



# **PLX SDK**

## **General Notes & FAQ**

---

**August 2013**

# Table of Contents

---

- 1. ABOUT THIS DOCUMENT ..... 2
- 2. GENERAL NOTES ..... 3
  - 2.1 VIEW PLX DRIVER/API DEBUG MESSAGES VIA DEBUGVIEW .....3
  - 2.2 UNABLE TO SELECT A PLX DEVICE IN WINDOWS DUE TO SECURITY ISSUES .....8
  - 2.3 INCREASE 8311 DMA PERFORMANCE.....11
  - 2.4 8311 64-BIT DMA ISSUES.....13
  - 2.5 8311 & 8111 POWER MANAGEMENT ISSUE WITH WINDOWS .....14
  - 2.6 WINDOWS PCI EXPRESS NATIVE MODE & PLX DEVICES .....15
  - 2.7 PROGRAMMING BLANK OR NO-SIGNATURE (5AH) EEPROMS.....17
  - 2.8 EEPROM PROTECTION .....20
  - 2.9 DEALING WITH A CORRUPT EEPROM.....21
- 3. FAQ..... 22

## 1. About this Document

---

This document is provided as an additional resource to cover specific issues & questions that require more elaborate detail to answer. The **General** section provides useful tips or detailed explanations. The **FAQ** section provides answers to common questions related to the PLX SDK.

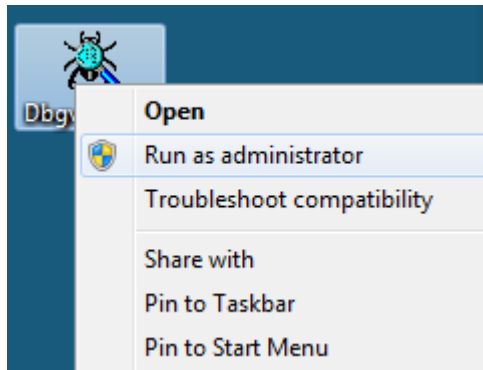
## 2. General Notes

---

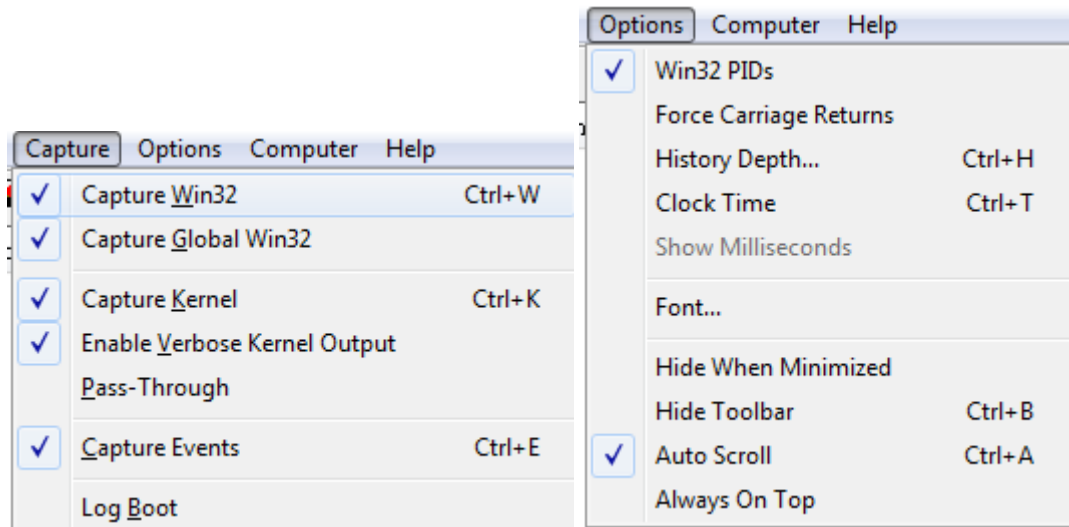
### 2.1 View PLX Driver/API Debug Messages via DebugView

This section describes how to setup DebugView to view debug messages from PLX debug builds of the API or drivers in Windows. PLX drivers and the API have numerous built-in debug messages designed to provide much useful information that generally aids in resolving issues. These messages are only built-in to debug builds of PLX software.

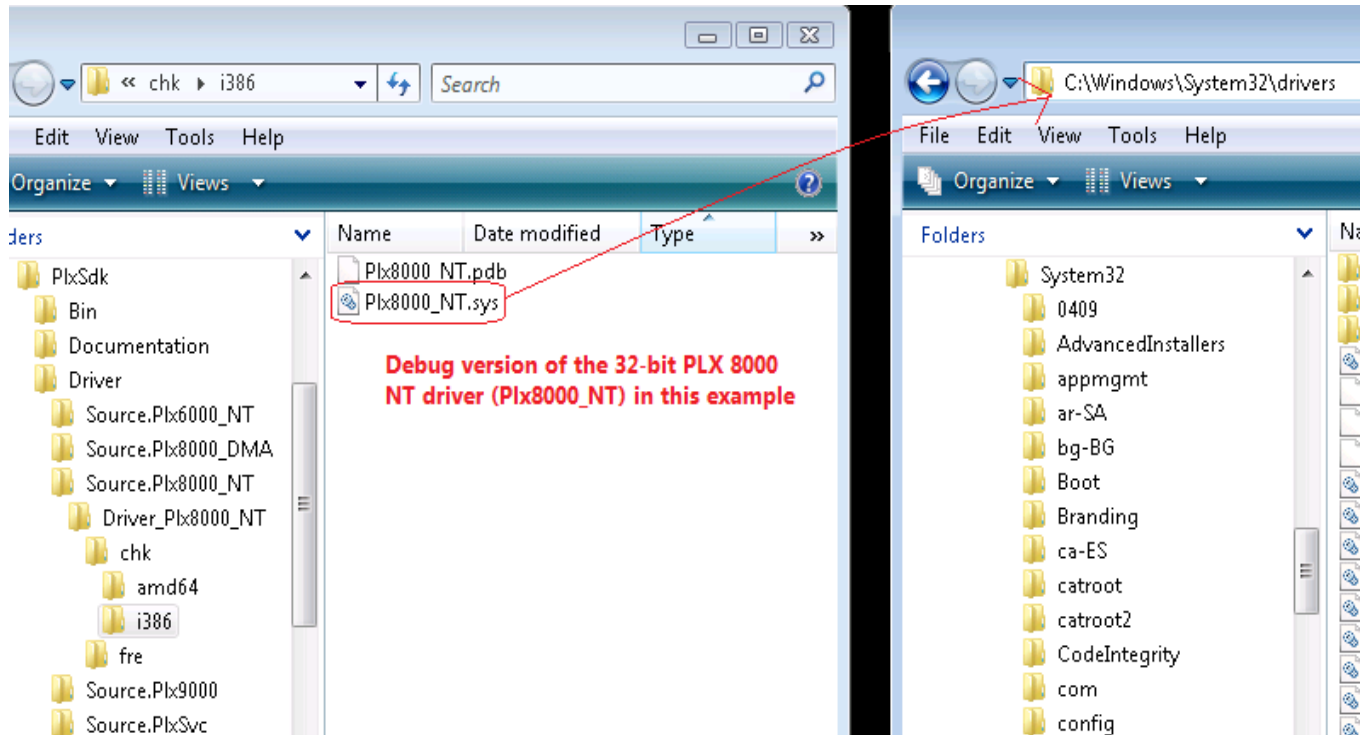
1. Download the latest version of Debug View  
<http://technet.microsoft.com/en-us/sysinternals/bb896647>
2. Run Debug View as Administrator



