
PCI
SIG™

PCI Express Advanced Software Topics

Davis Walker
Software Design Engineer
Microsoft Corporation

Prashant Sethi
Software Architect
Intel Corporation

Agenda

- Background
- PCI Express Configuration Regions
- Interrupts and MSI/MSI-X
- Break
- Power Management Events
- Power Negotiation
- Hot-Plug
- Summary
- Q and A

Agenda

- Background
- PCI Express Configuration Regions
- Interrupts and MSI/MSI-X
- Break
- Power Management Events
- Power Negotiation
- Hot-Plug
- Summary
- Q and A

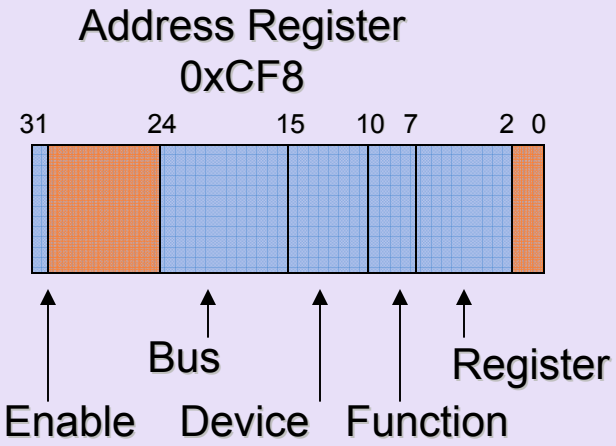
Background

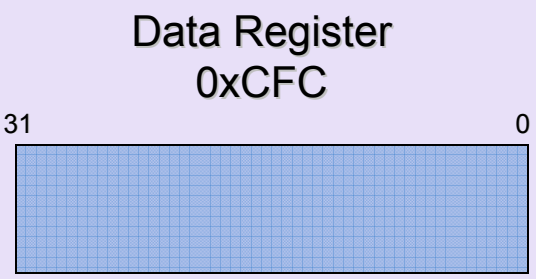
- PCI Express is designed to be software-compatible with PCI
  - ✓ Basic configuration registers are defined like PCI
  - ✓ Power management mechanisms use PCI-defined model
- PCI Express software model is an enhancement over PCI in two ways
  - ✓ New features were added
    - E.g., Active State Power Management
  - ✓ Improvements to existing features were made
    - E.g., PME
- Platform hardware and firmware abstracts some differences
  - ✓ Software-compatibility is made possible by special features in the platform

Agenda

- Background
- PCI Express Configuration Regions
- Interrupts and MSI/MSI-X
- Break
- Power Management Events
- Power Negotiation
- Hot-Plug
- Summary
- Q and A

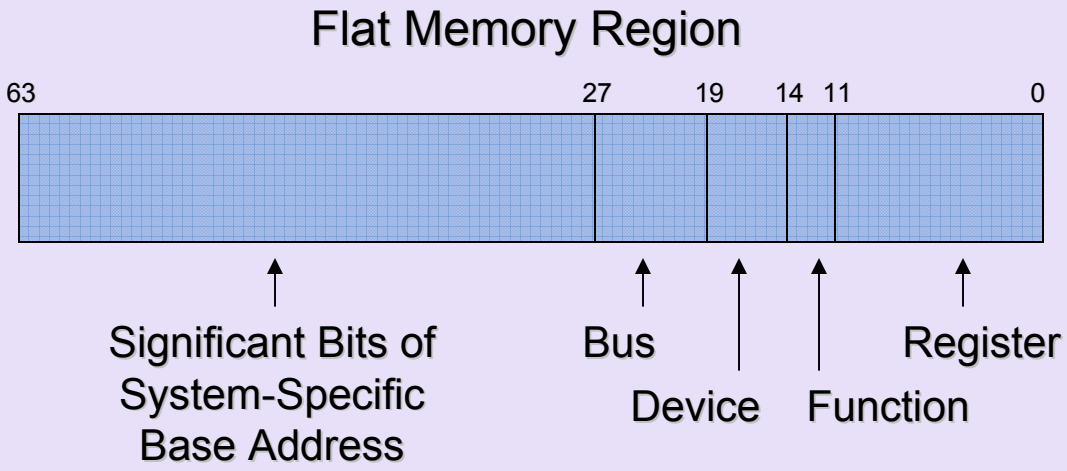
Legacy Access Model





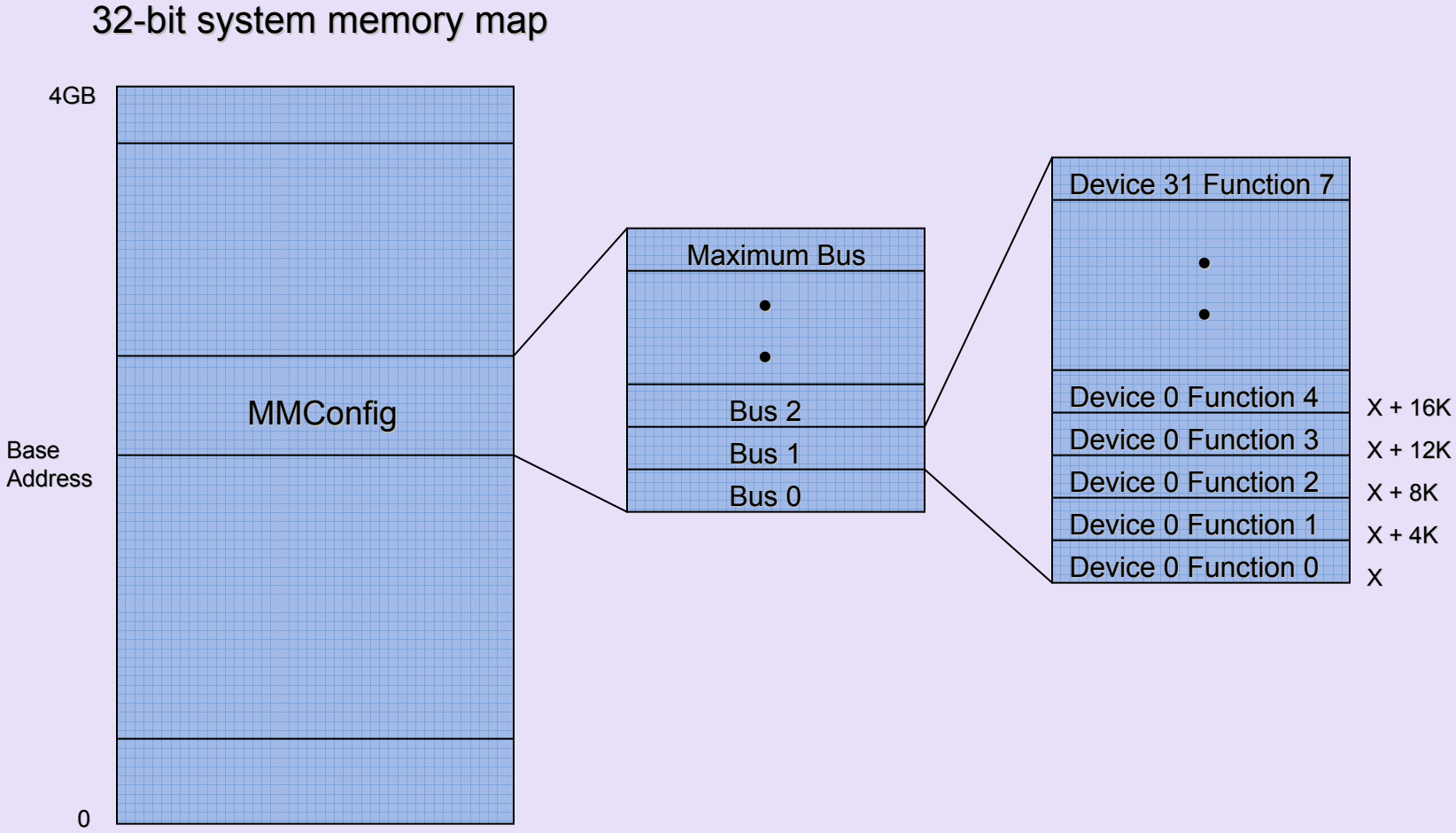
- I/O Mapped
- Addresses identical across machines
- Only supports 256 bytes/device
- Only supports 256 buses

