PCIe Hot Plug on Windows

Though the OS supports PCIe hot plug, on-premise indicators are sparsely specified. A method to ascertain system support is put forward.

- In Device Manager, hot plug support is part of Root Complex properties, PCIExpressNativeHotPlugControl. If it is absent, then the platform does not grant *native hot plug*.

```
PS > (
        Get-PnpDevice -InstanceId "ACPI\PNP0A08*" | Get-PnpDeviceProperty
        DEVPKEY_PciRootBus_PCIExpressNativeHotPlugControl |
        Where-Object { $_.Data }
    ).InstanceId
```

At startup, the OS feeds the *supported and controlled PCIe* features to the **ACPI _OSC method**. System firmware responds with a bitmask for granted features, part of *Control* field. *Section 4.5 / PCI Firmware Specification Revision 3.2.*

- In kernel mode, the root complex fully qualified name must be identified. Use `!amli find _OSC` and look at the 1st entry, as a child of **_SB.XXXX**. The `_HID` must match *EISAID("PNP0A08")*.

```
0: kd> !amli find _OSC
\_SB.PCI0._OSC
\_SB.PR00._OSC
\_SB.PR01._OSC
\_SB.PR02._OSC
\_SB.PR03._OSC
...
\_SB._OSC
0: kd> !amli dns /v \_SB.PCI0._HID
Integer(_HID:Value=0x00000000080ad041[134926401])
0: kd> !amli dns /v \_SB.PCI0.CTRL
ACPI Name Space: \_SB.PCI0.CTRL (ffffbb0e1bf6e3e0)
Integer(CTRL:Value=0x0000000000000015[21])
0: kd> !amli dns /v \_SB.PCI0.SUPP
ACPI Name Space: \_SB.PCI0.SUPP (ffffbb0e1bf6e330)
Integer(SUPP:Value=0x000000000000001f[31])
```

_OSC method suffers minor changes between processor generations. Within a generation, there are *use conditions* that demand different implementation.

Intel maintains a compact firmware implementation through slimbootloader project. The _OSC method is implemented in **HostBus.asl** for each CPU platform.

CTRL ACPI variable shows the granted access from firmware to OS. Bit 0 represents

> PCI Express Native Hot Plug control
> The operating system sets this bit to 1 to request control over PCI Express native hot plug. If the operating system successfully receives control of this feature, it must track and update the status of hot plug slots and handle hot plug events as described in the PCI Express Base Specification (including determining whether the associated slot(s) support hot plug).

The OS registers interrupt handlers for root port or downstream port.

```
0: kd> !idt
Dumping IDT: fffff8070701e000
50: pci!ExpressRootPortMessageRoutine (KINTERRUPT ffffa7016b711a00)
51: pci!ExpressDownstreamSwitchPortInterruptRoutine (KINTERRUPT ffffa7016b711000)

60: pci!ExpressRootPortMessageRoutine (KINTERRUPT ffffa7016b711b40)
61: pci!ExpressDownstreamSwitchPortInterruptRoutine (KINTERRUPT ffffa7016bfd6dc0)
```

The absence of these interrupts indicates lack of hot plug support.

Algorithm

Hot plug is handled by the ISR, DPC and collateral timer for error handling. Overview:

```
if (!SlotControl.HotPlugInterruptEnable) {
   return;
}
if (SlotControl.PresenceDetectEnable && SlotStatus.PresenceDetectChanged) {
   W1TC(SlotStatus.PresentDetectChanged);
   if (SlotStatus.PresenceDetectState) {
      IoInvalidateDeviceRelations(Pdo)
   } else {
      IoRequestDeviceEjectEx(ChildPdo);
   }
}
```

With `IoInvalidateDeviceRelations`, the PnP manager issues `IRP_MN_QUERY_DEVICE_RELATIONS` on the port. The PCI driver rescans the bus, creates a new device object.

```
pci!PciBus_QueryDeviceRelations
   pci!PciScanBus
      pci!PciProcessNewDevice
         lock xadd dword ptr [pci!PciDeviceSequenceNumber],r9d
         lea  r8,[pci!`string' (fffff807194f3af0)] = "\Device\NTPNP_PCI%04d"
         nt!IoCreateDevice
```