3. Setup the options as shown below

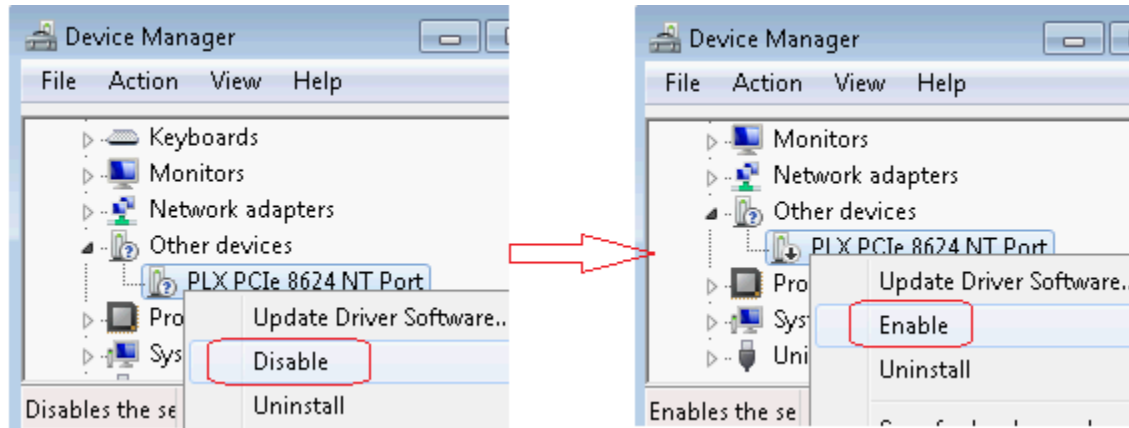


4. Copy the debug version of the PLX driver over the existing one in **Windows\System32\Drivers**. Make sure to match the running version of Windows to the correct driver version (ie “i386”=32-bit, “AMD64”=64-bit). To capture any PLX API debug messages, copy the PLX API debug build of the DLL from **PlxSdk\Windows\PlxApi\Debug** to **Windows\System32** (32-bit Windows) or **Windows\SysWOW64** (64-bit Windows)

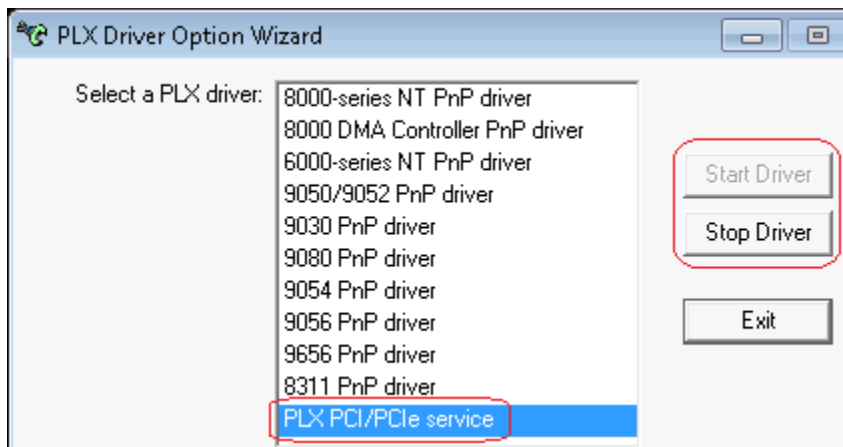


5. Load the new driver.

For a PLX Plug 'n' Play driver, use Device Manager to '**Disable**' the device, then '**Enable**' it.



For the PLX Service Driver, simply run PLX Driver Options Wizard, select PLX Service driver and then Start/Stop.



6. When the driver loads, & if everything works properly, PLX driver debug messages should appear. Debug View also allows capturing to a file, & other features. The output should be similar to the snapshots below.



The screenshot shows the DebugView application window titled "DebugView on \\PLX-INTEL-PC (local)". The window has a menu bar with "File", "Edit", "Capture", "Options", "Computer", and "Help". Below the menu bar is a toolbar with various icons. The main area displays a list of debug messages from the PLX driver, numbered 2 through 50. The messages include driver version information, registry path, device creation details, and PNP messages.

```
# Debug Print
2 Plx9656: <=====>
3 Plx9656: PLX driver v7.00 (32-bit) - built on Mar 22 2013 10:32:39
4 Plx9656: Driver supports WDM v6.0
5 Plx9656: OS is Windows 7 (WDM v6.1) or higher
6 Plx9656: Registry path = "\\REGISTRY\\MACHINE\\SYSTEM\\ControlSet001\\services\\Plx9656"
7 Plx9656: ...driver loaded
8
9 Plx9656: Created Device (\\Device\\Plx9656_v700-0)...
10 Plx9656: Create Win32 symbolic link (\\DosDevices\\Plx9656_v700-0)...
11 Plx9656: Attached device object to stack
12         Functional: 86CA7940
13         Physical   : 85ADA848
14         Lower      : 85ADA848
15
16 Plx9656: Received PNP Message (IRP=86386920) ==> IRP_MN_QUERY_LEGACY_BUS_INFORMATION
17 Plx9656: Forwarded IRP to next lower driver
18 Plx9656: ...Completed message
19
20 Plx9656: Received PNP Message (IRP=86386920) ==> IRP_MN_FILTER_RESOURCE_REQUIREMENTS
21 Plx9656: Forwarded IRP to next lower driver
22 Plx9656: ...Completed message
23
24 Plx9656: Received PNP Message (IRP=86386920) ==> IRP_MN_START_DEVICE
25
26 Plx9656: Received PNP Message (IRP=8612D378) ==> IRP_MN_QUERY_CAPABILITIES
27 Plx9656: Forwarded IRP to next lower driver
28 Plx9656: ...Completed message
29 Plx9656: Device information - PCI bus 04, slot 00, function 0
30 Plx9656: Resource list contains 9 descriptors
31         Resource 00
32             Type      : Memory
33             PCI BAR 0: FEOF600
34             Phys Addr: FEOF600
35             Size      : 00000200 (512 Bytes)
36             Property  : Non-Prefetchable 32-bit
37             Kernel VA: 8CA24600
38         Resource 01
39             Type: Device Private Data (ignoring - reserved for system use)
40         Resource 02
41             Type      : I/O
42             PCI BAR 1: 00001000
43             Phys Addr: 00001000
44             Size      : 00000100 (256 Bytes)
45         Resource 03
46             Type: Device Private Data (ignoring - reserved for system use)
47         Resource 04
48             Type      : Memory
49             PCI BAR 2: FD000000
50             Phys Addr: FD000000
```

```
DebugView on \\SAM-ABUNASSAR (local)
File Edit Capture Options Computer Help
# Debug Print
1
2 PlxSvc: <=====>
3 PlxSvc: PLX Service driver v7.00 (32-bit) - built May 10 2012 10:26:59
4 PlxSvc: Registry path = "\REGISTRY\MACHINE\SYSTEM\ControlSet001\services\PlxSvc"
5 PlxSvc: Create Device Object (\Device\PlxSvc_v700)
6 PlxSvc: Create Win32 symbolic link (\DosDevices\PlxSvc_v700)
7 PlxSvc: ACPI Probe - ACPI is v2.0 or higher (rev=2)
8 PlxSvc: ACPI Probe - 'RSD PTR ' found at 000FEB00
9 PlxSvc: ACPI Probe - RSD table at 000FD203 has 10 entries
10 PlxSvc: ACPI Probe - FACP table at 000FD2DB
11 PlxSvc: ACPI Probe - SSDT table at FFFD86F6
12 PlxSvc: ACPI Probe - APIC table at 000FD443
13 PlxSvc: ACPI Probe - BOOT table at 000FD4B5
14 PlxSvc: ACPI Probe - ASF! table at 000FD4DD
15 PlxSvc: ACPI Probe - MCFG table at 000FD544
16 PlxSvc: ACPI Probe - HPET table at 000FD582
17 PlxSvc: ACPI Probe - SSDT table at 7FE86C40
18 PlxSvc: ACPI Probe - SSDT table at 7FE87049
19 PlxSvc: ACPI Probe - SSDT table at 7FE87452
20 PlxSvc: ACPI Probe - PCIe ECAM at 00000000_F0000000
21 PlxSvc: Scan for devices...
22 PlxSvc: Add - 2770 8086 [b:00 s:00 f:0]
23 PlxSvc: Add - 2771 8086 [b:00 s:01 f:0]
24 PlxSvc: Add - 27D0 8086 [b:00 s:1c f:0]
25 PlxSvc: Add - 27D2 8086 [b:00 s:1c f:1]
26 PlxSvc: Add - 27C8 8086 [b:00 s:1d f:0]
27 PlxSvc: Add - 27C9 8086 [b:00 s:1d f:1]
28 PlxSvc: Add - 27CA 8086 [b:00 s:1d f:2]
29 PlxSvc: Add - 27CB 8086 [b:00 s:1d f:3]
30 PlxSvc: Add - 27CC 8086 [b:00 s:1d f:7]
31 PlxSvc: Add - 244E 8086 [b:00 s:1e f:0]
32 PlxSvc: Add - 27DE 8086 [b:00 s:1e f:2]
33 PlxSvc: Add - 27B8 8086 [b:00 s:1f f:0]
34 PlxSvc: Add - 27DF 8086 [b:00 s:1f f:1]
35 PlxSvc: Add - 27C0 8086 [b:00 s:1f f:2]
36 PlxSvc: Add - 27DA 8086 [b:00 s:1f f:3]
37 PlxSvc: Add - 0141 10DE [b:01 s:00 f:0]
38 PlxSvc: Add - 1677 14E4 [b:82 s:00 f:0]
39 PlxSvc: Device Scan: 17 device(s) found
40 PlxSvc: Allocate common buffer...
41 PlxSvc: Attempt to allocate physical memory (8 Kb)
42 PlxSvc: Allocated physical memory
43 PlxSvc: CPU Phys Addr: 7fe71000
44 PlxSvc: Bus Phys Addr: 7fe71000
45 PlxSvc: Kernel VA : A183F000
46 PlxSvc: Size : 00002000 (8192 bytes)
47 PlxSvc: Cacheable? : Yes
48 PlxSvc: ...driver loaded
```