Memory-Mapped Access Model (MMConfig)



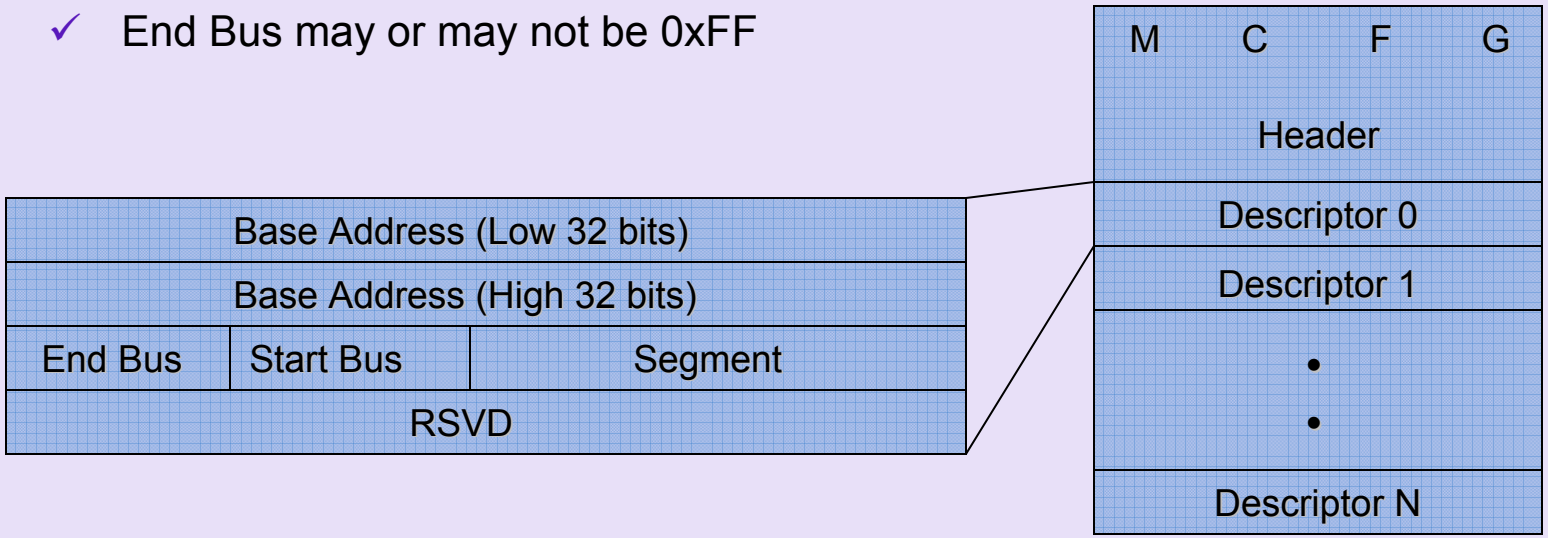
- Memory Mapped
- Requires up to 28 bits of memory
  - ✓ 256MB
- Supports 4K bytes/device
  - ✓ Each device is on its own 4K memory page
- Supports 256 buses per base address

MMConfig and the System Memory Map

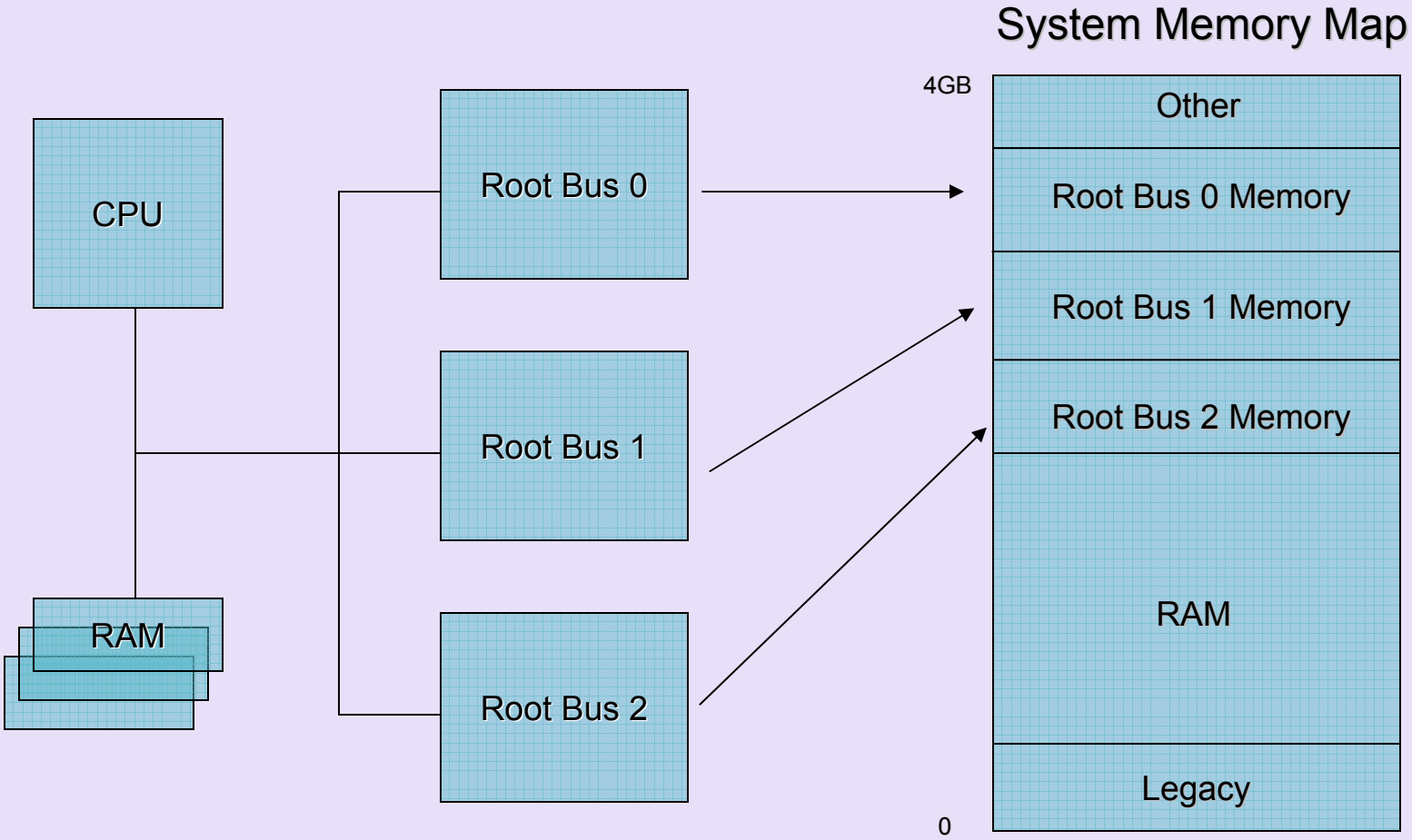


Report MMConfig through MCFG

- ACPI table defined in PCI Firmware Spec v3.0
- Memory range described here is reserved
  - ✓ 1MB per bus
- Simple case
  - ✓ 1 segment
  - ✓ Start Bus will be 0
  - ✓ End Bus may or may not be 0xFF



