



# Intel<sup>®</sup> Technology Journal

Technology with the Environment in Mind

## Making USB a More Energy-Efficient Interconnect

# Making USB a More Energy-Efficient Interconnect

Barnes Cooper, Mobile Platforms Group, Intel Corporation  
Paul Diefenbaugh, Corporate Technology Group, Intel Corporation  
Jim Kardach, Mobile Platforms Group, Intel Corporation

Index words: USB, low power, power friendly devices, platform power management

## ABSTRACT

The '64 Mustang is a classic: a car that people still talk and reminisce about 44 years on. Do you mess with success and change a winning formula? No, but you do update a design to fix weaknesses in the original (better audio, air-conditioning, reliability, etc.) and to address new consumer desires as the market changes (efficient engines to address fuel economy, catalytic converters to address the need for a cleaner environment, etc.).

The Universal Serial Bus (USB) is a classic of the computer world. It was introduced in 1996 and is now a ubiquitous computer interface. When it was developed in the mid '90s it was targeted for mainstream computers of the time, optimized primarily for consumer ease of use and low device cost. Around 2002, USB 2.0 was introduced offering a performance bump to 480 Mbps; again optimized to meet similar criteria.

Although many characteristics of the USB are top-notch, its impact on *platform* power consumption has been downright abysmal. While power consumption was not an important criterion of its original design, the USB has become a defacto feature for battery-powered platforms where low power is key. In addition, global concerns over energy consumption and carbon emissions have made energy efficiency an important market requirement even for desktop and server systems [1]. Therefore, like the classic Mustang, it's time to overhaul the USB in a manner that preserves the goodness which has helped make it such a successful interconnect.

In this paper we first outline the USB power issues and look at their impact on mobile platforms. We then discuss ways of resolving these issues. Although the focus here is clearly on notebook systems, most issues and solutions apply to other systems as well.

## INTRODUCTION

To comprehend USB's power problems you first need to have a basic understanding of how it works. We won't try and make you an expert on USB architecture; rather, we will just provide enough detail so you can understand the fundamental problems and how the proposed fixes address these.

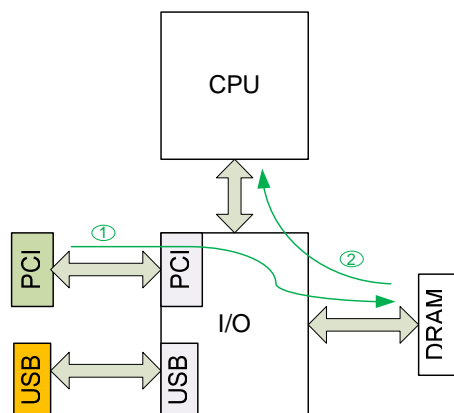
The root of most of the power issues is the fact that the USB is based on an architecture that constantly polls devices. Although this creates a simple and low-cost device model, it is fundamentally inefficient—especially when the device is idle or has little data to transfer. Specifically, a USB device is incapable of transferring data or generating an interrupt without being polled by the host. The best it can do is indicate the rate at which it wants to be polled in the event that activity occurs. This rate is typically assigned statically when the device is first configured and tuned for highly active phases (e.g., to maximize throughput).

We will go into a little more detail about how a USB device is designed to work in this polled environment and then discuss why polling creates power problems.

Figure 1 illustrates the behavior of normal (non-polled) data transfers for PCI devices. In this bus model, devices are generally implemented as fully capable bus masters. When a PCI device needs to transfer data it simply requests control of the bus and initiates one or more cycles to main memory (green line #1), which also results in a snoop cycle to the CPU (green line #2) to ensure data consistency in case the memory contents reside in the CPU cache.

Contrast this to the USB model where the device must wait until the next time it is polled by the host to transfer data, or more importantly, the host must continually poll a device just to see if it has data to transfer. The USB

provides two general models for data transfers: synchronous and asynchronous. Synchronous transfers are polled at a guaranteed periodic rate with a maximum frequency of once every microframe (125 microseconds). This corresponds to the Isochronous and Interrupt endpoint types. Conversely, asynchronous transfers are not polled at a guaranteed rate, but for most implementations this occurs quite frequently (many times per microframe) to achieve high data throughput when needed. Bulk and Control endpoints belong to this transfer type.



