with other space- and cost-saving features like crystal-less USB implementations and clock recovery, a truly low power universal M2M interface is realized without the need for any external components.

Summary

In this article, we have seen how the USB interface has evolved from a simple desire to reduce the cable clutter of the desktop PC to becoming the de-facto standard of interfacing consumer devices. This proliferation of USB-enabled, portable devices has forced new design-requirements on integrated USB-peripherals.

New, intelligent USB hardware enables cost reduction and extended battery-life, and when combined with crystal-less USB technology, allows all devices to be smart, connected and energy friendly.

You can learn more about Silicon Labs' energy-friendly USB MCUs at www.silabs.com/happy-gecko.