## 2.2 Unable to Select a PLX Device in Windows Due to Security Issues

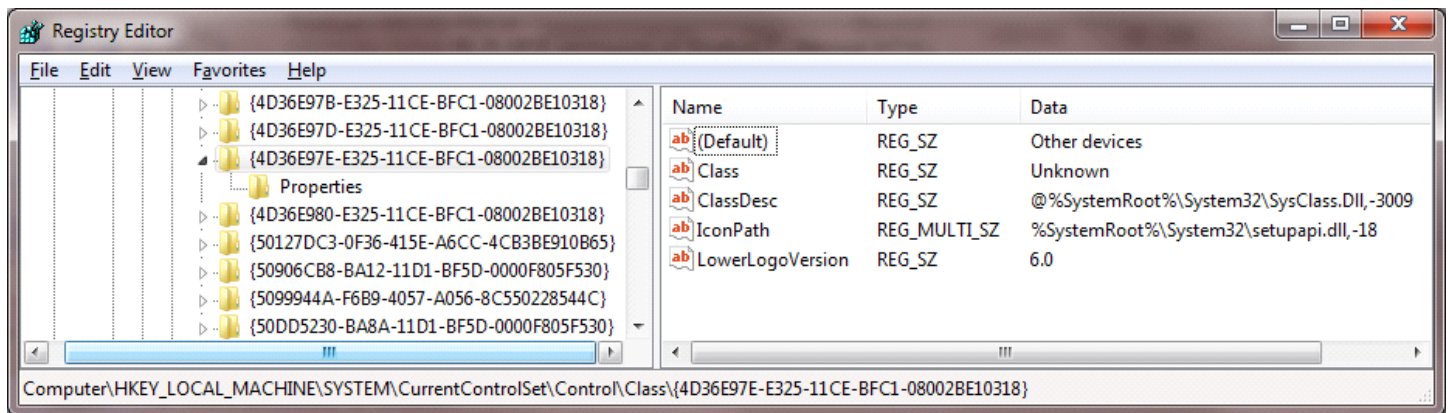
### PROBLEM & SYMPTOM

In some rare cases, applications using the PLX API are unable to open connection to a PLX device with **PlxPci\_DeviceOpen**. PLX has worked with a few customers who have reported such an issue. After investigation, this was narrowed down to a security setting in the system. The symptom occurs when the PLX API calls **CreateFile** to open a HANDLE to an installed PLX driver, which results in an extended error of ERROR\_ACCESS\_DENIED.

Another symptom is that the device is selected successfully though **PlxPci\_DeviceOpen**, but subsequent calls to some other PLX API calls, such as **PlxPci\_PciBarSpaceRead** or **PlxPci\_NotificationXxx**, return an error of **Unsupported Function**. In this case, the API is able to select the device via the PLX PCI/PCIe Service Driver, which is the last driver the API attempts to contact. The symptom manifests itself when certain API calls return Unsupported since the Service driver only supports a subset of the full PLX API. This issue doesn't apply to the service driver since it's not a Plug 'n' Play driver.

The problem likely starts due to some erroneous software/INF installation which modified the default security setting for the "Unknown" class of devices. The "Unknown" class is one of the pre-defined Windows classes for devices and determines where devices show up in Device Manager (e.g. Battery, HID device, Display, Network, etc.). PLX has always set SDK drivers, since they are generic, to be registered under the "Unknown" class in the INF. This is why the drivers are listed under "**? Other Devices**" in Device Manager.

The security setting may show up in the registry under either the Device Class (governs all devices of that class) or the device-specific registry entry. For example, here's the "Other Devices" class:



### Related links:

System Classes: [http://msdn.microsoft.com/en-us/library/windows/hardware/ff553428\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/hardware/ff553428(v=vs.85).aspx)

Vendor Classes: [http://msdn.microsoft.com/en-us/library/windows/hardware/ff553426\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/hardware/ff553426(v=vs.85).aspx)

Access Control Lists: [http://msdn.microsoft.com/en-us/library/aa374872\(v=VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa374872(v=VS.85).aspx)

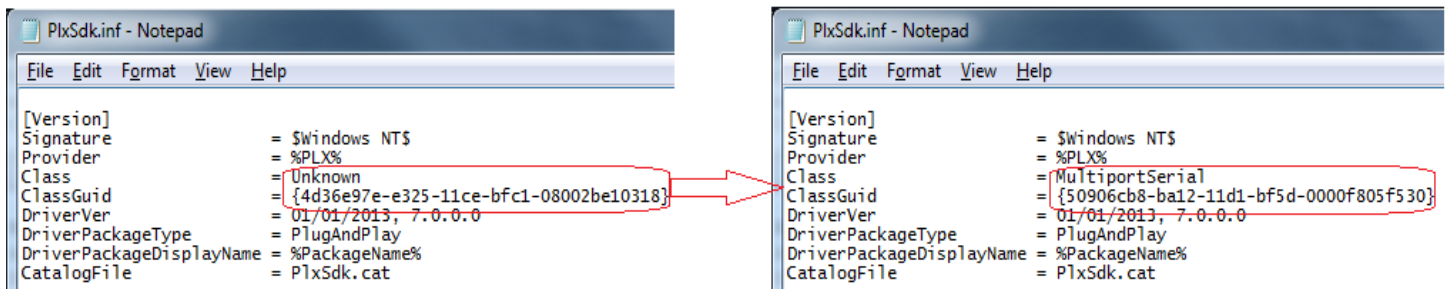
Controlling Device Access: [http://msdn.microsoft.com/en-us/library/ff542063\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/ff542063(v=vs.85).aspx)