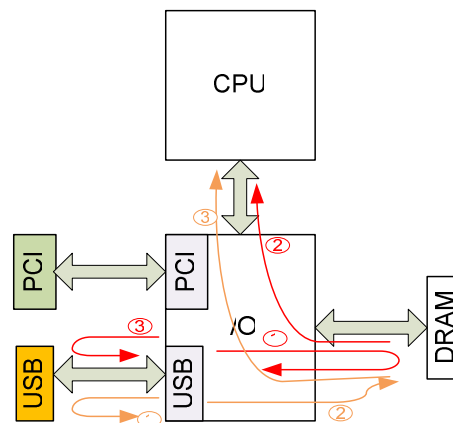
**Figure 1: PCI data transfer (non-polled)**

In addition, many USB host controllers rely on main memory for their schedule information. Data structures within the USB schedules inform the host controller of the (active) synchronous and asynchronous endpoints that need to be serviced, the polling frequency for synchronous endpoints, memory locations for data transfers, etc. The host controller must access these structures frequently, both to understand when endpoints need to be serviced (polled) and to initiate each transfer request—regardless of whether data are actually transferred.

Figure 2 illustrates the behavior for a typical USB Bulk IN transfer (read from device, write to main memory). The host controller first reads the transfer descriptor information from its schedule in main memory (red line #1), which in turn causes a snoop cycle to the CPU (red line #2) to maintain cache coherency. Once read, the host controller initiates the transfer to poll the targeted device (red line #3). If the device has no data to transfer it returns a NAK response (tan line #1). Otherwise, an ACK is returned along with whatever data the device needs to transfer (tan line #1), which the host controller then writes to main memory (tan line #2), and again causes a snoop cycle to the CPU (tan line #3).

USB transfers are inherently less efficient than equivalent PCI transfers, requiring a total of six cycles (three being snoops) versus two cycles (one snoop) on PCI. But the bigger issue is that USB endpoints that have no data to

move (constantly NAK) continue to be polled by the host resulting in a fairly active USB subsystem that generates frequent memory accesses, snoop cycles, and USB transfers. This behavior does not occur on PCI or other non-polled interconnects. Thus, USB works quite hard at doing nothing, which translates into poor energy efficiency.



**Figure 2: USB data transfers (polled)**

It is also important to notice the majority of power increases occurs upstream of the USB host controller. For example, a host controller polling a single Bulk IN endpoint can generate bursts of activity every 8-16 microseconds (us), which prevents most of the core logic (CPU, memory, backbone busses, clocking, etc.) from entering a low power state. This in turn can have a huge impact on platform idle power and drastically decrease battery life.

## Background on USB 2.0

The Universal Serial Bus 2.0 specification [2] is defined by the Universal Serial Bus Implementer's Forum, Inc. ([www.usb.org](http://www.usb.org)). It supersedes and is backwards-compatible with the USB 1.1 specification. USB 2.0 encompasses three distinct data rates: low-speed at 1.5 Mbps, full-speed at 12 Mbps, and high-speed at 480 Mbps. USB 2.0 uses a 4-pin bus with two differential signaling lines (D+/D-). Fundamentally, the USB 2.0 bus is a polled bus in that data and control transactions are initiated by the host, not the device. Because polling directly translates to increased power consumption across the platform, device design techniques are especially important. The USB 2.0 bus standard has a low power state known as Suspend, but today the latencies associated with entry and exit make it problematic to use as a dynamic flow control and link power management mechanism.

The USB 2.0 specification defines four distinct traffic classes (control, bulk, periodic, isochronous) and three data rates (low, full, high). This is typically managed on

Intel® Architecture (IA) platforms using two different host controller types: Enhanced Host Controller Interface (EHCI) for high-speed devices and Universal Host Controller Interface (UHCI) for low- and full-speed devices.