Tracing

Several opcodes indicate *ETW* support.

```
pci!PciDevice_Start+0x13a93:
cmp     dword ptr [pci!EntryReg+0x18 (fffff807194fc0c8)],5
jbe     pci!PciDevice_Start+0x13b68 (fffff8071951e4f8)
pci!PciDevice_Start+0x13aa0:
call    pci!TlgKeywordOn (fffff807194e391c)

pci!PciPreScanAssignBusNumbersSubtree+0x58:
cmp     dword ptr [pci!EntryReg+0x18 (fffff807194fc0c8)],5

pci!PciProcessNewRootBus+0x178:
cmp     dword ptr [pci!EntryReg+0x18 (fffff807194fc0c8)],5

dword ptr [pci!EntryReg+0x18 (fffff807194fc0c8)],5
jbe     pci!PciStartHotPlugController+0x17b (fffff807194ec56f)
pci!PciStartHotPlugController+0x84:
call    pci!TlgKeywordOn (fffff807194e391c)
```

The `logman.exe start -p <GUID>` that can be used is undocumented.

```
0: kd> uf pci!TraceLoggingRegisterEx
mov     rax,qword ptr [pci!EntryReg+0x20 (fffff807194fc0d0)]
lea     rcx,[rsp+20h]
movups  xmm0,xmmword ptr [rax-10h]
movdqu  xmmword ptr [rsp+20h],xmm0
call    nt!EtwRegister (fffff807040bfb00)

0: kd> dt nt!_GUID poi(pci!EntryReg+0x20)-0x10
{69a770dd-1cdb-46c0-91c9-cd2a9b76e061}
```

Automation

NativeHotPlugSupport.ps1 shows:

- ACPI _OSC implementation, CTRL and SUPP values
- Hot plug ISRs in the interrupt descriptor table
- Device objects associated with ISRs
- CPU name and model

```
PS > .\NativeHotPlugSupport.ps1 -Path .\MEMORY.DMP
ffffbb0e1bf6e6b2:[\_SB.PCI0._OSC]
ffffbb0e1bf6e6b2 : Store(Arg3, Local0)
ffffbb0e1bf6e6b5 : CreateDWordField(Local0, Zero, CDW1)
ffffbb0e1bf6e6bc : CreateDWordField(Local0, 0x4, CDW2)
ffffbb0e1bf6e6c4 : CreateDWordField(Local0, 0x8, CDW3)
ffffbb0e1bf6e6cc : If(LEqual(Arg0, GUID))
ffffbb0e1bf6e6d5 : {
ffffbb0e1bf6e6d5 : | Store(CDW2, SUPP)
ffffbb0e1bf6e6de : | Store(CDW3, CTRL)
ffffbb0e1bf6e6e7 : | If(LNot(NEXP))
ffffbb0e1bf6e6ee : | {
ffffbb0e1bf6e6ee : | | And(CTRL, 0xfffffff8, CTRL)
ffffbb0e1bf6e6fc : | }
ffffbb0e1bf6e6fc : | If(LEqual(TBTS, One))
ffffbb0e1bf6e704 : | {
ffffbb0e1bf6e704 : | | And(CTRL, 0xfffffff7, CTRL)
ffffbb0e1bf6e712 : | }
ffffbb0e1bf6e712 : | If(Not(And(CDW1, One, ), ))
ffffbb0e1bf6e71d : | {
ffffbb0e1bf6e71d : | | If(And(CTRL, One, ))
ffffbb0e1bf6e726 : | | {
ffffbb0e1bf6e726 : | | | NHPG()
ffffbb0e1bf6e72a : | | }
ffffbb0e1bf6e72a : | | If(And(CTRL, 0x4, ))
ffffbb0e1bf6e734 : | | {
ffffbb0e1bf6e734 : | | | NPME()
ffffbb0e1bf6e738 : | | }
ffffbb0e1bf6e738 : | }
ffffbb0e1bf6e738 : | If(LNot(LEqual(Arg1, One)))
ffffbb0e1bf6e73e : | {
ffffbb0e1bf6e73e : | | Or(CDW1, 0x8, CDW1)
ffffbb0e1bf6e749 : | }
ffffbb0e1bf6e749 : | If(LNot(LEqual(CDW3, CTRL)))
ffffbb0e1bf6e755 : | {
ffffbb0e1bf6e755 : | | Or(CDW1, 0x10, CDW1)
ffffbb0e1bf6e760 : | }
ffffbb0e1bf6e760 : | Store(CTRL, CDW3)
ffffbb0e1bf6e769 : | Store(CTRL, OSCC)
ffffbb0e1bf6e772 : | Return(Local0)
ffffbb0e1bf6e774 : }
ffffbb0e1bf6e774 : Else
ffffbb0e1bf6e776 : {
ffffbb0e1bf6e776 : | Or(CDW1, 0x4, CDW1)
ffffbb0e1bf6e781 : | Return(Local0)
ffffbb0e1bf6e783 : }

ffffbb0e1e9b825a:[\NHPG]
ffffbb0e1e9b825a : Store(Zero, \_SB_.PCI0.RP01.HPEX)
ffffbb0e1e9b826f : Store(Zero, \_SB_.PCI0.RP02.HPEX)
ffffbb0e1e9b8452 : Store(One, \_SB_.PCI0.RP01.HPSX)
ffffbb0e1e9b8467 : Store(One, \_SB_.PCI0.RP02.HPSX)

Integer(SUPP:Value=0x000000000000001f[31])
Integer(CTRL:Value=0x0000000000000015[21])

50: pci!ExpressRootPortMessageRoutine (KINTERRUPT ffffa7016b711a00)
Location: Bus 0x0, Device 1b, Function 6.
  DevObj 0xffffbb0e1c7f8060  Parent FDO DevExt 0xffffbb0e1aa04a30
  Vendor ID 8086 (INTEL CORPORATION)  Device ID A32E
  Subsystem Vendor ID 8086 (INTEL CORPORATION)  Subsystem ID 7270
```

```
51: pci!ExpressDownstreamSwitchPortInterruptRoutine (KINTERRUPT ffffa7016b711000)
Location: Bus 0x2e, Device 0, Function 0.
  DevObj 0xffffbb0e1c9b4570  Parent FDO DevExt 0xffffbb0e1c9064c0
  Vendor ID 10b5 (PLX TECHNOLOGY, INC.)  Device ID 8724
  Subsystem Vendor ID 10b5 (PLX TECHNOLOGY, INC.)  Subsystem ID 8724

60: pci!ExpressRootPortMessageRoutine (KINTERRUPT ffffa7016b711b40)
Location: Bus 0x0, Device 1b, Function 0.
  DevObj 0xffffbb0e1c7f7060  Parent FDO DevExt 0xffffbb0e1aa04a30
  Vendor ID 8086 (INTEL CORPORATION)  Device ID A32C
  Subsystem Vendor ID 8086 (INTEL CORPORATION)  Subsystem ID 7270

61: pci!ExpressDownstreamSwitchPortInterruptRoutine (KINTERRUPT ffffa7016bfd6dc0)
Location: Bus 0x2e, Device 1, Function 0.
  DevObj 0xffffbb0e1c9b5730  Parent FDO DevExt 0xffffbb0e1c9064c0
  Vendor ID 10b5 (PLX TECHNOLOGY, INC.)  Device ID 8724
  Subsystem Vendor ID 10b5 (PLX TECHNOLOGY, INC.)  Subsystem ID 8724

ProcessorNameString = REG_SZ Intel(R) Core(TM) i7-9850HE CPU @ 2.70GHz
Identifier = REG_SZ Intel64 Family 6 Model 158 Stepping 10
```

Notes

- IT personnel can use **DEVPKEY_PciRootBus_PCIExpressNativeHotPlugControl** to reveal hot plug support.
- Escalation accounts for **ACPI _OSC** method and **CTRL** value for the host bridge, part of the root complex.
- Script processing takes 40+ seconds, mostly spent in *!amli*.
- ETW traces can be enabled with **{69a770dd-1cdb-46c0-91c9-cd2a9b76e061}** provider.
- To locate the root or downstream ports that own the hot plug interrupt, dump the IDT, match the **KINTERRUPT→ConnectionData→Vectors[0].ControllerInput.Gsiv** with **!arbiter 4**

```
PS > $prefix = "https://raw.githubusercontent.com/armaber/scripts/refs/heads/main/";
     "Detect_OSC.js", "NativeHotPlugSupport.ps1" | foreach {
         Invoke-WebRequest $prefix/NativeHotPlugSupport/$PSItem -OutFile $PSItem;
      }
```