SDDL for Device Objects: [http://msdn.microsoft.com/en-us/library/ff563667\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/ff563667(v=vs.85).aspx)

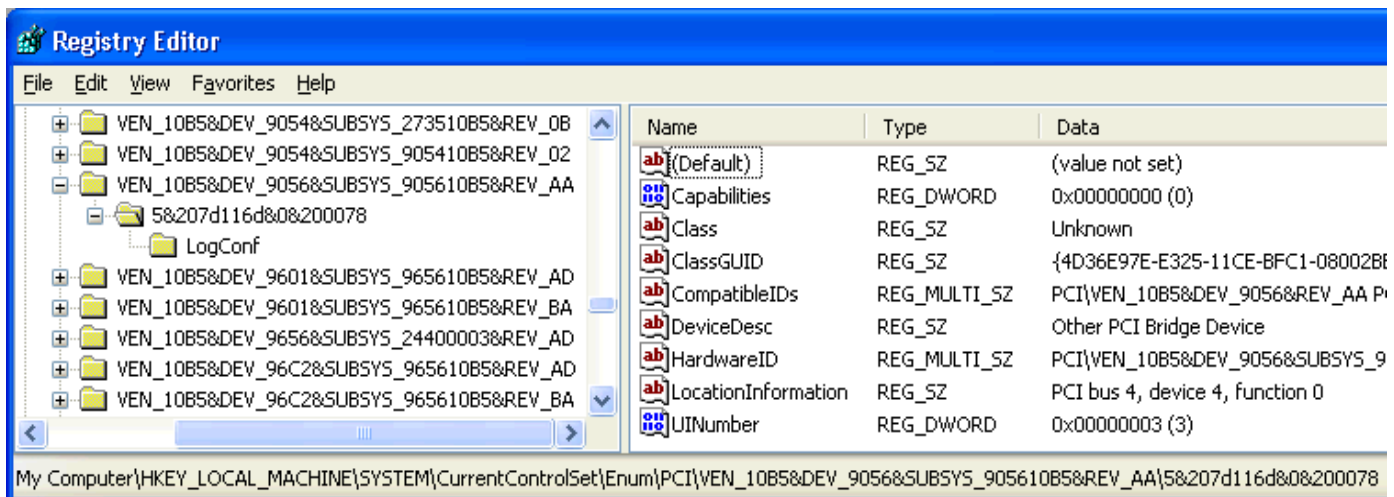
Secure Device Installations: [http://msdn.microsoft.com/en-us/library/ff540212\(v=VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ff540212(v=VS.85).aspx)

## POSSIBLE SOLUTIONS

1. In all cases, a complete re-install of the OS resulted in removal of the problem. This results in the security setting for the “Unknown” class of devices reverting back to default. If System Restore is enabled in the system, restoration of the registry to a date prior to the erroneous software modification of the security descriptor may resolve the issue. Note that re-installing the erroneous software/INF will result in a return of the problem.
2. **[Recommended]** In the INF, modify the class to one that matches the type of device that will ship. In general, this step should be followed anyway. This change then avoids the security descriptor setting for the **Unknown** device class. Uses either a Microsoft provided Vendor class or [refer to MSDN](#) for creating a custom OEM class. Creating a custom class is beyond the scope of this document, but essentially involves adding a **ClassInstall32** section to the INF. Below is an example of changing to a Multi-port Serial Adapter class of devices.



3. In the registry, check both the **Unknown** class (**HKLM\SYSTEM\CurrentControlSet\Control\Class\{4D36E97E-E325-11CE-BFC1-08002BE10318}**) location & the registry entry for the device itself. The device location will depend upon its Dev/VenID, etc., but will be a sub key under **HKLM\SYSTEM\CurrentControlSet\Enum\PCI**. Here’s an example of a PLX 9056 RDK in a system:



**If there is some sort of “Security” key under either of these, delete them & reboot & that may solve the problem. Most likely it’s under the class key.**

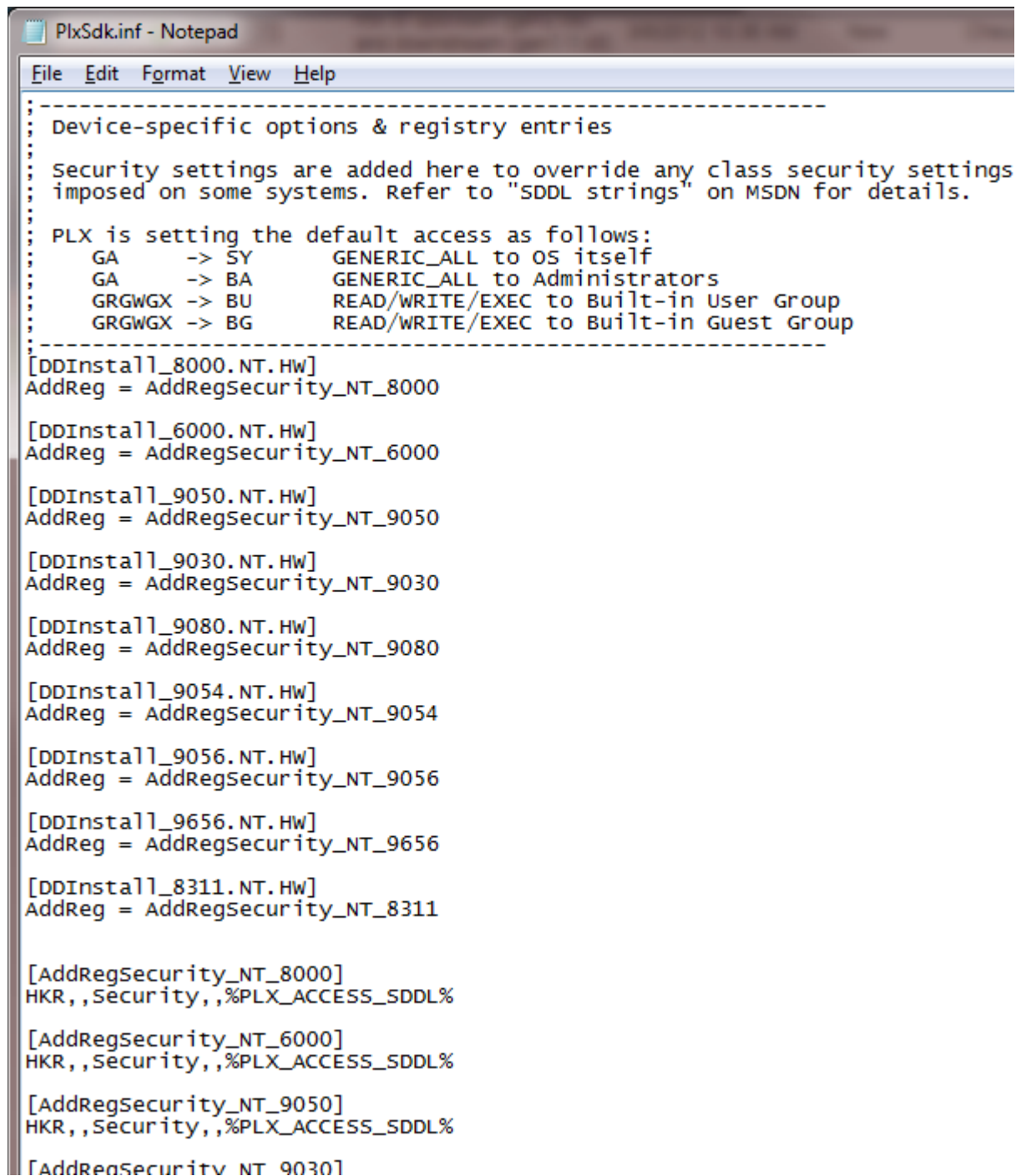
4. Starting with PLX SDK v6.50, PLX added a device security key in the INF so that it’s added for all device installations. Since PLX was never able to reproduce this issue in-house, there is no guarantee this setting will have any effect on the problem. The security descriptor of the parent class may override device-specific security. Below are snapshots from the PLX INF to demonstrate the changes:

```

;-----
; String information
;-----

[Strings]
InstallDisk      = "PLX Installation Disk"
PLX              = "PLX Technology, Inc."
PLX_ACCESS_SDDL  = "D:P(A;;;GA;;;SY)(A;;;GRGWGX;;;BA)(A;;;GRGWGX;;;BU)(A;;;GA;;;BG)"