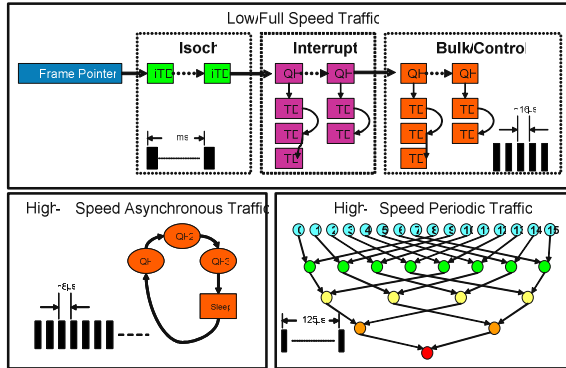


Figure 3: Host controller schedules

Figure 3 illustrates the various schedules, traffic classes, and data patterns for low and full-speed transactions associated with low-, full-, and high-speed devices. For low- and full-speed devices serviced by the UHCI controller, the host controller maintains a frame list pointer that references a physical address in main memory. The host controller parses this schedule every frame (1ms interval) to fetch memory structures (descriptors) that tell the host controller how to poll devices. The operating system (OS) software is responsible for populating the schedule. This specifies which transactions the host controller will attempt during each frame. In the Windows® OS, periodic transfers are layered first starting with isochronous endpoints that are allocated a fixed bandwidth. After this, the OS places interrupt endpoints that are generally polled at some derived periodicity, typically using a binary tree (poll rates of 1ms, 2ms, 4ms, 8ms, 16ms, 32ms, etc.). Bulk and control endpoints are added next and typically arranged as a linked list. The host controller typically parses the periodic elements once per frame, spending the rest of its time (until the next frame) processing bulk and control endpoints.

Table 1: General UHCI/EHCI power implications

	UHCI (Low/Full-Speed)		EHCI (High-Speed)	
	Periodic	Asynchronous	Periodic	Asynchronous
Bus Cycles	Every 1ms	Every ~16µs	Every 125µs	Every ~8µs
Power Impact	Low	Very high	High	Very high

The EHCI controller services USB 2.0 high-speed devices and contains two distinct schedules. The asynchronous schedule consists of bulk and control endpoints that are typically arranged as a linked list. The periodic schedule

contains isochronous and interrupt endpoints that are linked at a specific periodicity. The EHCI controller is capable of processing periodic transactions at an accelerated rate referred to as a microframe (125µs)—eight times more frequently than UHCI. Thus, periodic transfers may be scheduled at a maximum rate of once every microframe (125µs).

Table 1 summarizes the platform power implications when servicing low-, full-, and high-speed endpoints using traditional UHCI and EHCI host controller designs.

### Effect of USB Activity on System Power

When bus master traffic is generated by a USB host controller on an otherwise idle system, the platform will immediately transition out of a low power state to process this traffic. This flow is represented in Figure 4 which loosely depicts an Intel® Core™ 2 Duo mobile processor-based system.

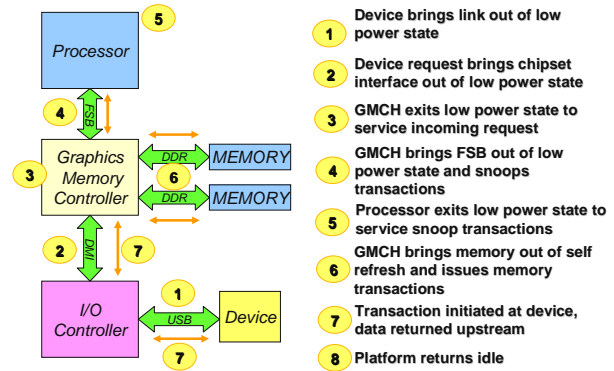


Figure 4: System power impact of USB activity

Because this activity is a platform-wide event, the resulting power impact can be large. Figure 5 illustrates a bus master transfer from a WLAN device fielding a keep-alive packet from an 802.11g access point. Although the actual transfer is short-lived, the component and platform power scales up dramatically to process this activity.

