Notes

By Craig Hackney

Introduction

In typical PCIe based systems, PCIe buses are enumerated and resources allocated to each PCIe endpoint device during system initialization. Due to limitations in the enumeration and resource allocation algorithms, once initialized, the PCIe topology is fixed, meaning no new endpoint devices or switches may be connected to the system.

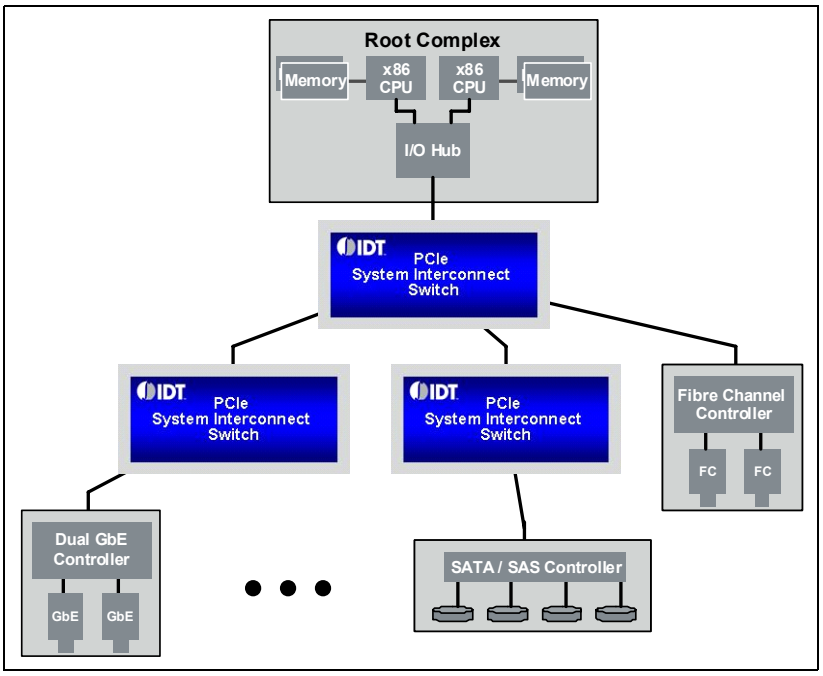

Figure 1 Typical PCIe Topology

The IDT PCIe Hot-Swap driver is a loadable Linux module that employs IDT proprietary enumeration, resource allocation, and device detection algorithms, allowing PCIe endpoint devices and switches to be connected to or disconnected from a system at runtime without compromising the operational state of the other PCIe devices in system. Using Figure 1 as reference, if the dual GigE controller is disconnected from the system and subsequently reconnected to a port on a different switch, the operational state of the Fibre Channel controller and the SATA/SAS controller will remain unchanged. This differs from the Windows Vista implementation of PCIe Hot-Plug which must disable, re-enumerate, and reallocate resources to all devices sharing the same root port before the newly connected device becomes operational. Devices may simply be connected to or disconnected from the system in an arbitrary fashion or in a more graceful fashion by writing to files located under the */sysfs* directory. Writing 0 to the */sysfs/bus/pci/slots/0000:bb:dd.f/power* file (where bb specifies the bus number, dd specifies the device number, and f specifies the function number of a downstream port) will disable the specified downstream port whilst writing a 1 to the same file will enable the downstream port. When a downstream port is disabled, all devices connected to that port are removed from the Linux kernel and the PCI drivers remove() function will be called to perform the required cleanup. By the same token, when a downstream port is enabled, any devices connected to it will be discovered,

Notes

added to the Linux kernel and the PCI drivers probe() function will be called to perform the device initialization. Downstream ports may also be enabled or disabled by third-party kernel mode drivers by utilizing the Hot-Swap drivers API described in the section titled Enabling and Disabling Downstream Ports.

Figure 2 below depicts examples of the types of devices that may be connected to or disconnected from a system running the IDT PCIe Hot-Swap driver.