```



```

PlxSdk.inf - Notepad
File Edit Format View Help
;-----
; Device-specific options & registry entries
;-----
; Security settings are added here to override any class security settings
; imposed on some systems. Refer to "SDDL strings" on MSDN for details.
;
; PLX is setting the default access as follows:
;   GA      -> SY      GENERIC_ALL to OS itself
;   GA      -> BA      GENERIC_ALL to Administrators
;   GRGWGX  -> BU      READ/WRITE/EXEC to Built-in User Group
;   GRGWGX  -> BG      READ/WRITE/EXEC to Built-in Guest Group
;-----
[DDInstall]_8000.NT.HW]
AddReg = AddRegSecurity_NT_8000

[DDInstall]_6000.NT.HW]
AddReg = AddRegSecurity_NT_6000

[DDInstall]_9050.NT.HW]
AddReg = AddRegSecurity_NT_9050

[DDInstall]_9030.NT.HW]
AddReg = AddRegSecurity_NT_9030

[DDInstall]_9080.NT.HW]
AddReg = AddRegSecurity_NT_9080

[DDInstall]_9054.NT.HW]
AddReg = AddRegSecurity_NT_9054

[DDInstall]_9056.NT.HW]
AddReg = AddRegSecurity_NT_9056

[DDInstall]_9656.NT.HW]
AddReg = AddRegSecurity_NT_9656

[DDInstall]_8311.NT.HW]
AddReg = AddRegSecurity_NT_8311

[AddRegSecurity_NT_8000]
HKR,,Security,,%PLX_ACCESS_SDDL%

[AddRegSecurity_NT_6000]
HKR,,Security,,%PLX_ACCESS_SDDL%

[AddRegSecurity_NT_9050]
HKR,,Security,,%PLX_ACCESS_SDDL%

[AddRegSecurity_NT_9030]

```

## 2.3 Increase 8311 DMA Performance

The 8311 device is actually a PLX 9056 with a PLX 8111 PCIe-to-PCI bridge in a single die. The introduction of the 8111 device does introduce some latency & affects performance, especially for PCIe-to-Local bus DMA operations. To increase performance, some settings may be modified. Below are suggested settings to modify to achieve higher throughput. These settings apply to Forward-mode operation (*PCIe-to-PCI*).

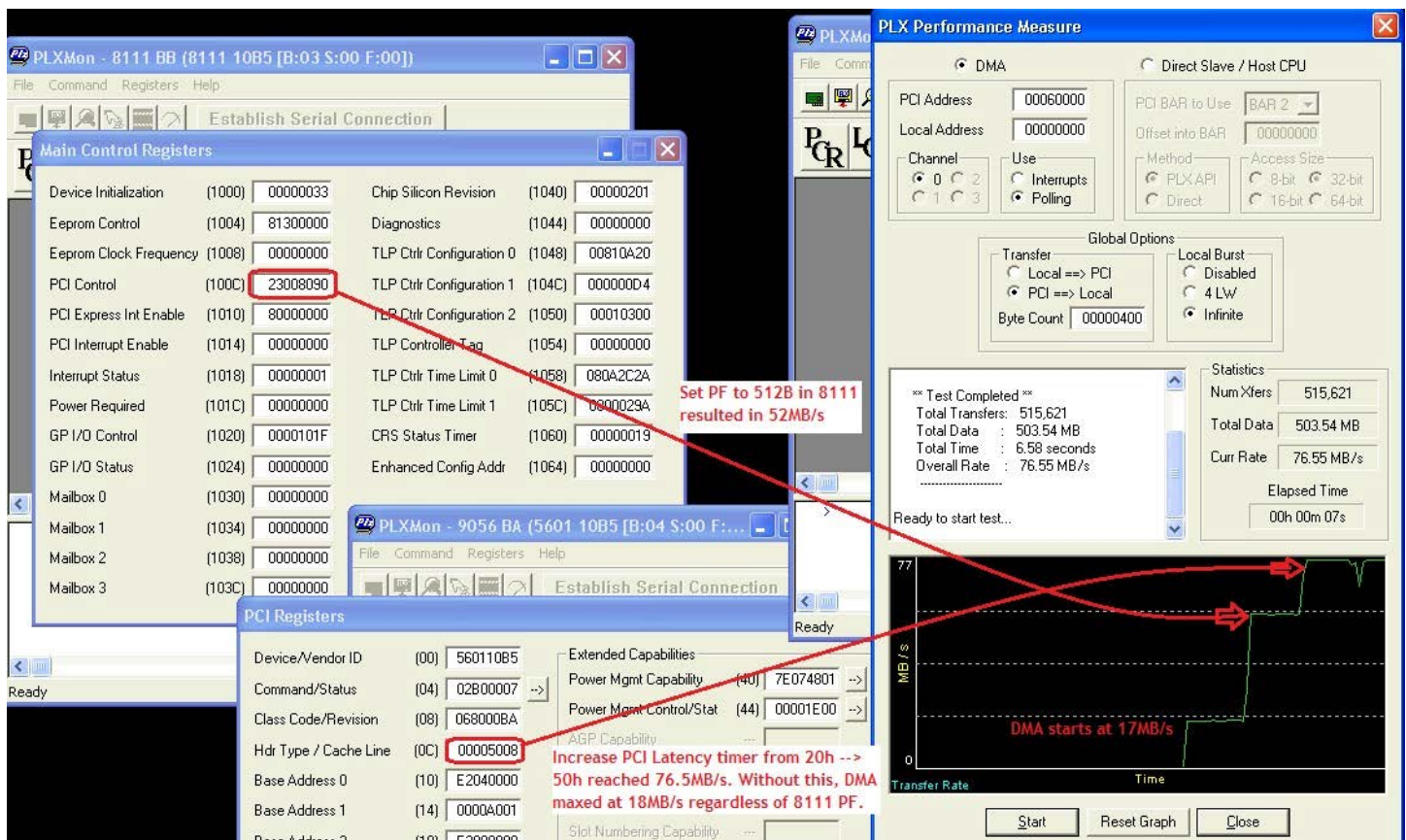
### 8111

- Enable blind prefetch (*48h[0] & 100Ch[29:27]*) and set to a reasonable value, like 512B
- Increase the secondary PCI latency timer (*18h[31:24]*)