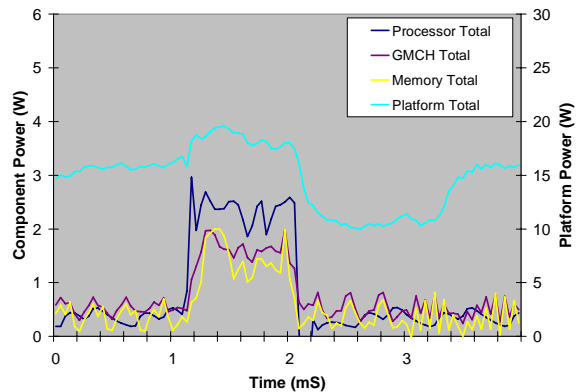


Figure 5: Platform power impact for WLAN activity

Thus, the general solution for addressing USB's power issues requires that we significantly reduce the amount of activity the host controller generates, especially when USB devices are otherwise idle (no data to transfer).

## ARCHITECTURE

Addressing the power issues associated with USB is a challenging one. Figure 6 depicts the high-level vision for a truly energy-efficient model for USB. The general idea is to keep the entire path from main memory through the host controller and down to the device completely quiescent until meaningful data needs to be transferred, thereby transforming today's continuously polled architecture to one where devices are only polled when needed.

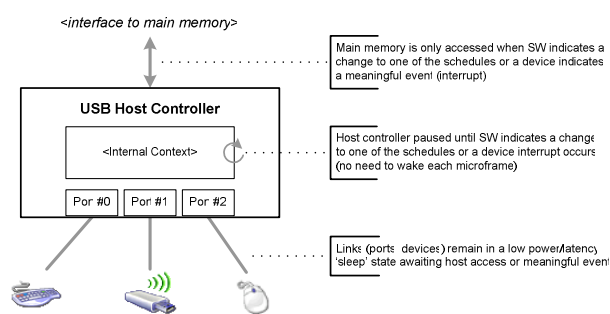


Figure 6: Energy-efficient USB vision

But in order to maintain compatibility with mainstream OSs it was important to avoid changes to the upper levels of the USB software stack. This focused the scope of our solution on the lower levels (miniport driver and hardware), as illustrated in Figure 7.

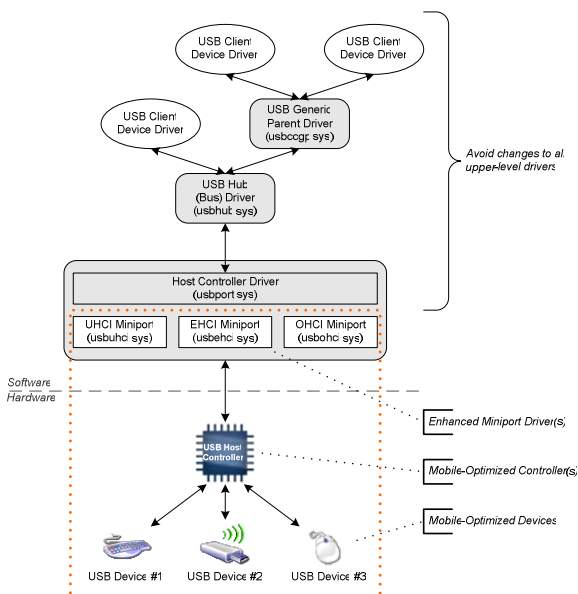


Figure 7: Design constraints

## Making USB Power Friendly

In the next few sections, we discuss various energy-efficiency optimizations based upon the following criteria:

1. If no devices are connected or no work is scheduled then USB hardware should remain in a low-power state.
2. Suppress host-side activity (upstream of the host controller, e.g., to main memory) when there is no meaningful work to do.
3. Suppress device-side activity (downstream of the host controller, on the USB bus) when there are no data to send/receive from devices.

## Miniport Drivers

Because of the polled architecture, the host controller's interaction with devices is very important, and if it is not done properly it can adversely affect platform power. In the architecture overview we talked about host controller schedules and how these are used to poll devices and to perform data transfers. Proper management of these schedules is absolutely necessary for producing a power-friendly USB subsystem.

