PCI/PCI Express Configuration Space Access

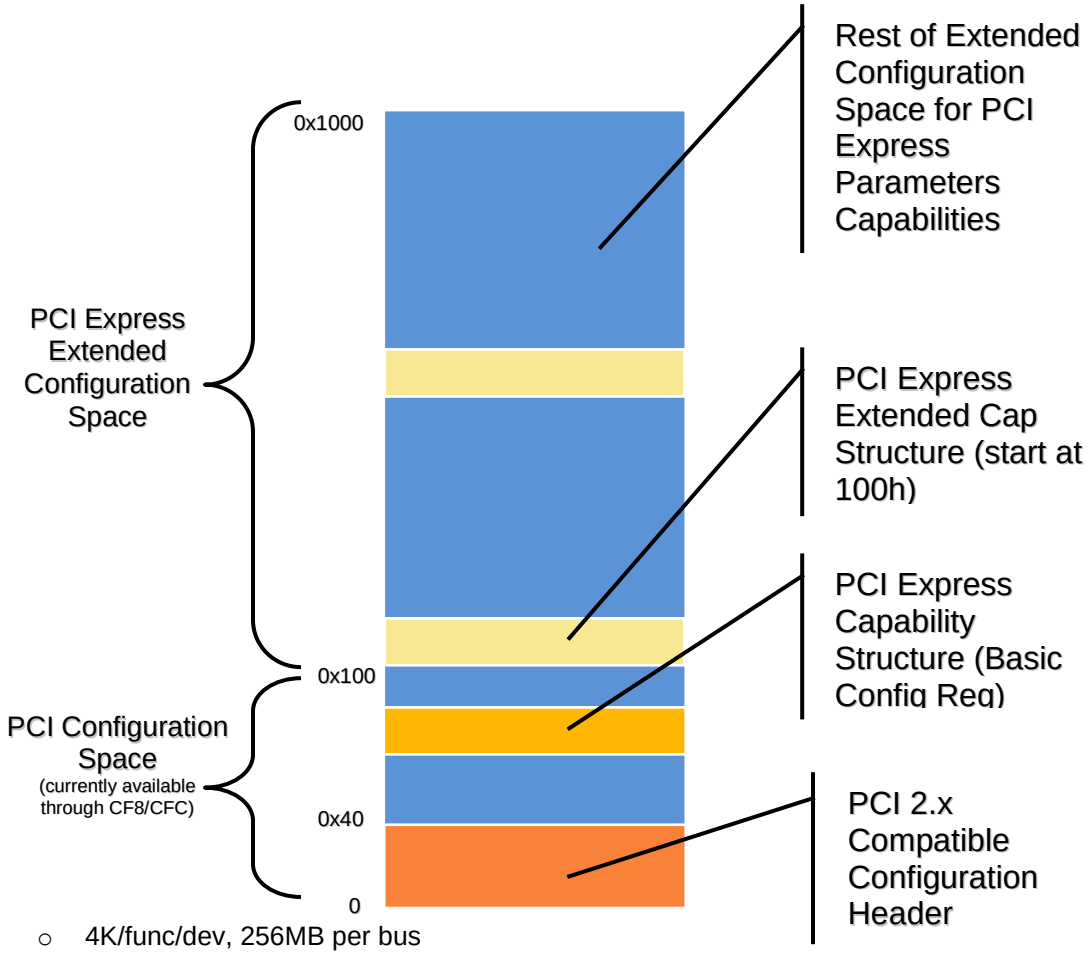
Advanced Micro Devices, Inc.

May 2008

1. Introduction

PCI devices have a set of registers referred to as 'Configuration Space' and PCI Express introduces Extended Configuration Space for devices. Configuration space registers are mapped to memory locations. Device drivers and diagnostic software must access the configuration spaces and operating systems have APIs to allow access to device configuration space. When the operating system does not have access methods defined or APIs for memory mapped configuration space requests, the driver or diagnostic software has the burden to access the configuration space in a manner that is compatible with the operating system underlying access rules. In all systems, device drives are encouraged to use APIs provided by the operating system to access the configuration space of the device. For systems that do not implement a processor-architecture specific firmware interface standard that allows access to configuration space, PCI Express defines an Enhanced Configuration Access mechanism (ECAM). This whitepaper outlines the best coding practices for device drivers and diagnostic software developers to use, when accessing PCI/PCI Express Configuration Space. Separate sections are provided for Windows, Solaris, and Linux operating systems.