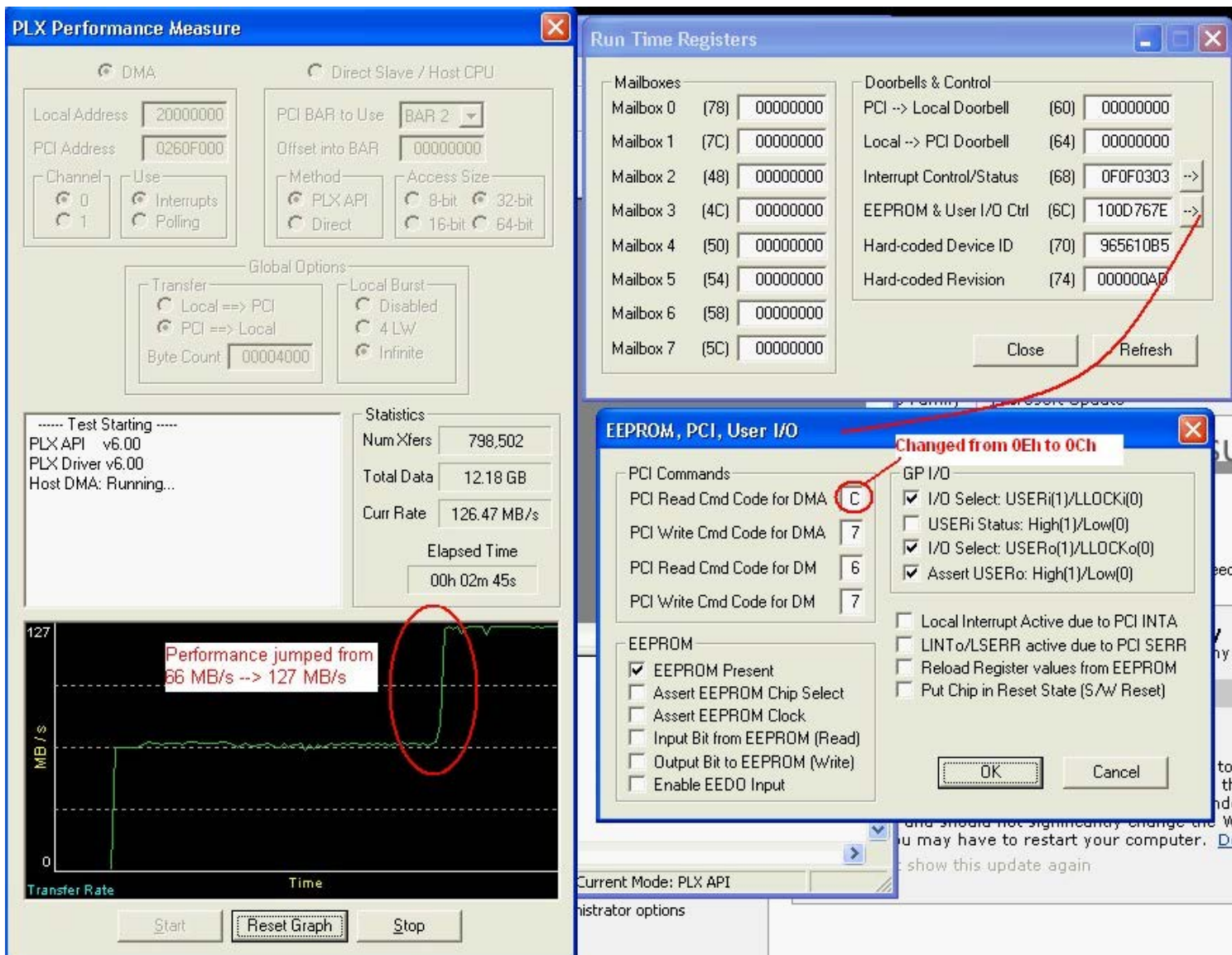
### 9056/8311

- Increase the PCI latency timer (*04h[15:8]*) to a larger value, like 50h.
- Disable the DMA latency & pause timers (*08h[17:0]*).
- Change the DMA read command code (*6Ch[3:0]*) from 0Eh (MemRead Line)→0Ch (MemRead Multiple).

Below are some sample snapshots demonstrating the changes & performance impact on a PLX 8311 RDK.







## 2.4 8311 64-bit DMA Issues

### PROBLEM & SYMPTOM

On some systems, DMA may fail when *PlxPci\_DmaTransferUserBuffer* is used with an 8311. This API call uses SGL mode of the DMA engine of the 9056/8311 to transfer to/from the individual pages of the provided user-mode buffer. On systems with large amounts of RAM (eg 3GB+), some of these user pages may reside in memory above 4GB (64-bit address). The PLX driver will detect this & use the Dual-Address Cycle (DAC) mode of the 9056 DMA to support 64-bit addressing.

Since the 8311 includes an 8111, the 8111 has an issue with forwarding DAC cycles upstream. As a result, the DMA operation will block & fail when a user page is in 64-bit space.

### SOLUTION

This issue is similar to erratum #5 of the 8111. To allow the 8111 to support DAC cycles, a PECS/SPI EEPROM is required in most cases. The fix involves setting PCI 24h[0] to enable 64-bit Prefetchable Memory, which can only be set by EEPROM. The other fix requires setting PCI 28h at runtime to a value larger than the upper 32-bits of system RAM. A value such as 20h should work for most systems. With these settings, the 8111 should then properly forward DMA DAC cycles from the 9056.

The image shows two windows from a PCI configuration utility. The '8111 EEPROM' window on the left has a 'Registers' section with a table:

Reg	Value	Bytes
0034	00000050	12 (2 regs)
0024	00000001	

Annotations in red and green text point to these values: 'Disable Power Management' for 0034 and 'Support 64-bit Prefetchable Memory' for 0024. To the right, the 'PCI Registers' window shows a list of registers with their values. The entry for 'PF Base (high 32-bits)' at offset (28) has the value 00000020, which is circled in red.

Register Name	Offset	Value
Device/Vendor ID	(00)	811110B5
Command/Status	(04)	00100007
Class Code/Revision	(08)	06040021
Hdr Type / Cache Line	(0C)	00010008
Base Address 0	(10)	00000000
Base Address 1	(14)	00000000
Prim/Sec Bus Num	(18)	00050504
I/O Limit/Base	(1C)	2200A0A0
Memory Limit/Base	(20)	E5F0E400
PF Memory Limit/Base	(24)	E6F0E600
PF Base (high 32-bits)	(28)	00000020
PF Limit (high 32-bits)	(2C)	00000000

## 2.5 8311 & 8111 Power Management Issue with Windows

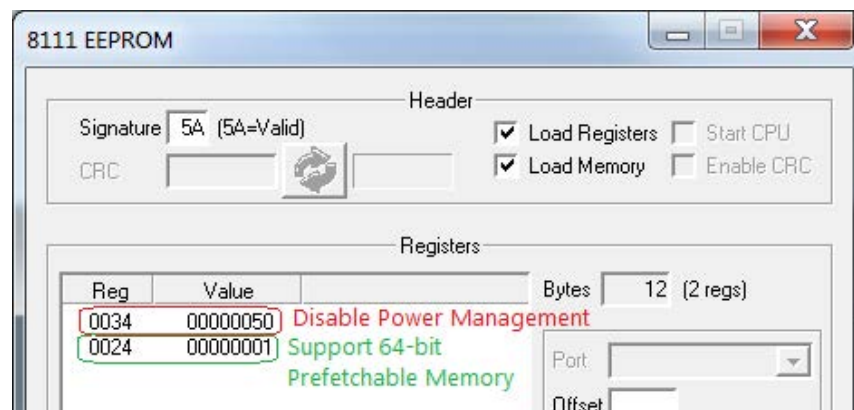
### PROBLEM & SYMPTOM

When an 8311 device is plugged into Windows system, the system may not boot or may crash when Windows loads the default PCI-to-PCI bridge driver for the 8111. This also applies to designs using an 8111 with an OEM endpoint.