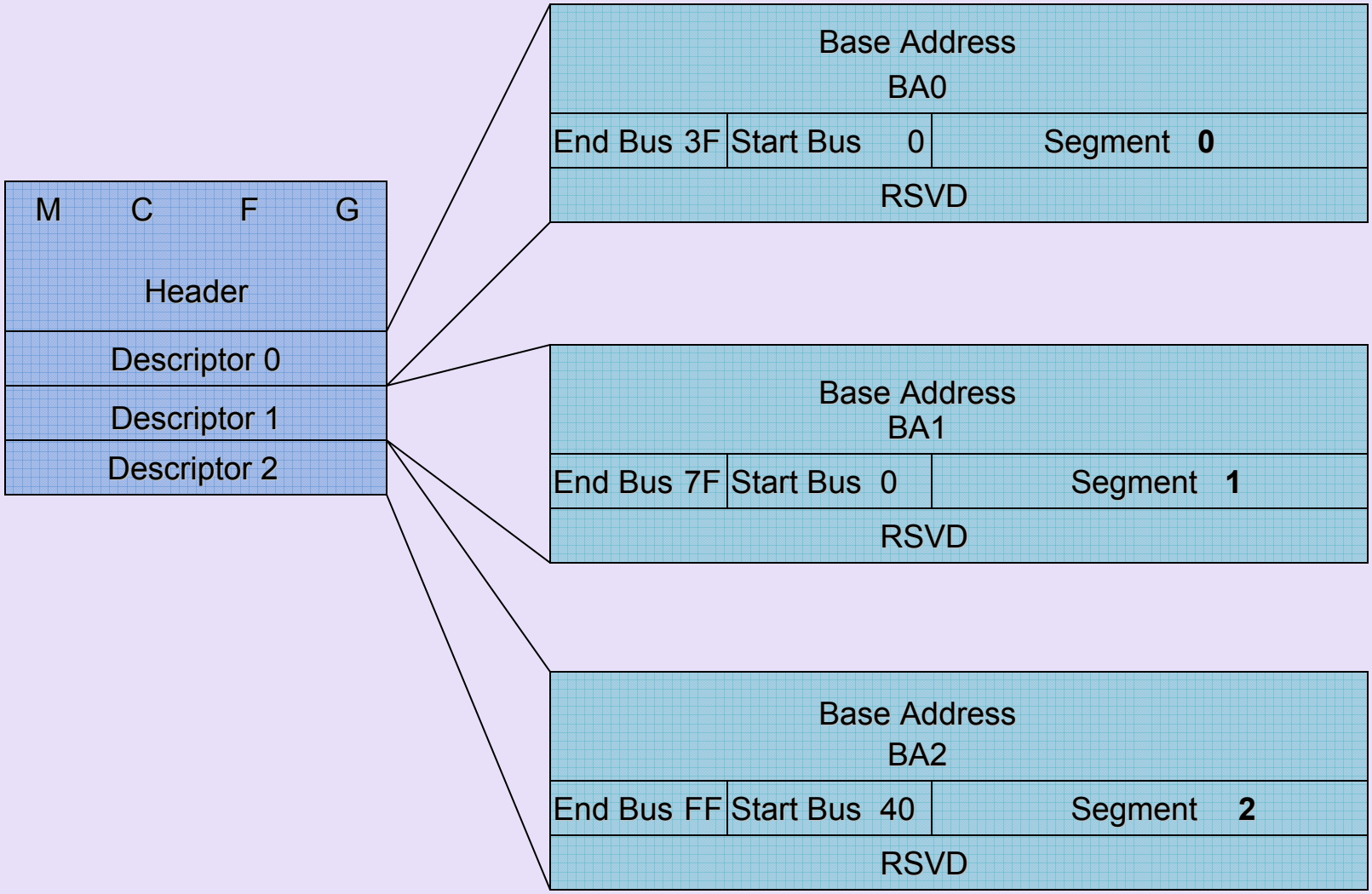
Multi-Root Machines And Multiple MMConfig Regions



Multiple Base Addresses Means Segments

- MCFG table only supports one base address per segment
- Segment background
  - ✓ A way to get different bus number domains
    - E.g., two PCI-PCI bridges could have the same bus number
  - ✓ Requires a way to differentiate between segments when issuing config cycles
    - This is wasn't supported by CFC/CF8
    - It is supported by memory-mapped config – each base address corresponds to a segment
- A new usage
  - ✓ Originally designed for systems where 256 buses weren't enough
  - ✓ These multi-root systems aren't bus-number constrained, just need multiple base addresses

Updated MCFG Table



Agenda

- Background
- PCI Express Configuration Regions
- Interrupts and MSI/MSI-X
- Break
- Power Management Events
- Power Negotiation
- Hot-Plug
- Summary
- Q and A

Interrupts

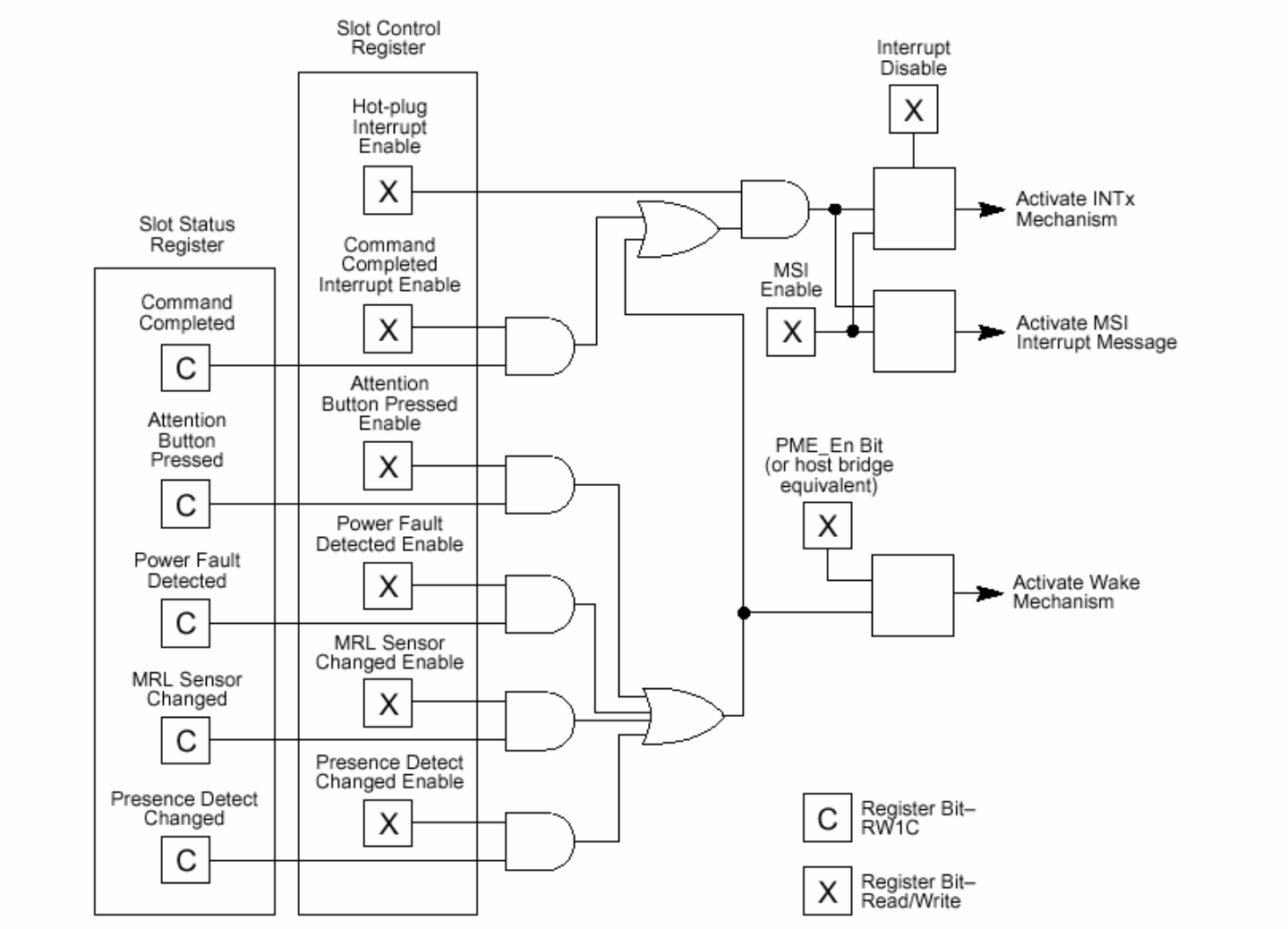
- PCI Express* supports two interrupts mechanisms:
  - ✓ INTx: level triggered
  - ✓ MSI: edge triggered
- INTx and MSI are mutually exclusive
  - ✓ Enabling MSI disables INTx
  - ✓ MSI is not controlled by INTx disabled bit
  - ✓ MSI is a memory request and can be disabled by BME (Bus Master Enable bit)
  - ✓ Neither MSI nor INTx can be generated in non-D0 state