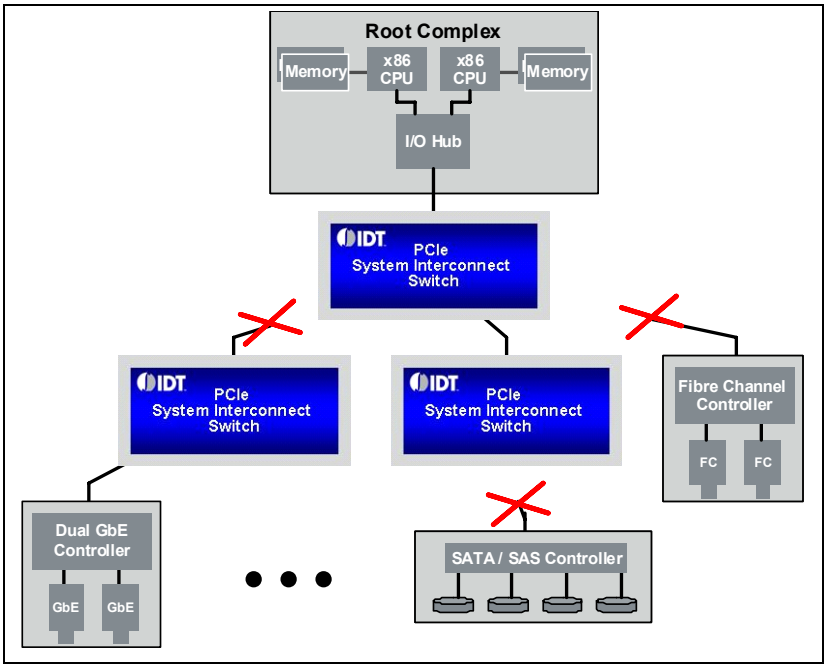

Figure 2 Device Removal

The key to achieving this type of Hot-Swap functionality is the ability to assign additional unused bus numbers and memory resources to each downstream port in the system. When a new device is connected to the system, these additional resources are utilized in order to bring the device to an operational state without having to compromise the operational state of the other devices sharing the same root port.

System Interconnect Considerations

The IDT System Interconnect Topology[1] shown in Figure 3 consists of a single x86 based Root Complex Processor (RP) connected to one or more intelligent x86 based Endpoint Processors (EP) via an IDT System Interconnect PCIe switch. An IDT Inter-Domain switch is utilized by each EP to connected them to the downstream ports of the System Interconnect PCIe switch. This topology in conjunction with the IDT System Interconnect Software[2] allows high speed data transfer between any two peers in the system.

Notes

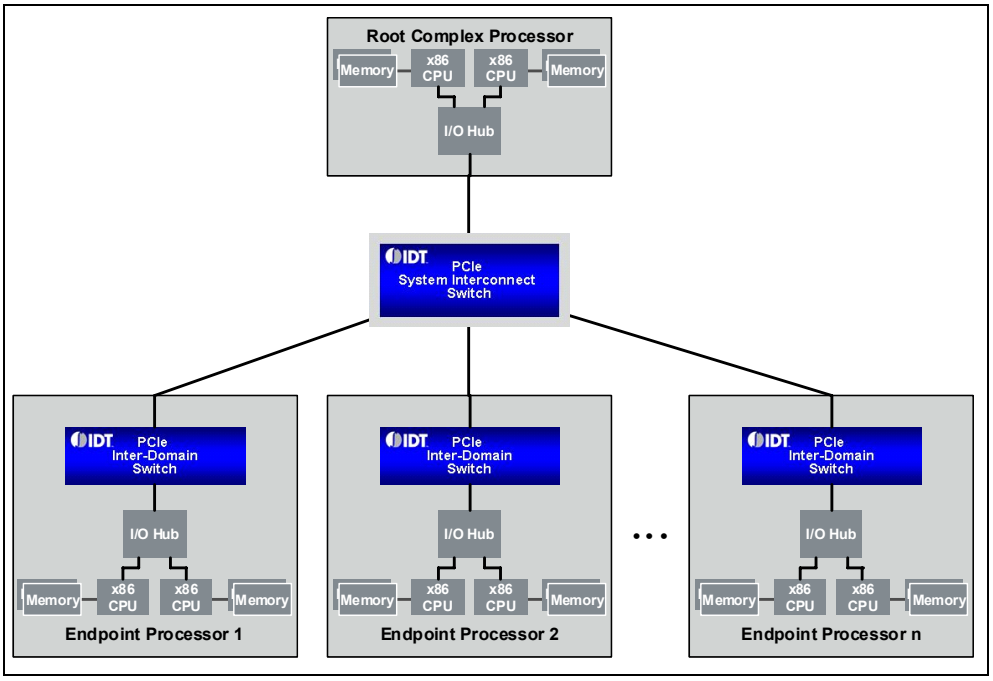

Figure 3 IDT System Interconnect Topology

When a new EP is connected to or disconnected from a system utilizing this type of topology, the Hot-Swap driver treats the EP the same as any normal endpoint device. There is, however, a difference in the way in which an RP is treated when it is connected to or disconnected from the system.