### SOLUTION

This issue is related to a Power Management (PM) erratum in the 8111. The only known workaround is to disable PM in the 8111 via the SPI EEPROM. An SPI EEPROM must be present in the design to implement the fix.

To disable PM, add the value **50h** for offset **34h** in the SPI EEPROM. The PM Capability is located at 40h in the 8111, so placing a value of 50h in 34h “hides” this capability from the Operating System by bypassing it in the PCI Capabilities list.



Upon reboot, verify 50h is loaded into 34h.

PCI Registers			
Device/Vendor ID	(00)	811110B5	
Command/Status	(04)	00100007	-->
Class Code/Revision	(08)	06040021	
Hdr Type / Cache Line	(0C)	00010008	
Base Address 0	(10)	00000000	
Base Address 1	(14)	00000000	
Prim/Sec Bus Num	(18)	00050504	
I/O Limit/Base	(1C)	2200A0A0	
Memory Limit/Base	(20)	E5F0E400	
PF Memory Limit/Base	(24)	E6F0E600	
PF Base (high 32-bits)	(28)	00000000	
PF Limit (high 32-bits)	(2C)	00000000	
I/O L/B (high 16-bits)	(30)	00000000	
Capabilities Pointer	(34)	00000050	
Expansion ROM Base	(38)	00000000	

## 2.6 Windows PCI Express Native Mode & PLX Devices

### PROBLEM & SYMPTOM

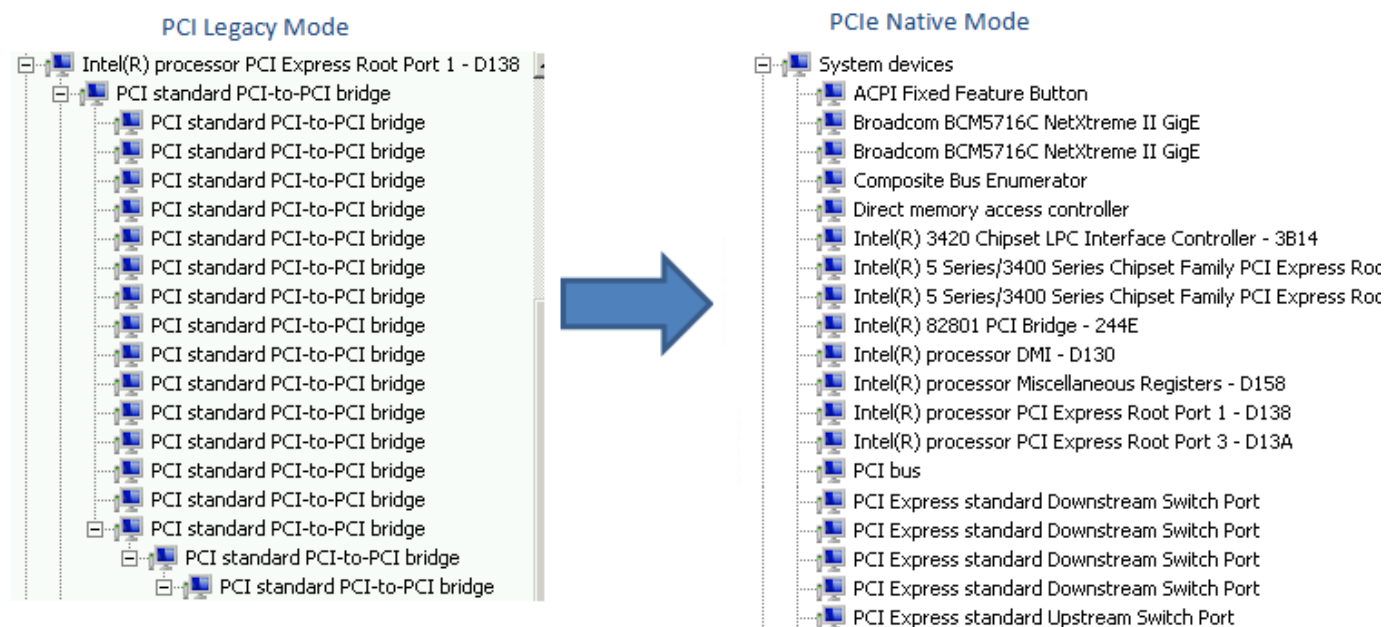
Starting with Windows Vista, Microsoft added native PCI Express support into the OS for PCIe-specific features, such as Hot Plug, PCIe Power Management, Advanced Error Reporting, PCIe Capability, etc. When Windows loads, it will launch into either **PCI Legacy Mode** or **PCI Express Native Mode**. This internal decision is based features exported by the platform BIOS/ACPI. In most cases, only high-end server based systems include ACPI support for Windows PCIe Native Mode. PCIe Native Mode is required to support PCIe features; otherwise, full PCIe support is disabled.

In most case, PLX devices will work fine with either Windows mode. For the following situations, if Windows is in PCIe Native Mode, it will blindly disable PLX devices & prevent a driver from loading. In Windows Device Manager, the only indication will usually be an error reported for the device with the message *“The device cannot start (Code 10)”*. The situations are:

- PLX 8111/8112 or 8114 in Reverse Mode (PCIe → PCI)
- PLX 8500 series Non-Transparent endpoint
- PLX 8600 series Non-Transparent endpoint if the chip either does not support or is not configured in NT P2P mode. This means the NT Endpoint is a direct child of the Upstream port without an intermediate Downstream port.

### DETECT WINDOWS MODE

There is no obvious way to detect which mode Windows is running in. One method PLX has noticed is in Device Manager, the description of PCIe upstream & downstream ports will be different between PCI Legacy and PCIe Native modes. In PCIe Native mode, switches are listed as *“PCI Express standard Upstream/Downstream Switch Port”*, whereas they are *“PCI standard PCI-to-PCI bridge”* in Legacy mode.

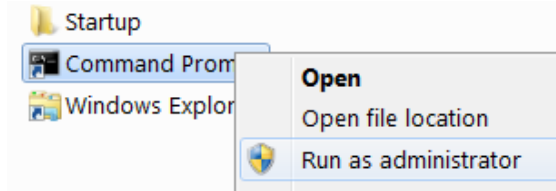




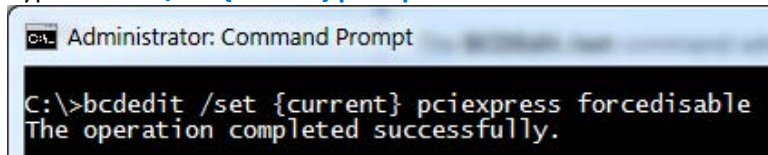
## SOLUTION

At this time, the only known solution is to always force Windows into PCI Legacy Mode. This is accomplished by setting a boot option via the BCD utility. The steps are:

1. Boot to Windows normally
2. Open a Command Prompt window (*with Administrator rights*)



3. Type “**bcedit /set {current} pciexpress forcedisable**”



4. Reboot

Windows should then boot to PCI Legacy Mode every time. This setting can easily be reversed with either of the commands “**bcdedit /deletevalue {current} pciexpress**” or “**bcdedit /set {current} pciexpress default**”.