Interrupts

- Level triggered interrupts can result in interrupt storms
  - ✓ Especially high risk in virtual wire scenarios if de-assert messages not sent correctly
  - ✓ Be sure to:
    - De-assert INTx in low power states
    - De-assert INTx when source masked
    - De-assert INTx when interrupts disabled
  - ✓ Asserts / De-asserts must be sent in pairs
- Switches must synthesize de-asserts as necessary
  - ✓ For example, if attached device is surprise removed

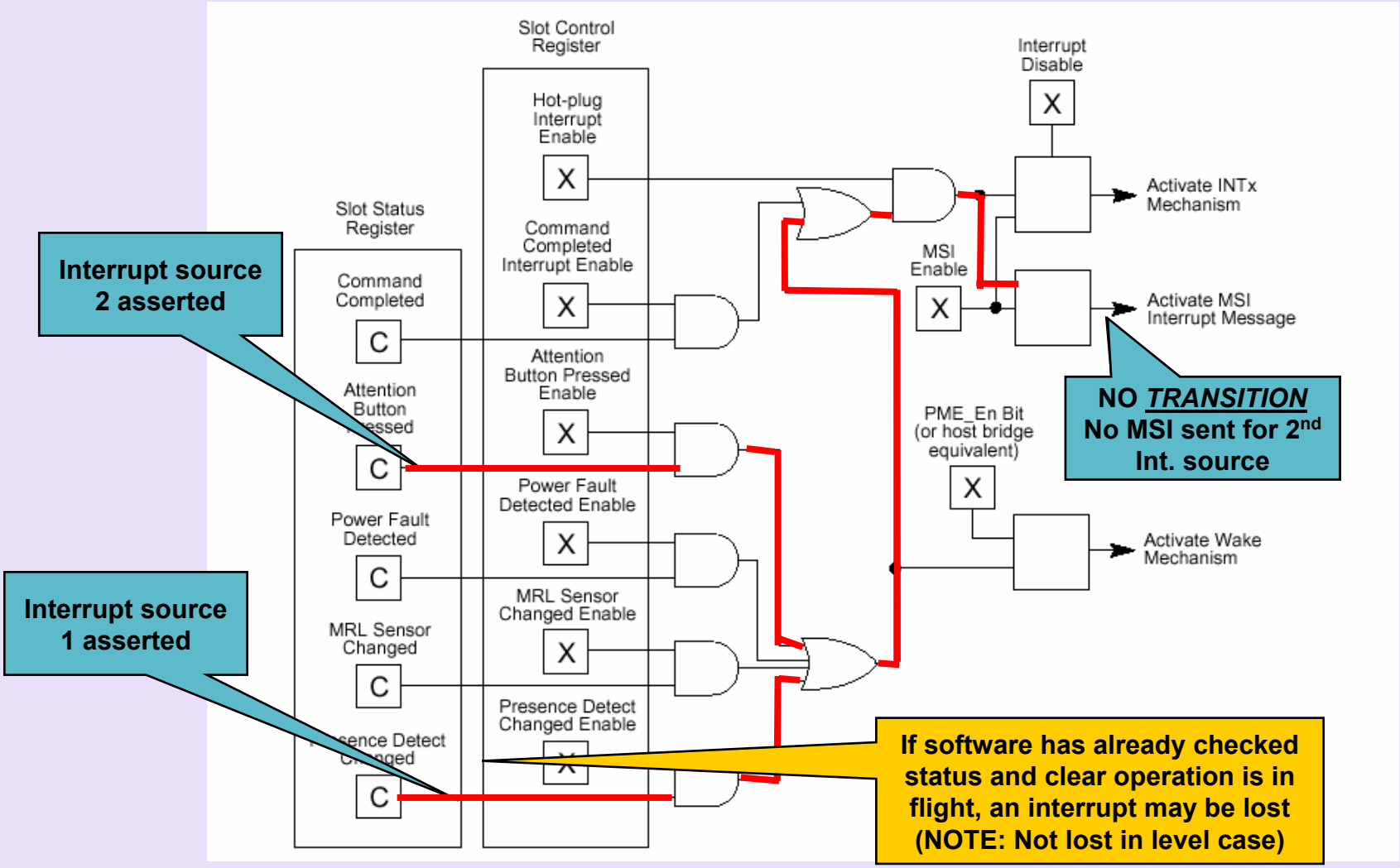
Level Triggered Interrupts: Design Example

NOTE: Example intended to illustrate device interrupt logic. Do not use example for hot-plug.



MSI Gotchas with simple Wire-OR

NOTE: Example intended to illustrate device interrupt logic. Do not use example for hot-plug.