1.1 PCI/PCI Express Configuration Space Memory Map



- 4K/func/dev, 256MB per bus
- Flat memory mapped access
- Firmware indicates memory base
- First 256 bytes PCI compatible
- Do not assume CF8/CFC available for extended space access

2. Windows Operating System

2.1 Windows Overview

In Windows, access is facilitated for drivers which are part of the driver stack which owns the configuration space.

Configuration space access has been provided in a uniform way since the Windows 2000 product was introduced. Complete information is contained in the following KB article:

http://support.microsoft.com/kb/253232

2.2 Coding practices

To access configuration space a PnP I/O Request Packet (IRP) is sent from an in-stack driver at passive level. This writes or reads a buffer of data at a specified offset. Parameters for the call are set up as follows:

```
irpStack = IoGetNextIrpStackLocation( irp );
if (ReadOrWrite == 0) {
      irpStack->MinorFunction = IRP_MN_READ_CONFIG; }
else { irpStack->MinorFunction = IRP_MN_WRITE_CONFIG; }
irpStack->Parameters.ReadWriteConfig.WhichSpace =PCI_WHICHSPACE_CONFIG;
irpStack->Parameters.ReadWriteConfig.Buffer = Buffer;
irpStack->Parameters.ReadWriteConfig.Offset = Offset;
irpStack->Parameters.ReadWriteConfig.Length = Length;
```

After the call, in the case of a read operation the data will be in the specified buffer area.

2.3 Windows Vista and Windows Server 2008

Extended Configuration space access is provided in Vista and subsequent operating systems. The access method (PnP IRP) has been modified to cover the extended configuration space so the driver writers have the same interface as before.

3. Linux

3.1 Linux Overview

The Linux PCI subsystem provides a bunch of functions for PCI configuration space access.

- pci_{read,write}_config_byte()
- pci_{read,write}_config_word()
- pci_{read,write}_config_dword()

These functions read or write data of byte-, word- or double-word-size from or to PCI configuration space.

In the end, the kernel decides which configuration space access method (IO or MMIO) is used.

3.2 Coding practices

This code snippet demonstrates how a device driver can use some of the functions mentioned above to access the configuration space of a PCI device.

```
void foo(struct device *dev)
{
      struct pci_dev *pdev = to_pci_dev(dev);
      u8 rega;
      u32 regb;

      ...

      pci_read_config_byte(pdev, REG_FOO_1, &rega);
      pci_read_config_dword(pdev, REG_FOO_2, &regb);
```

```
        ...

        pci_write_config_word(pdev, REG_FOO_3, 0x0815);

        ...

}
```

3.3 MMIO access with different Linux Kernel versions

Support for MMIO access for PCI configuration space depends on the Linux Kernel version and configuration, and the existence of an MCFG ACPI table.

IO access to PCI configuration space is always possible and will be used if MMIO access is not available. To access extended configuration space (between byte 256 and 4095) MMIO access is required.

To enable the MMIO access method you have to enable the following kernel configuration options.

For ARCH=x86_64 on all recent kernel versions (at least versions later than and including 2.6.12)

CONFIG_PCI_MMCONFIG

"Bus options"->"PCI support"->"Support mmconfig PCI config space access"

For ARCH=i386 and kernel versions later than and including version 2.6.24

CONFIG_PCI_MMCONFIG

"Bus options"->"PCI support"->"Support mmconfig PCI config space access"

For ARCH=i386 and kernel versions prior to and including 2.6.23 (at least down to 2.6.12)

CONFIG_PCI_GOMMCONFIG or CONFIG_PCI_GOANY

"Bus options"->"PCI support"->"PCI access mode"->"MMConfig" or
"Bus options"->"PCI support"->"PCI access mode"->"Any"

Note: In both cases Linux supports the IO access method as a fallback if MMIO access is not possible (e.g. if MCFG ACPI table is missing).