For example, suppose no devices are attached to the system. Obviously a power-friendly USB software stack should schedule no work when there are no devices, and it should immediately remove all associated work from the schedules when a device is removed. If this sort of basic "schedule" and "controller" management is not performed well, any additional power-efficient enhancements will have limited impact. Thus, it is critically important to ensure the miniport drivers do effective work scheduling, turn controllers off when not used, and remove all associated descriptors from host controller schedules when devices are unplugged, disabled, or the work has completed.

Several critical changes were identified in Windows XP\* SP2 that have resulted in tremendous power savings. Intel worked with Microsoft engineers to develop these changes and make them available for both Windows XP and Windows Vista\*. This includes support for the UHCI run/stop bit, EHCI run/stop, and asynchronous/periodic schedule enable bits, as well as aggressive schedule idle detection. These software optimizations have in turn enabled other hardware optimizations, which we discuss in the next section.

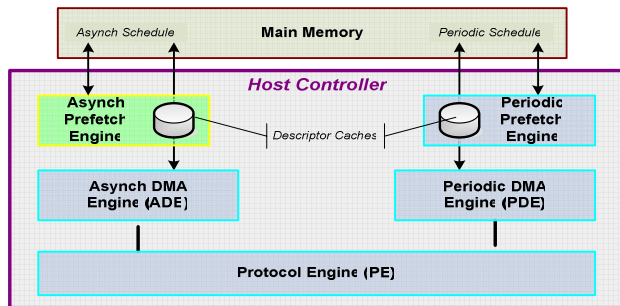
## Host Controllers

New features for Intel's mobile USB host controllers were identified to allow for power management opportunities when one or more schedules are enabled (endpoints

present and active). The enhancements include the following key concepts.

**Caching**

The Caching technique allows the host controller to store schedule information (descriptors) in controller-local memory in order to significantly reduce accesses to main memory, particularly in the case where devices are relatively idle. These data are typically stored in an abbreviated format where just enough information is provided to generate a transfer request (poll). Figure 8 illustrates this technique.



**Figure 8: Caching technique**

If the device NAKs the transaction, the host controller remains completely idle (since the information needed to generate the transaction was stored locally). If the device ACKs the transaction, the host controller typically must open the path to memory to move the actual data (to/from the device).

The caching feature is especially helpful for endpoints that observe a high NAK rate (for example, streaming or networking devices with bulk asynchronous endpoints open all of the time).

**Deferring and Link Power Management**

Although caching is fairly good at quiescing host-side activity it does nothing to address downstream (device-side) activity. The USB 2.0 Suspend state was originally intended for this purpose, but it is very difficult to use because of entry and exit latencies and other limitations. It takes considerable time to enter and exit this state (3ms + OS overhead for entry, 30ms + OS overhead for exit), and devices are severely limited in the amount of power they can consume while residing in this state. Additionally, Suspend is coupled with the device D3 state where the OS assumes hardware context is lost (and thus the device needs to be re-initialized and context restored upon exit), which adds significant latency and often interrupts device functionality.

The L1 state is a new Link Power Management (LPM) state that addresses the key deficiencies of the existing Suspend state (herein referred to as L2) by reducing state latencies and decoupling the link state from the device

state (allowing the device to remain in D0). The L1 state is intended to be used dynamically when the device is operational (D0), but otherwise idle, and able to quickly enter and exit this low power state without disrupting normal operation. Host controllers can safely negotiate L1 entry with idle devices, progressively decreasing downstream (device-side) activity until all devices reside in L1 (or L2), at which point no downstream activity will occur until either the host or device wakes the link to an active (L0) state.

The L1 transitions have significantly lower entry and exit latencies (10s of  $\mu$ s) than those of L2 (10s of ms). As with L2, both device- and host-initiated wake events are supported from the L1 state, noting that L1 device-initiated wake events play a prominent role in another key technique known as Deferring.