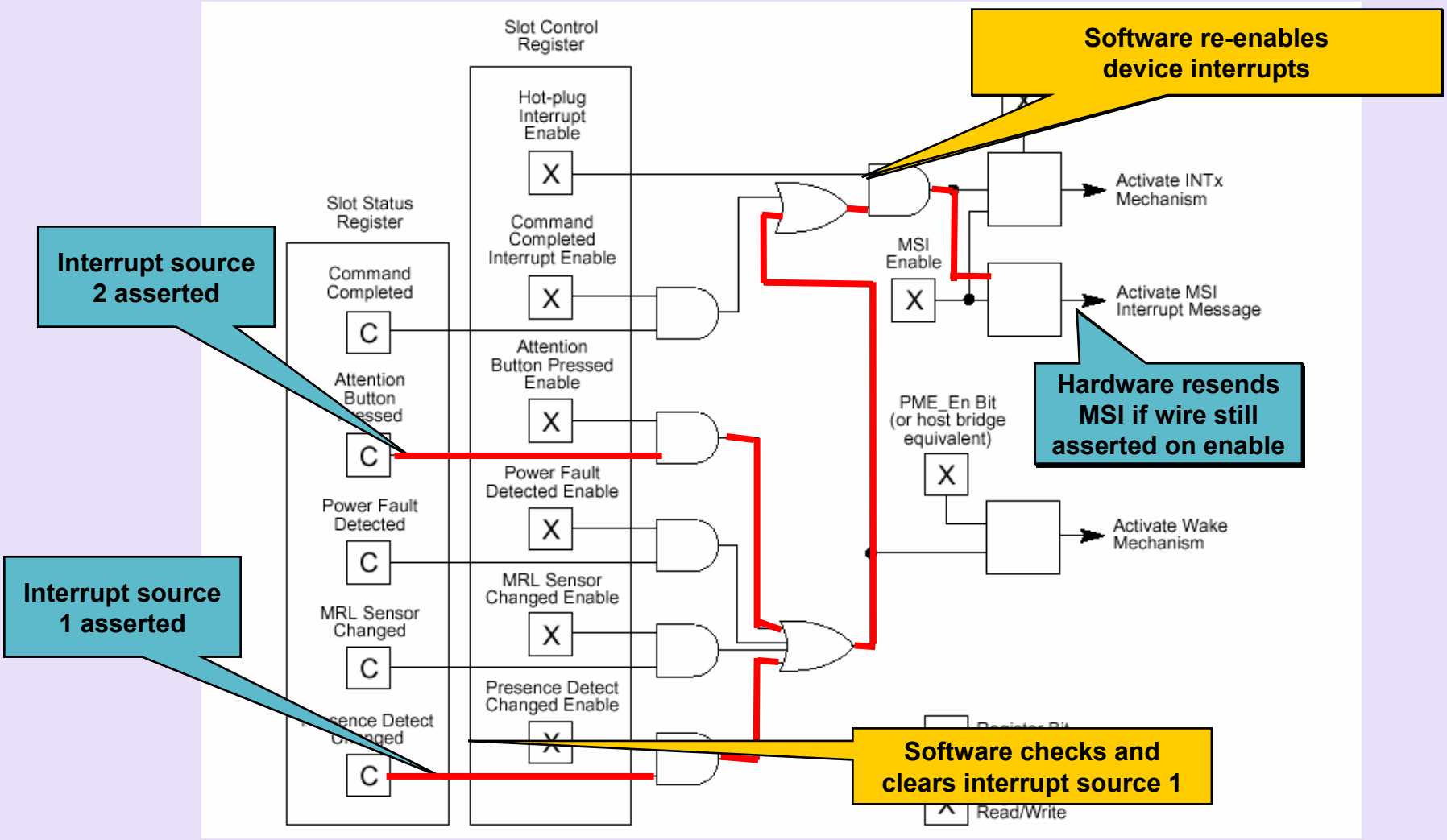


Avoid Losing Interrupts

- Existing design wire-OR signals cannot be simply converted to edge at wire-OR output
- Simplest fix requires both hardware and software to be carefully designed
- Device must evaluate internal line status when interrupts are masked/disabled
  - ✓ Can be a global device specific mask or enable
    - – But *NOT* the “MSI Enable” bit in MSI Cap Structure
- Device (re)sends any message on interrupt enable if status (or wire-OR status) still set
- ISR must be designed correctly

Edge Triggered Interrupts: Design Example

NOTE: Example intended to illustrate device interrupt logic. Do not use example for hot-plug.



Representative MSI ISR

```
Device_MSI_ISR()
{
  DisableInterrupts();
  CheckAndClearSources();
  // Clear only if status set for given source

  ServiceSources();
  // Or alternatively queue for deferred
  // processing

  EnableInterrupts()
  // Hardware must re-evaluate and resend any
  // pending interrupts
}
```

MSI Benefits and Optimizations

- Possibility of interrupt storms is greatly reduced
- MSI is a memory write and pushes data on the same VC
  - ✓ PIO type status checks can be avoided in many cases if using exclusive unshared interrupts
- Multiple MSI / MSI-X usages
  - ✓ Allows dedicated interrupt service routines for internal device sources
    - For example, separate reader/writer threads
    - Dedicated (non-shared) ISR does not have to check possible interrupt sources prior to servicing
  - ✓ Multiple MSIs can target different processors for interrupt load balancing

Agenda

- Background
- PCI Express Configuration Regions
- Interrupts and MSI/MSI-X
- Break
- Power Management Events
- Power Negotiation
- Hot-Plug
- Summary
- Q and A

Agenda

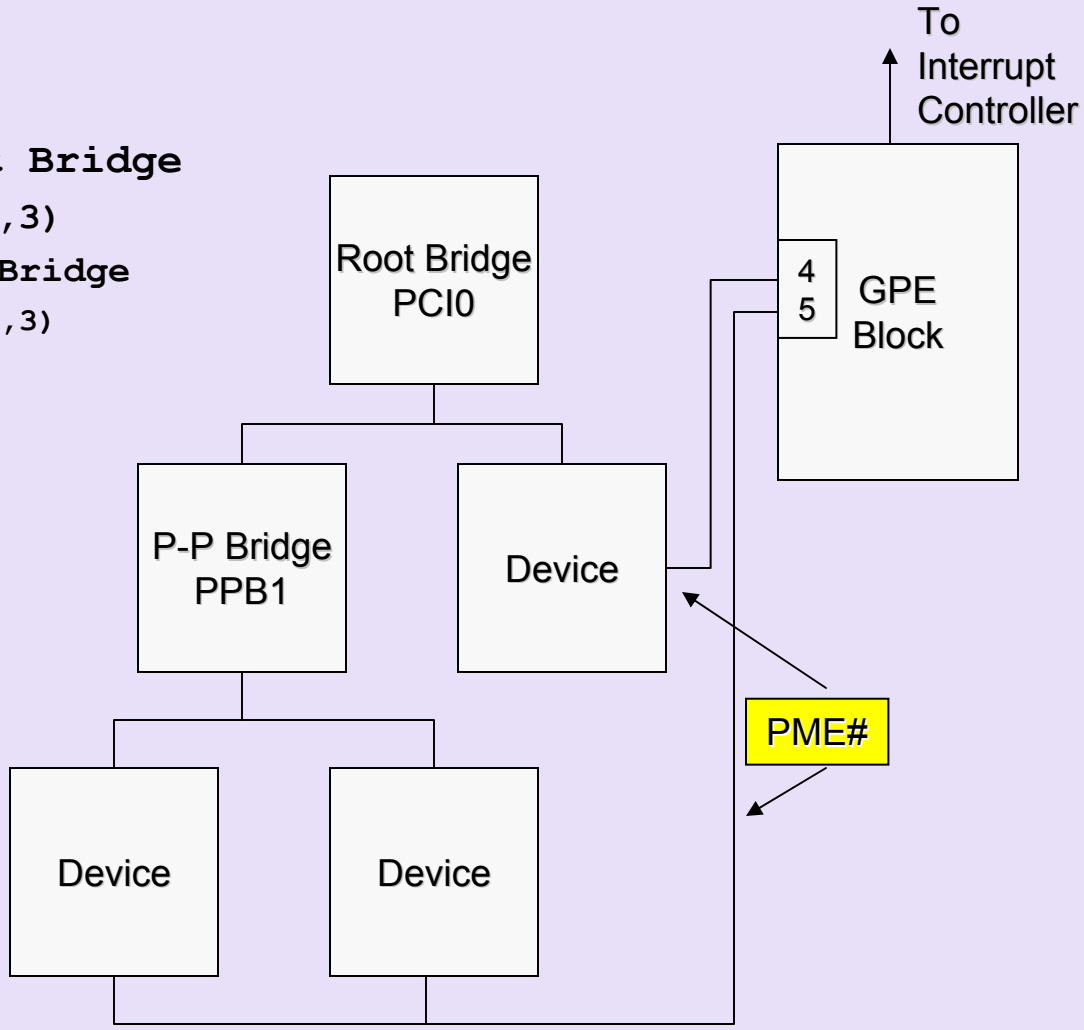
- Background
- PCI Express Configuration Regions
- Interrupts and MSI/MSI-X
- Break
- Power Management Events
- Power Negotiation
- Hot-Plug
- Summary
- Q and A

Legacy PCI/PCI-X PME model

- PME# signal wired together
- Each of these wires connected to a GPE
- _PRW object on bridge correlates GPE to PME for that bus
- When PME fires, OS runs _Lxx method for GPE
  - ✓ This ASL issues a Notify(2) to the bridge causing the wakeup
  - ✓ Notify causes OS to signal driver of waking device
- Danger in sharing PME# per bus
  - ✓ OS may not be able to locate the source of the PME
    - – PCI-PCI bridge does not signal PME Status for downstream device PME# assertion
  - ✓ GPE interrupt storms crash the system