A kernel (with version prior to and including 2.6.25) with enabled MMIO access method is able to do MMIO access if and only if the BIOS provides a valid MCFG ACPI table and if the configuration space is "E820-reserved". If these requirements are not met, IO access is used and the kernel logs an error message like this

```
PCI: BIOS Bug: MCFG area at e0000000 is not E820-reserved
PCI: Not using MMCONFIG.
PCI: Using configuration type 1
```

Beginning with kernel version 2.6.25 Linux always uses the IO access method to access configuration space below byte 256. Thus MMIO access is used only for extended configuration space.

For upcoming kernels the following changes are pending:

- The restriction that configuration space must be "E820-reserved" will partially be lifted.
- For AMD family 10h CPUs, MMIO access will also work without an MCFG ACPI table as long as MSR C001_0058 (the MMIO Configuration Base Address Register) is properly configured.

3.4 Access to NB devices of AMD CPUs

The Linux Kernel uses always IO type when accessing configuration space of NB devices on AMD family 0fh CPUs,

Extended configuration space access on AMD family 10h CPUs is not possible with kernel versions prior to and including 2.6.25. It is highly probable that the next kernel version (i.e. 2.6.26) will support extended configuration space access.

3.5 Accessing configuration space from user space

PCI configuration space access from user space is possible via sysfs. This is done through a "config"-attribute provided with each PCI device sysfs-representation.

Tools such as pciutils (e.g. lspci, setpci) make use of this interface.

Example:

```
 # lspci  | grep VGA
01:00.0 VGA compatible controller: ATI Technologies Inc RV370 [Sapphire X550 Silent]

# hexdump /sys/bus/pci/devices/0000:01:00.0/config
0000000 1002 5b63 0107 0010 0000 0300 0010 0080
0000010 0008 d000 a001 0000 0000 fe9f 0000 0000
0000020 0000 0000 0000 0000 0000 0000 148c 2134
0000030 0000 fe9c 0050 0000 0000 0000 010a 0000
0000040 0000 0000 0000 0000 0000 0000 148c 2134
0000050 5801 0602 0000 0000 8010 0001 0260 012c
0000060 0800 0000 1d01 0000 0040 1101 0000 0000
0000070 0000 0000 0000 0000 0000 0000 0000 0000
0000080 0005 0080 0000 0000 0000 0000 0000 0000
0000090 0000 0000 0000 0000 0000 0000 0000 0000
*
0000100 0001 0001 0000 0000 0000 0000 2011 0006
0000110 0000 0000 0000 0000 0000 0000 0001 0400
0000120 000f 0000 0000 0107 e734 ad07 0000 0000
0000130 0000 0000 0000 0000 0000 0000 0000 0000
*
0001000
```

It is highly recommended to use that interface when accessing configuration space from user space. It is, of course, possible to access PCI configuration space via /dev/mem but this will bypass the kernel API and might cause severe problems including a possible system hang.

4. Solaris

4.1 Overview

Solaris provides device driver developers with a set of DDI routines that manage access to PCI/PCI-X/PCI-E Configuration space, including PCI-E Extended Configuration space.

- `pci_config_setup`(9F). Sets up resources and returns a handle for configuration space accessors
- `pci_config_get`XX/`pci_config_put`XX (9F). Read or write single datum of various sizes to the memory-mapped Configuration space. XX is one of `8`, `16`, `32`, or `64`, signifying datum's width in bits.
- `pci_config_teardown`(9F). Destroys resources allocated by `pci_config_setup`.

4.2 Coding practices

Here is a simple example demonstrating use of these interfaces to read 1 byte from Configuration space:

```
#include <sys/types.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>

int
xxx_read_pcicfg(dev_info_t *dip, off_t offset, uint8_t *val)
{
      ddi_acc_handle_t handle;

      if (pci_config_setup(dip, &handle) != DDI_SUCCESS)
            return DDI_FAILURE;

      *val = pci_config_get8(handle, offset);

      pci_config_teardown(&handle);

      return DDI_SUCCESS;
}
```

Note that even though Solaris DDI provides 64-bit (QWORD) accessors to PCI Configuration space, their use is discouraged (see, for example, Implementation Note to section 7.2.2.2 of the PCI Express Base Specification, rev 2.0). Instead, developers should use 2 32-bit (DWO

Appendix A: References

Specifications:

PCI Express Base Specification v1.0a http://www.pcisig.com
PCI Firmware Specification v3.0 http://www.pcisig.co