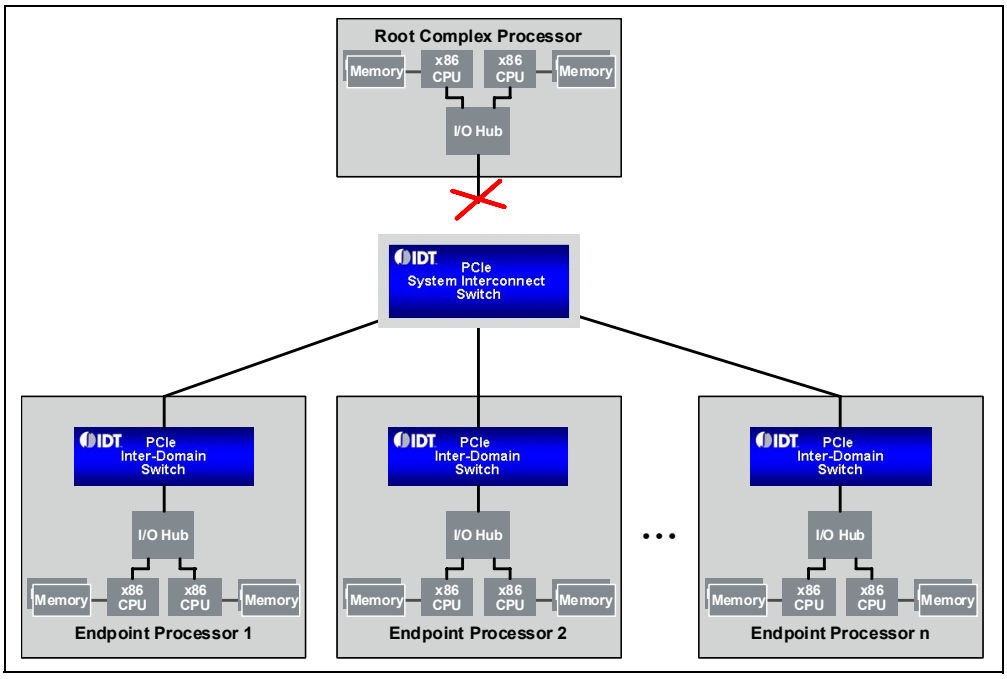

Figure 4 System Interconnect Root Complex Processor Removal

Typically, when the upstream port of a PCIe switch is disconnected as shown in Figure 4, the switch will issue a hot reset to all of its downstream ports. This functionality is undesirable in the System Interconnect topology since it would prevent the EPs from communicating with each other when the RP is disconnected from the system. The Hot-Swap driver overcomes this issue by detecting IDT PCIe System Interconnect

Notes

Switches and disabling their ability to generate a hot reset when their upstream ports are disconnected.This functionality may be disabled in the Hot-Swap driver by setting the *linkDownHotReset* module parameter to a non zero value.

The Hot-Swap driver also sports a passive resource allocation algorithm. Used primarily with the IDT System Interconnect Topology, this algorithm allows an RP to be connected to an operational system with minimal system interruption. Typically, when an endpoint device is connected to the Root Complex, it is the Root Complex that assigns memory resources to the device. In the case of passive resource allocation, it is the Root Complex that is being connected to the system and the endpoint devices that inform the Root Complex which memory resources should be used. This functionality is enabled by default in the Hot-Swap driver, but it may be disabled by setting the *passiveAllocationDisable* module parameter.

Supported Architectures

The IDT PCIe Hot-Swap driver module is supported on and has been tested with the following environment.