PCI/PCI-X PME model example

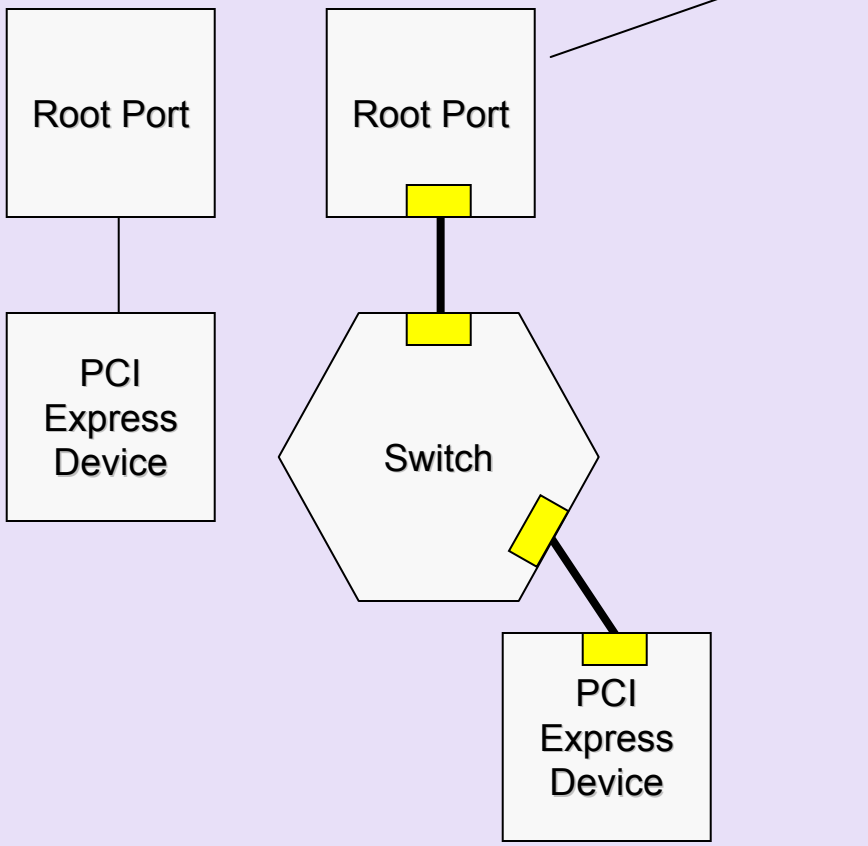
- `Device(PCI0) //Root Bridge`
  - ✓ `Name(_PRW,Package(4,3)`
  - ✓ `Device(PPB1) //P-P Bridge`
    - – `Name(_PRW,Package(5,3)`
- `Scope(\_GPE)`
  - ✓ `Method(_L04,0)`
    - – `Notify(PCI0,2)`
  - ✓ `Method(_L05,0)`
    - – `Notify(PPB1,2)`



PCI Express PME Model

- PME transmitted in-band as a message
  - ✓ If power is off and message can't be transmitted, WAKE# is signaled side-band
  - ✓ WAKE# turns power back on but doesn't cause any OS-visible event
- Root Port traps the message
  - ✓ Source of the message is recorded
  - ✓ If enabled, a native device interrupt is triggered by the root port
  - ✓ OS registers an ISR and receives the interrupt
  - ✓ Handled entirely in the OS – no ASL involved
- Three important bits
  - ✓ Root PME Requester ID – Source of message
  - ✓ Root PME Interrupt Status – Whether a message has been received
  - ✓ Root PME Interrupt Enable – Whether a message generates an interrupt

PCI Express PME Model Example



Message is Transmitted

Message is Trapped
PME Requestor ID
-B/D/F of PCI Express Device
PME Status = 1

Interrupt is Generated
If PME Interrupt Enable = 1
And other interrupt enables are set

Supporting Both PCI and PCI Express Models

- Need a MUX
  - ✓ Controlled by a chipset-specific “Native Mode” bit
    - If “Native Mode” is 0, PME message causes GPE
    - If “Native Mode” is 1, PME message causes an Interrupt
  - ✓ “Native Mode” bit cannot be any PCI Express spec-defined bit
    - Specifically, “Native Mode” bit cannot be Root PME Interrupt Enable
    - Scenario
      - OS takes native control of PME
      - OS decides to stop handling PME interrupts – clears Root PME Interrupt Enable
      - Subsequent PMEs cause GPEs to occur
- GPE Sharing
  - ✓ PCI Express root ports can share a single GPE
  - ✓ Avoid sharing with PME# from a PCI/PCI-X bus
    - Legacy workarounds for PCI PME# might interfere

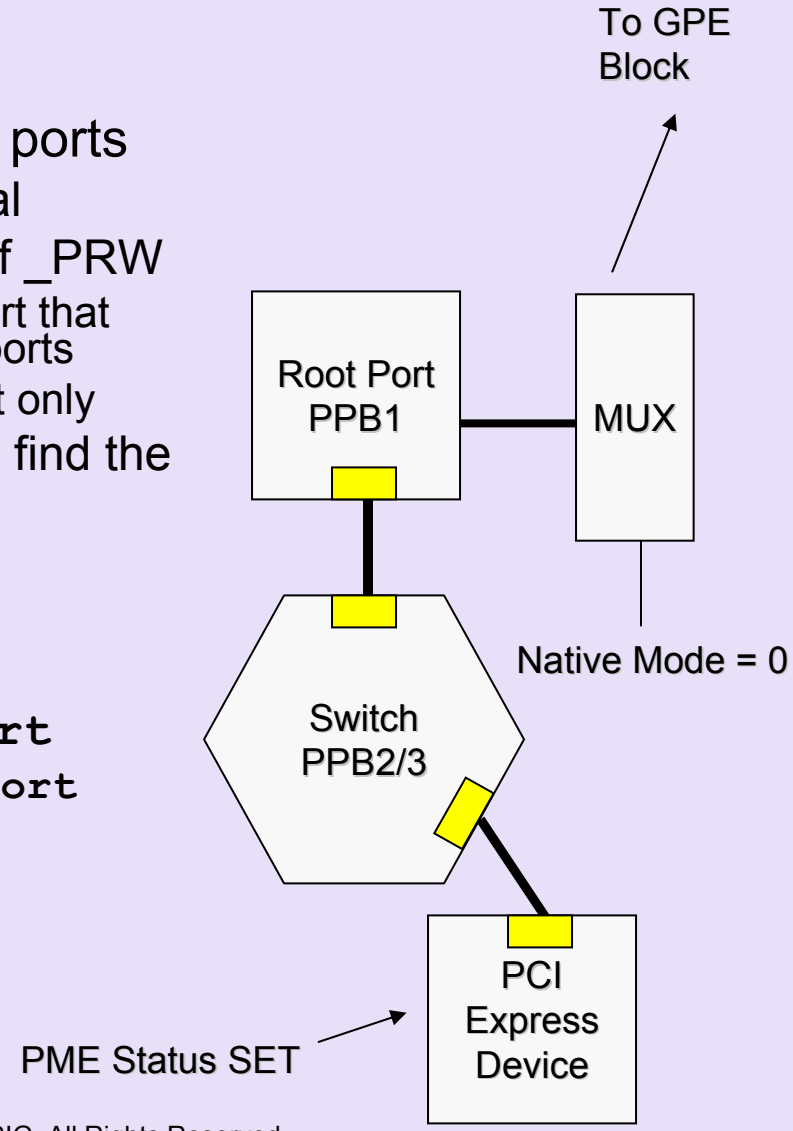
PCIEXP_WAKE_ Bits

- Two new bits in PM1 registers
  - ✓ PCIEXP_WAKE_DIS in PM1 Enable, PCIEXP_WAKE_STS in PM1 Status
  - ✓ If PCIEXP_WAKE_DIS is 1, PCI Express PME cannot cause system wakeup
  - ✓ PCIEXP_WAKE_STS indicates whether PCI Express PME caused system wakeup
- Hardware must support these bits
  - ✓ See PCI Firmware Spec v3.0
- Firmware does not need to modify them
  - ✓ PCIEXP_WAKE_DIS set to 0 allows PCI Express wakeup

Simulating PCI PME with PCI Express Hardware

- Implement _PRW on PCI Express ports
  - ✓ Include GPE routing of legacy signal
  - ✓ See OS requirements for location of _PRW
    - OS might require _PRW on every port that can generate a PME – not just root ports
    - OS might support _PRW on root port only
  - ✓ Fundamentally, OS must be able to find the source device