Supporting the L1 state requires modifications to both USB host controllers and devices. The L1 state is a new feature that augments USB 2.0 power management; it does not replace the existing L2 (suspend/resume) mechanism. The proposed L1 definition is backward compatible in that a new host can determine whether a device supports L1. A new device will continue to work properly with legacy hosts (obviously without L1 transitions), and old devices will continue to work on new host controllers. The only time L1 will be used is when a device acknowledges support for this feature on a new host controller.

The policy for using the L1 state is platform and implementation specific and will likely depend on the type of endpoint being served by the host controller. For periodic (interrupt or isochronous) transactions, the host controller would likely implement a policy whereby the device is immediately placed into the L1 state as shown in Figure 9.



**Figure 9: Example L1 policy for periodic devices**

For asynchronous (bulk or control) transactions, the host controller would likely implement a policy whereby the device is polled some number of microframes or frames at the nominal asynchronous poll rate before attempting to transition the device to L1, as shown in Figure 10. This is done in order to reduce the overhead for devices that stall for short periods between subsequent data phases.



**Figure 10: Example L1 policy Asynch devices**



keyboards, where the end-user may perceive the increased latency associated with L2 entry/exit (e.g., choppy mouse movement) when using these devices.

### Use LPM L1 Dynamically

As discussed previously, the long-term path to fully addressing the power efficiency limitation of USB 2.0 requires that the device and platform implement a new low-power link state known as LPM L1. For device implementations, it is important to note that entry into the L1 state should not result in any loss of functionality, as it is intended to be used while the system and device may be idle between bursts of activity. It is also important that the device pay attention to the Host Initiated Response Duration (HIRD) field in the host command sent to the device to request entry into the L1 state. This parameter is indicative of the depth of lower power state the platform is expecting to enter. If the platform is semi-active, the field may indicate a light response duration (e.g., <200us), whereas if the platform and devices are more deeply idle, the field may indicate a bigger number (~1ms). The device should use this parameter to control the depth of power management in use by the device to save power, for example, by shutting off PLLs only when a “long” (~1ms) L1 entry transaction is identified.

### Design True Composite Devices

The use of integrated hubs within multifunction devices has been a common practice to streamline and simplify hardware implementations. Although convenient, this approach has a number of power management pitfalls and is therefore strongly discouraged. For example, many Deferring scenarios are not feasible for devices that are attached to a downstream hub rather than directly to one of the host controller’s root ports.

The most energy-efficient designs involve true composite devices. Here multiple logical functions (devices) reside behind a single USB 2.0 physical device interface where each independent function is exposed as sets of one or more endpoints.

### Application/Driver Synchronization

Many devices such as streaming (media playback, cameras) or occasional use (fingerprint sensor, GPS) are bundled with application software. It is critical that when the application stream is shut down, care must be taken in the device function driver to ensure that the application properly cleans up driver requests on exit or inactivity (pause, mute, etc.) to avoid dangling transactions pending on the device; otherwise, these transactions remain un-serviced or are continually retried.

### Avoid Polling Integrated Buttons

Many devices such as integrated cameras support a so-called “Instant On Feature,” whereby the device has local

buttons that are typically serviced by a periodic interrupt endpoint. The buttons require a continuously running periodic interrupt endpoint to poll the button, and this wastes power. It is recommended that devices purposefully designed for mobile platforms do not support buttons (better to enable through applications or traditional keyboard hotkeys), or if they do support buttons that must be functional, you should work with the platform designer to provide platform-level notifications mechanisms through sideband signals and Advanced Configuration and Power Interface (ACPI) BIOS modifications. By using such a scheme, the notifications may be delivered on demand, and the function driver can be the target of these notifications providing the same net effect for Instant On Features without having to continuously run the periodic schedule.

If the button can’t be avoided, then architect a very long poll interval for the button (10s to 100s of milliseconds) to reduce the inevitable platform power impact. Such a long polling interval will give other hardware optimizations a chance to kick-in (Caching, Deferring, L1, etc.).