- Fedora Core 6 (kernel version 2.6.18)
- GCC 4.1.1
- Intel Bensley & Stoakley based PC's
- AMD K8N based PC's

Driver Operation

It may not be desirable to have all PCIe root ports on a system hot-swapable. For this reason, when the driver module is loaded, the *slots* module parameter may be used to specify which PCIe root ports should be placed under the control of the Hot-Swap driver.

Bus Enumeration

The number of buses assigned to each root port may be specified by the *minRootBuses* module parameter (set to 32 by default) when the driver is loaded. Should a switch be connected to the system at runtime, the number of buses assigned to the parent port are shared evenly between the downstream ports of the newly connected switch.

Device Detection

One of the most important aspects of any Hot-Swap driver is to be able to successfully detect when a device has been connected to or disconnected from the system. Since there is no defined methodology to perform such a task, the IDT Hot-Swap driver employs a three tier approach in order to successfully detect PCIe devices.

First, the link capabilities register on all downstream ports is checked to see if it supports Data Link Layer Link Active (DLLLA) status reporting. If DLLLA is supported, the DLLLA status is used to determine when a device is connected or disconnected.

Second, some manufactures have a vendor-specific status register that may be used to determine if a device is connected. Although this results in software changes to the Hot-Swap driver, the driver already contains the hooks necessary to implement custom link status checking functions on a per vendor/device ID basis.

Third, an attempt is made to perform configuration reads to obtain the device/vendor ID of the device connected to the downstream ports. If a device is connected, a valid device/vendor ID will be returned. If no device is connected, 0xffff will be returned for the vendor/device ID.

Notes

Resource Allocation

As with the bus enumeration, a module parameter *rootIOMemSize* is used to specify the amount of memory space that should be allocated to each root port under the control of the Hot-Swap driver (defaults to 0x2000000 bytes). Should a new switch be connected to the system at runtime, the memory resources allocated to the parent port will be shared evenly between the downstream ports of the newly connected switch.

Device Driver Application Programming Interface

The following sections describe the API exported by the Hot-swap driver for use by third-party software.

Custom Link Status Checking

The Hot-swap driver exports the following function that allows a third-party kernel mode driver to assign a custom link status checking function to the specified downstream port.

```
void *registerLinkStatusHandler(struct pci_dev *dev, int
(*handler)(struct pci_dev *dev))
```

Register a custom link status checking function. The *handler* parameter should be a pointer to a function that accepts a single parameter that points to the PCI device structure for the downstream port to be checked. The return value from the link status checking function should be 1 if a device is detected and 0 if a device is not detected.

Parameters:

dev Pointer to the downstream ports PCI device structure.

handler Pointer to the new link status checking function.

Returns:

A pointer to the previous link status handler if successful, otherwise NULL.

Pausing Link Status Checking

The Hot-swap driver exports the following function that allows a third-party kernel mode driver to pause the automatic link status check.

```
int pauseResumeLinkCheck(int pause)
```

Pause or Resume the Hot-swap driver automatic link status checking.

Parameters:

pause 1 to pause link checking, 0 to resume.

Return Values:

0 Link checking was previously running.

1 Link checking was previously paused.

Preventing Link Status Checking

The Hot-swap driver exports the following functions that allow a third-party kernel mode driver to exclude or include ports from automatic link status checking at runtime.

```
int pcieHotSwapExclude(struct pci_dev *dev)
```

Exclude the specified downstream port from automatic link status checking.

Parameters:

dev Pointer to the PCI device structure for the downstream port.

Notes

Return Values:

0 Success.

-ENODEV Device not found.

```
int pcieHotSwapInclude(struct pci_dev *dev)
```

Include the specified downstream port in automatic link status checking. The downstream port should first have been excluded from automatic link status checking by a call to `pcieHotSwapExclude()`.

Parameters:

dev Pointer to the PCI device structure of the downstream port.

Return Values:

0 Success

-ENODEV Device not found.

-EINVAL Invalid device structure pointer.

Enabling and Disabling Downstream Ports

The Hot-swap driver exports the following functions that allow a third-party kernel mode driver to enable or disable a downstream port at runtime.

```
int pcieHotSwapDisable(struct pci_dev *dev)
```

Disable the specified downstream port. All devices connected to the port will be removed, all resources released and the PCI `remove()` function called for any drivers associated with the devices being removed.