- **`Device(PPB1) //Root Port`**
  - ✓ **`Name(_PRW, Package(4,3))`**
  - ✓ **`Device(PPB2) //Upstream Port`**
    - **`Device(PPB3) //Downstream Port`**
      - **`Name(_PRW, Package(4,3))`**



Notifying OS of PME Source

- Implement ASL method corresponding to GPE number
  - ✓ _L08 (or _E08 for edge-triggered) for GPE 8
- This routine needs to issue a Notify(2)
- Appropriate target of Notify(2) is OS-specific
  - ✓ If Notify must be issued to the source bus

  (1) Find source of PME from root port PME Requester ID field

  (2) Bus number can be mapped to requesting bus (see next slide)
- In all cases

  (3) Issue Notify

  (4) Clear the Root PME Status bit

  (5) If shared, repeat for all root ports
- _Lxx method should not be run in Native Mode
  - ✓ Will be if PCI Express PME is shared with other sources
  - ✓ In this case, do nothing for PME handling if run in Native Mode

Mapping Bus Number to Bridges

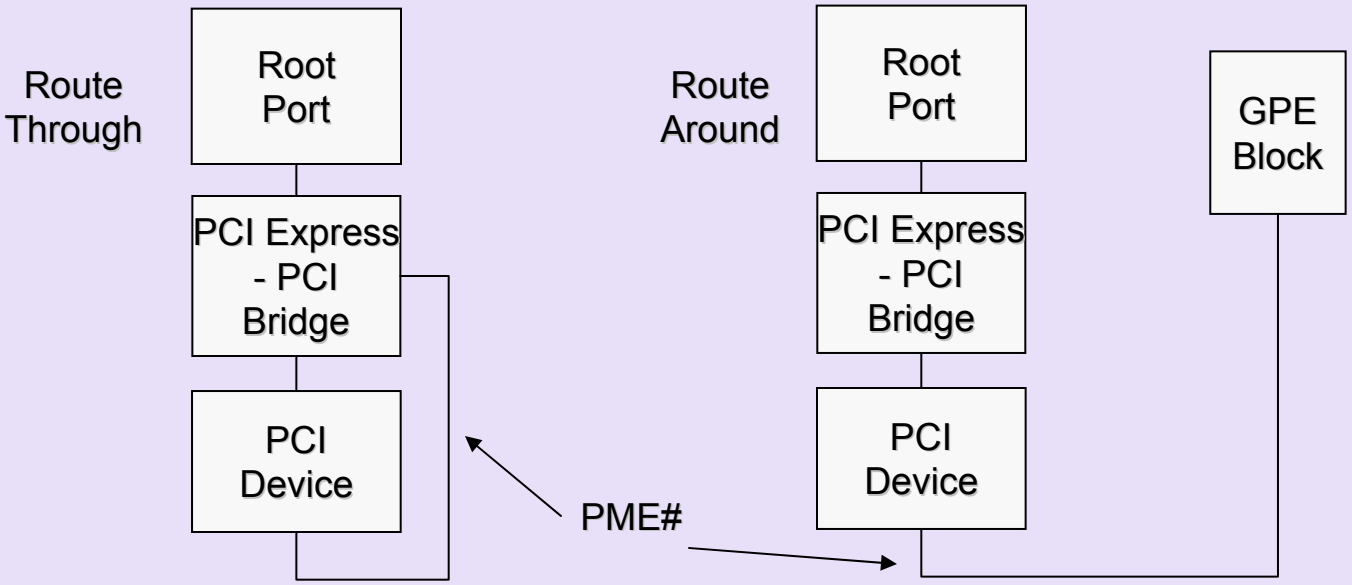
- Notify(PPB3,2)
  - ✓ PPB3 is the name of the ACPI device object for the source bridge
  - ✓ _Lxx method knows bus number of PME source
    - needs to map the source bus number to source bridge name
  - ✓ Means tracking which bus numbers are assigned to which bridges
- Mapping table
  - ✓ For each bridge described in the namespace, assign a numerical handle
  - ✓ In _REG for each bridge, read Secondary bus number
  - ✓ Keep 256 entry mapping table
  - ✓ In entry for secondary bus number, record bus cookie
- _Lxx logic
  - ✓ Get handle for PME source bus number
  - ✓ A big if statement (or something more elegant)
    - if cookie==1 Notify(PCI1,2), else if cookie==2 Notify(PCI2,2)

Switching To Native Mode

- System should boot in legacy mode
  - ✓ Both coming out of S4 and S5
- PCI Express aware OS can switch into Native Mode
- _OSC
  - ✓ Implement per host bridge
  - ✓ Needs to switch “Native Mode” bit
  - ✓ No other changes needed – namespace doesn’t need to change
    - – _PRWs can stay in place – they will be ignored for native OS
  - ✓ Can’t be un-done – system will be in native mode until the next boot
- Firmware can reject transition
  - ✓ OS policy whether to allow this rejection
  - ✓ Attend firmware presentation

PCI Express To PCI Bridges

- PCI/PCI-X and PCI Express PME mechanisms need to inter-operate
- Two options:
  - ✓ Route PCI/PCI-X PME# around the bridge and directly to GPE block
  - ✓ Route PCI/PCI-X PME# through the bridge
    - Bridge will convert it into a PCI Express PME message
    - Message will have requester ID of the secondary side of the bridge



Which Option?

- Route Through Bridge
  - ✓ Pros
    - PME Is Delivered As A Native Message, So When Native OS Is Available, No Firmware Is Required
  - ✓ Cons
    - May Not Work For Hierarchies More Than 1 Deep
      - For OS's requiring direct source bus notification
      - Requester ID Only Identifies The Immediate Secondary Side Of The Bridge
      - No Way To Track Back To Initial Source
    - Some Bridges May Not Support This Feature
- Route Around Bridge
  - ✓ Con: Requires Firmware To Handle GPE – Extends Legacy
  - ✓ Only Use If Routing Through The Bridge Won't Work
  - ✓ The Same Rules For All PCI/PCI-X PME Apply
    - One PME Per Bus
    - One GPE Per Bus