### Challenges

Clearly there were and are numerous challenges associated with making USB 2.0 an energy-efficient interconnect. We are quite pleased with the progress thus far, but note the biggest remaining challenge is the broad and timely adoption of these devices, OS, and platform features by the ecosystem.

### RESULTS

There are two main reasons why the USB needs to be overhauled: to reduce the power directly consumed by USB devices and host controllers, and (more importantly) to eliminate the drastic increase in power consumption that current USB behavior has on other platform components. The techniques described herein fully address both.

As an example, Figure 12 illustrates the total platform power savings opportunity for the *Caching* and *Deferring* techniques on an Intel® Core™2 Duo mobile processor-based system. The results were derived using measured data and best-known practices. Note that platform power increases by a whopping 5.7W when a high-speed bulk endpoint is active but constantly NAKing, as is the case for most wireless network devices. Here the *Caching* technique achieves a ~4.7W improvement by localizing activity to the USB host controller (EHCI), link (Port), and downstream device (WLAN), enabling the processor, GMCH, memory, and DMI interconnect to enter and remain in a low-power state. The *Deferring (LPM L1)* technique addresses most of the residual power by

allowing the controller, port, and device to only become active when necessary (no longer polled).

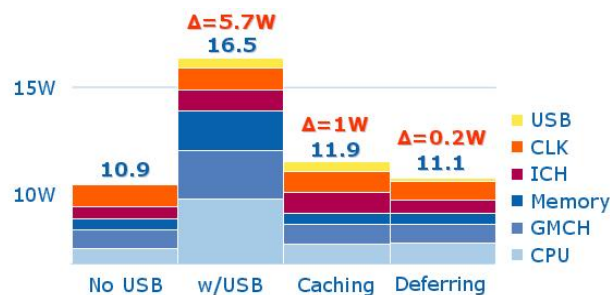


Figure 12: Resulting platform power savings

## DISCUSSION

We discussed a number of techniques to transform USB into an energy-efficient interconnect in this paper. These techniques are based on several basic principals for power-friendly design:

1. An efficient transfer rate (bandwidth per Watt) is important but should not be the only focus when designing an interconnect. Specifically, robust and low-power idle states are an absolute necessity.
2. Power management states should be defined such that these states can be used effectively across a variety of idle to pseudo-active scenarios.
3. It's all about platform power. Optimizing for low subsystem power while ignoring the subsystem's impact on the rest of the platform is a recipe for failure. Developers need to analyze both component and platform power consumption in order to catch unexpected behavior or other power-related artifacts. A poorly designed device or host controller may only consume tens of milliwatts but this can result in a multi-watt increase throughout the platform.
4. Good idle behavior is key. An idle device should burn (nearly) zero power; the same applies to buses and host controllers. "Do nothing efficiently!"

Although USB has always had a relatively efficient data transfer rate, this provided little advantage to the platform designer because of the interconnect's significant idle penalties.

Concerning a low-power idle state, the original Suspend state was intended to address a variety of usage cases, but high latencies and other characteristics have prevented its widespread use. And although certain flow control mechanisms do exist, these were designed to address platform performance (vs. power) concerns and failed to address the fundamental issues of constant polling and associated upstream and downstream activity. The new LPM L1 state fills this void.

The Caching technique addresses upstream (host-side) activity that has prevented much of the platform from residing in a low-power state even when all USB devices (and the rest of the system) are pervasively idle.

The Deferring technique addresses downstream (device-side) activity, where entry into L1 state is used as a means by which host controllers can safely defer polling when all endpoints for a device become idle. Here the host can resume the device (and thus polling of its endpoints) when it has meaningful data to transfer and vice versa.

The combination of techniques has transformed USB from a constantly polled architecture with frequent activity to one where activity occurs only when there are meaningful data to transfer, approaching the energy efficiency of other non-polled interconnects such as PCI.

## CONCLUSION

Like the original '64 Mustang, USB is a classic. Although it has been widely successful, the time has come for an "energy efficiency" overhaul. The key attributes that contributed to USB's success (simple, low cost, decent bandwidth) needed to be preserved while at the same time modernizing and enhancing this interconnect for today's environment where "green-ness" and "power efficiency" have become equally important.