**NOTE:** *Disabling Windows PCIe Native Mode will result in the disable of support for PCIe features, such as Hot Plug, etc.*

### **Related Links:**

Windows PCIe Native Mode: <http://msdn.microsoft.com/en-us/windows/hardware/gg487424>

BCD /set Command: [http://msdn.microsoft.com/en-us/library/windows/hardware/ff542202\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/hardware/ff542202(v=vs.85).aspx)

## 2.7 Programming Blank or No-Signature (5Ah) EEPROMs

### PROBLEM & SYMPTOM

All PLX devices support an optional EEPROM to store PLX register contents a chip will attempt to load on power up/reset. EEPROMs are generally recommended to support override of default register values or to work around issues. This section will provide options for dealing with issues related to programming blank EEPROM devices.

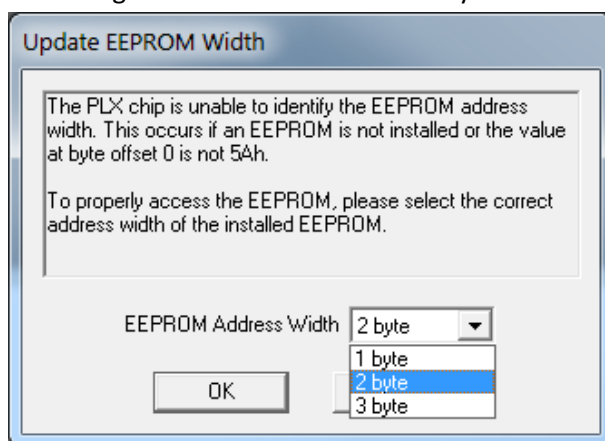
### SOLUTION

When an EEPROM is blank, it can be programmed easily using PLX software tools or an equivalent. For legacy PLX devices, such as 6000 & 9000 series, this procedure is generally trouble free. PLX tools such as *PlxCm*, *PLXMon GUI*, *PlxEep* support programming any of these devices. A PLX EEPROM API with complete source code is also provided in the PLX SDK for custom applications.

Newer PLX PCIe devices (e.g. 8100, 8500, 8600, 8700, etc) behave differently than legacy PLX devices with respect to EEPROM access. These devices contain an integrated EEPROM controller which software uses to access an EEPROM. Additionally, various EEPROM sizes are supported, requiring 1, 2, or 3 byte addressing. The EEPROM controller must be able to identify the EEPROM byte addressing in order to properly access it. This introduces a problem when an EEPROM is blank or doesn't contain a valid signature byte (5Ah at byte offset 0). During power up, the EEPROM controller probes the EEPROM device for 5Ah signature byte and, if detected, is then able to properly set the byte addressing. If no signature byte exists, the byte addressing in the EEPROM controller will usually default to 1-byte, preventing proper access to 2 & 3-byte parts.

The following solutions apply to programming a blank EEPROM of PLX PCIe 8000 parts.

- **[8700 & newer 8600 series]** These PLX devices support manual override of the EEPROM byte addressing in the memory controller. 260h[21] is the manual override bit & 260h[23:22] set the byte address width. 260h[21] must be set 1<sup>st</sup> & remain set to keep the override in effect. To program a blank EEPROM, perform one of the following depending upon the method used:
  - **[PLXMon GUI]** PLXMon will query the PLX chip & prompt the user if the byte addressing is not set with the dialog below. Select the correct byte addressing.



- **[PlxEep]** Use the '-w' option to set the byte addressing. For example, '*PlxEep -l MyEep.bin -w 2*'.
- **[PlxCm]** Manually issue register write commands to set the byte addressing.

```
// List devices & select upstream port
dev
dev XX (Refer to list for device number)

// Set bit 21 to override the EEPROM width
mmr 260 00200000

//
// Select one of the byte-addressing options below to match EEPROM
//
// Set 1 byte addressing ([23:22]=01b)
mmr 260 00600000

// Set 2 byte addressing ([23:22]=10b)
mmr 260 00A00000

// Set 3 byte addressing ([23:22]=11b)
mmr 260 00E00000

// EEPROM accesses should start to work
eep 0 0000005A

// To manually write 5Ah at offset 0, the following commands may be used.
// This assumes 2B addressing, so adjust bits [23:21] accordingly for 1B or 3B

// Prepare 32-bit data value
mmr 264 0000005A

// Issue write-enable (6) followed by write (2) command
mmr 260 00A0C000
mmr 260 00A04000
```

- **[8500 (Altair family only) & early 8600 series]** These devices do not support a byte address width override. The technique to resolve the issue is to send a special pattern to the EEPROM controller which will result in writing 0000\_005Ah to the 1<sup>st</sup> DW of the EEPROM. Upon reboot, the EEPROM controller will then be able to detect the address width. The register writes sent depend upon the EEPROM byte addressing. Use PlxCm to issue one of the following commands:
  - **[1-Byte]**  
eep 0 0000005A
  - **[2-Byte]**  
eep 0 00005A00  
eep 0 00000001

- **[3-Byte]**
  - eep 0 005A0000
  - eep 0 00000200
- **[8111 & 8112]** For these devices, the PLX chip reports detected EEPROM byte width in 1004h[24:23]. If the width is not detected, EEPROM programming software must be informed of the byte addressing to use. At this time, only the *PLXMon GUI* & *PlxEep* (via '-w' option) support this feature for these devices. Custom software can refer to the PLX API *PlxPci\_EepromSetAddressWidth()*.

## 2.8 EEPROM Protection

### PROBLEM & SYMPTOM

Most EEPROMs support write-protection of some or all regions. PLX software assumes EEPROM write-protection is disabled and does not clear the protection if it is set. If protection is enabled, this will block EEPROM writes.

### SOLUTION

**PlxCm** may be used to issue manual commands to view, set, or clear the EEPROM protection. Some sample commands are provided below. To program a protected EEPROM, for example, issue the commands to disable protection, followed by **eep\_load** to program the EEPROM. Protection can then be restored afterwards.

- **Disable EEPROM protection**

```
// Enable Write latch
mmr 260 0000C000
sleep 10
// Clear WPEN & BP[1:0]
mmr 260 00002000
sleep 10
// Disable Write latch
mmr 260 00008000
```

- **Enable EEPROM protection**

```
// Enable Write latch
mmr 260 0000C000
sleep 10
// Set WPEN & BP[1:0]=11
mmr 260 8c002000
sleep 10
// Disable Write latch
mmr 260 00008000
```

- **View current EEPROM status register (in [31:24])**

```
// Send Read Status Register command
mmr 260 02005000
sleep 10
mmr 260
```

## 2.9 Dealing With a Corrupt EEPROM

### PROBLEM & SYMPTOM

In some cases, EEPROM contents may get corrupted or programmed with invalid values. These may result in an unstable system, the PLX chip not recognized, or other issues. This section will provide options for programming blank or corrupt EEPROM devices.

### SOLUTION

The following list possible solutions to get around a corrupt EEPROM. Some options involve handling the EEPROM chips when the system is live. You may perform these at your own risk & PLX is not responsible for any damage caused by these methods. Usually, if done carefully, there is little risk of issues.

- Use an external EEPROM programmer to reprogram the EEPROM if it is removable
- For newer PLX PCIe switches, if possible, use an I2C connection with PLX software to reprogram the EEPROM.
- **[At your own risk]** If EEPROM is socketed, remove it, power up the system with the PLX chip, replace the EEPROM & reprogram it with software. Refer to the [Programming Blank or No-Signature \(5Ah\) EEPROMs](#) section.
- **[At your own risk]** Before powering up the system, temporarily short the EEDO pin of the EEPROM to ground. A moment after the system boots, remove the short. This will make the PLX chip think the EEPROM is blank or non-existent & will revert to default values. Reprogram the EEPROM with software. Refer to the [Programming Blank or No-Signature \(5Ah\) EEPROMs](#) section.

### 3. FAQ

---