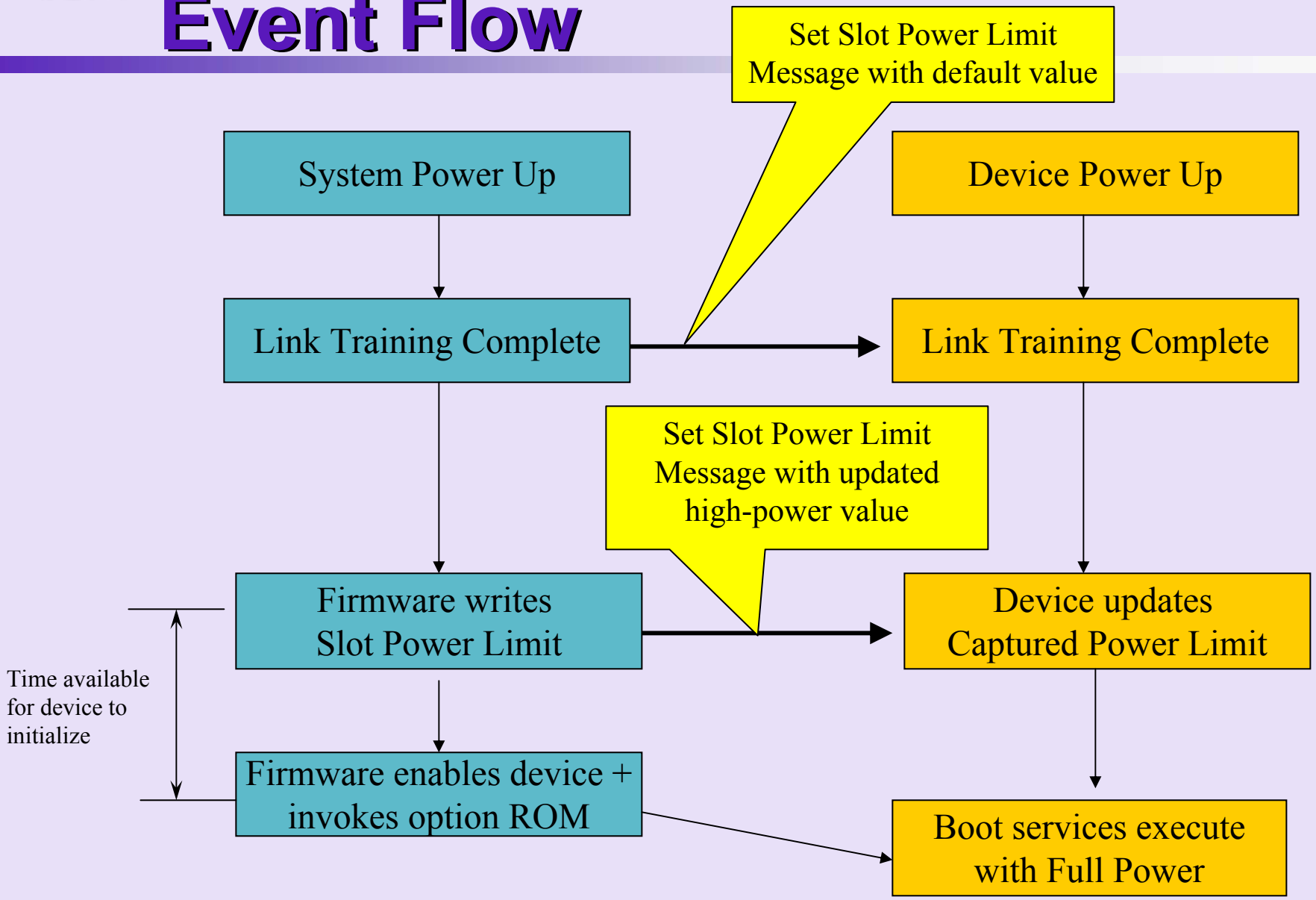
Agenda

- Background
- PCI Express Configuration Regions
- Interrupts and MSI/MSI-X
- Break
- Power Management Events
- Power Negotiation
- Hot-Plug
- Summary
- Q and A

Power Negotiation

- Slot Power Notification mechanism
  - ✓**Flexible and scalable mechanism to notify a PCI Express adapter of additional available slot power**
- Conditions that trigger the message:
  - ✓**On a Configuration Write to the Slot Capabilities register (when the Data Link Layer reports DL_Up status)**
  - ✓**Anytime when a Link transitions from a non-DL_Up status to a DL_Up status**

Power Negotiation: Event Flow



Power Negotiation Considerations

- Devices strongly encouraged to provide minimum level of functionality within the CEM specification power
  - ✓**Driver or Option ROM can detect available power through captured power limit registers**
- Power Budgeting Extended Capability can allow flexibility in power redistribution
  - ✓**Implementation specific dynamic power redistribution**
  - ✓**Devices are strongly encouraged to implement this capability**

Agenda

- Background
- PCI Express Configuration Regions
- Interrupts and MSI/MSI-X
- Break
- Power Management Events
- Power Negotiation
- Hot-Plug
- Summary
- Q and A

PCI Express Hot-Plug Model

- Based on PCI/PCI-X Hot-Plug standard - SHPC
- Port contains slot registers
  - ✓ Control for power to the slot
  - ✓ Control for indicators on the slot
  - ✓ Notification of card insertion/removal
  - ✓ Notification of latch open/close
  - ✓ Notification of attention button press
- Notification is done through device interrupt
- Some form factors will implement different sets of these features
  - ✓ PCI Express card – power control and maybe the others
  - ✓ Server I/O Module – all but the latch (as currently defined)
  - ✓ ExpressCard – insertion/removal notification only

Current Hot-Plug Support

- Hot-Plug controllers are assumed to be vendor-specific
  - ✓ Features are largely the same, however
- Control is done through ACPI
  - ✓ Notification is through a GPE
  - ✓ _Lxx method for GPE calls Notify to inform OS of event
  - ✓ _EJx method called when device ejection completes
    - _EJx is only included when system controls slot power
    - E.g., ExpressCard slots shouldn't have _EJx
  - ✓ _RMV method tells OS device is removable
    - Required if _EJx isn't implemented
- For PCI Express Hot-Plug, the model remains the same
  - ✓ Registers are spec-defined instead of device-specific

Hot-Plug Support: Insertion Example

- Hardware detects that a device is inserted and signals a GPE
- _Lxx method for GPE runs
  - ✓ Firmware applies power to the device and turns on the power indicator
  - ✓ Firmware calls Notify(0) on the bridge that exposes the slot
- OS handles Notify instruction
  - ✓ A bus scan gets triggered
  - ✓ The new device is found
  - ✓ Drivers are loaded and the device is brought online

Hot-Plug Support Removal Example

- Hardware-initiated Removal with Power Control (e.g., Edge-style cards)
  - ✓ Hardware detects a button press on the chassis and signals a GPE
  - ✓ _Lxx method for GPE runs
    - Firmware calls Notify(3) on the device in the slot
  - ✓ OS handles Notify instruction
    - Application/driver handles are closed
    - _EJx method is called on device in the slot
  - ✓ Firmware handles _EJx method
    - Power is removed from the slot and the indicator is turned off
- Hardware-initiated Surprise Removal (e.g. ExpressCard)
  - ✓ Hardware detects that device has been surprise removed and signals a GPE
  - ✓ Firmware calls Notify(0) on the bridge exposing the slot
  - ✓ Bus is re-scanned, drivers and app handles are surprise-closed
- OS-initiated Removal
  - ✓ Handles are closed and drivers are unloaded
  - ✓ _EJx method is called on device in the slot

Supporting PCI Express Hot-Plug on Existing OS

- As in PME, need a MUX
  - ✓ Controlled by a chipset-specific “Native Mode” bit
  - ✓ Governs whether hot-plug event causes GPE or interrupt
  - ✓ “Native Mode” bit cannot be any PCI Express spec-defined bit
    - – Specifically, can’t be Hot Plug Interrupt Enable
- Needed per-port
  - ✓ Chipset can provide this feature for root ports
  - ✓ Switches also need to provide it
  - ✓ This is different from PME
- Operating in legacy mode
  - ✓ GPE handling code should handle events and call Notify
  - ✓ ASL flow is exactly the same as for PCI/PCI-X Hot-Plug

Switching To Native Mode

- Virtually the same as PME
- One of the _OSC capability bits is for PCI Express Hot-Plug
  - ✓ _OSC needs to exist on each host bridge
- OSHP
  - ✓ Defined by SHPC and PCI Express 1.0 specs
  - ✓ Namespace collision
  - ✓ No feedback from firmware

Agenda

- Background
- PCI Express Configuration Regions
- Interrupts and MSI/MSI-X
- Break
- Power Management Events
- Power Negotiation
- Hot-Plug
- Summary
- Q and A

Summary

- Multiple PCI Express Configuration Regions can be used to support very large topologies
- MSI has several advantages but correct interrupt handling is required to avoid loss of edge-triggered interrupts
- PCI Express implementations must consider power management strategies for both current and future OS
- PCI Express Hot-Plug can be implemented on existing OS with ACPI support

Agenda

- Background
- PCI Express Configuration Regions
- Interrupts and MSI/MSI-X
- Break
- Power Management Events
- Power Negotiation
- Hot-Plug
- Summary
- Q and A

Thank you for attending the
PCI-SIG Developers Conference 2004.

For more information please go to
www.pcisig.com

PCI Express Advanced Software Topics

Davis Walker
Software Design Engineer
Microsoft Corporation

Prashant Sethi
Software Architect
Intel Corporation


PCI
SIG™