We have demonstrated techniques and offered suggestions that transform USB into a much more energy-efficient interconnect, primarily by optimizing the idle behavior of USB host controllers and devices. This complements USB's relatively good bandwidth per watt characteristics to produce a robust and power-friendly solution.

## ACKNOWLEDGMENTS

We thank John Howard, Corporate Technology Group, Intel Corporation; Karthi Vadivelu, Client Components Group, Intel Corporation; and Michael Derr, Client Components Group, Intel Corporation.

## REFERENCES

- [1] "Energy Star\* System Implementation," April 2007, at <http://www.intel.com/design/core2duo/316478.pdf>
- [2] "Universal Serial Bus Revision 2.0 specification," April 27, 2000, USB SIG, at <http://www.usb.org/developers/docs/>
- [3] "Designing Power-Friendly Devices," May 8, 2007, WinHec 2007, at [http://download.intel.com/technology/EEP/designing\\_power\\_friendly\\_devices.pdf](http://download.intel.com/technology/EEP/designing_power_friendly_devices.pdf)

## AUTHORS' BIOGRAPHIES

**Barnes Cooper** is a Principal Engineer within the Intel Mobile Platforms Group (MPG) and has been working on mobile platform architecture and power/thermal management since 1992. He was a key contributor to numerous power/thermal innovations such as ACPI 1.0, Enhanced Intel SpeedStep® technology, and multi-core/thread power management. He works extensively with Microsoft engineers on defining OS power management innovations. His e-mail is barnes.cooper at intel.com.

**Paul Diefenbaugh** is an architect within the Intel Corporate Technology Group (CTG) leading efforts there on Platform Power Management. He holds an M.S. degree in Computer Science from Oregon Graduate Institute. Paul joined Intel in 1992 and has focused on mobile platforms and energy efficiency during the last ten years. He was a key contributor to the ACPI 2.0 specification, the Linux implementation of ACPI and OSPM, and various Intel power-management innovations addressing processors, interconnects, displays, storage, and platforms. His e-mail is paul.s.diefenbaugh at intel.com.

**Jim Kardach** is a Senior Principal Engineer within the Intel Mobile Product's Group (MPG). He joined Intel in 1986 where for the last 20 years he has worked on notebook processors and chipsets. Jim worked on developing technologies for the notebook platform (including the original ACPI specification). Jim has about 50 issued patents, was runner-up in the 1996 Discover Magazine's Innovation Award, and was one of the original inductees into the Bluetooth Hall of Fame for his work on that standard. His e-mail is jim.kardach at intel.com.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino logo, Core Inside, FlashFile, i960, InstantIP, Intel, Intel logo, Intel386, Intel486, Intel740, IntelDX2, IntelDX4, IntelSX2, Intel Core, Intel Inside, Intel Inside logo, Intel Leap ahead., Intel Leap ahead. logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, IPLink, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium Inside, skool, Sound Mark, The Journey Inside, VTune, Xeon, and Xeon Inside are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Intel's trademarks may be used publicly with permission only from Intel. Fair use of Intel's trademarks in advertising and promotion of Intel products requires proper acknowledgement.

\*Other names and brands may be claimed as the property of others.

SPEC®, SPECint® and SPECfp® are registered trademarks of the Standard Performance Evaluation Corporation. For more information on SPEC benchmarks, please see <http://www.spec.org>

Microsoft, Windows, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Bluetooth is a trademark owned by its proprietor and used by Intel Corporation under license.

Intel Corporation uses the Palm OS® Ready mark under license from Palm, Inc.

Copyright © 2008 Intel Corporation. All rights reserved.

This publication was downloaded from <http://www.intel.com>.

Additional legal notices at: <http://www.intel.com/sites/corporate/tradmarx.htm>.

**THIS PAGE INTENTIONALLY LEFT BLANK**

For further information visit:

[developer.intel.com/technology/itj/index.htm](http://developer.intel.com/technology/itj/index.htm)