Parameters:

dev Pointer to the PCI device structure of the downstream port.

Return Values:

0 Success.

-ENODEV Device not found.

-EINVAL Invalid PCI device structure.

```
int pcieHotSwapEnable(struct pci_dev *dev)
```

Enable the specified downstream port. Any devices or switches connected to the port being enabled are configured, resources allocated and the PCI `probe()` function is called for any drivers associated with the devices being added.

Parameters:

dev Pointer to the PCI device structure of the downstream port.

Return Values:

0 Success.

-ENODEV Device not found.

-EINVAL Invalid PCI device structure.

Building the Hot-Swap Driver

The IDT PCIe Hot-Swap driver module requires access to a number of symbols that are private to the Linux kernel. In order for the driver to work correctly, the Linux kernel must be patched and rebuilt to allow access to the required symbols. Before building the kernel sources, it will be necessary to obtain a copy of the kernel source RPM (kernel-2.6.18.src.rpm) and the unifdef RPM (unifdef-1.171-5.fc6.i386.rpm). The following procedure should be used to install, configure, patch, and build the kernel sources while logged in as *root*.

Notes

- Installing the RPMs

  ```
  rpm -ivh kernel-2.6.18.src.rpm
  rpm -ivh unifdef-1.171-5.fc6.i386.rpm
  ```

- Configuring the Sources

  ```
  cd /usr/src/redhat/SPECS
  ```

  `rpmbuild -bp --target=<arch> kernel-2.6.spec` (Where <arch> is the target architecture i.e. i686)

- Create a Symbolic Link to the Source

  ```
  ln -s /usr/src/redhat/BUILD/kernel-2.6.18/linux-2.6.18 /usr/src/linux
  ```

- Patch the Kernel

  ```
  cd /usr/src/linux
  ```

  `patch -p0 < IDT_hotswap_2.6.18.patch` (The patch file is located in the hotswap source directory)

- Configure the Kernel

  ```
  make oldconfig
  ```

- Build the Kernel

  ```
  make
  ```

- Install the new Kernel Modules

  ```
  make modules_install
  ```

- Install the new Kernel

  ```
  make install
  ```

Edit */boot/grub/grub.conf* and make sure that the default boot kernel is set to the newly built kernel before rebooting the machine.

After the machine has been rebooted with the newly created kernel, the Hot-Swap driver may be built. Login as *root* and change to the directory that contains the Hot-Swap driver source code. Enter the following command to build and install the Hot-Swap driver module.

```
make install
```

Loading the Hot-Swap Driver

Once installed, the Hot-Swap driver may be loaded by using the *modprobe* utility as shown below.

```
modprobe idt-hotswap [module parameters]
```

In order to automatically load the Hot-Swap driver when the machine boots, add the above command to the file called */etc/rc.modules*. If */etc/rc.modules* does not exist, create it with executable permissions. The Hot-Swap driver may be unloaded by invoking the *modprobe* utility with the *-r* command line option as shown below.

```
modprobe -r idt-hotswap
```

Module Parameters

The following table lists the Hot-Swap driver module parameters, their default values, and their usage.

Notes

Parameter Name	Default Value	Description
slots	NULL	Used to specify which root ports should be under the control of the Hot-Swap driver. The format of this parameter is “bb:dd.f” where bb is the bus number, dd is the device number and f is the function number of the root port to be controlled by the Hot-Swap driver. Multiple root ports may be specified if separated by a comma.
exclude	NULL	Used to specify the downstream ports, in “bb:dd.f” format, that should be excluded from automatic link status checking when a link status checking function external to the Hot-swap driver is employed. Multiple downstream ports may be specified if separated by a comma.
minRootBuses	32	The number of bus numbers that should be assigned to each root port specified by the *slots* parameter.
rootIOMemSize	0x2000000	The amount of IO memory space that should be assigned to each root port specified by the *slots* parameter.
linkDownHotReset	0	When set to 1, any IDT System Interconnect switch will generate a hot reset to its downstream ports when its upstream port is disconnected.
passiveAllocationDisable	0	Set to 1 to disable passive resource allocation when not using the Hot-Swap driver with the IDT System Interconnect topology.
firstBus	0	Used to specify the first bus number that should be used when re-enumerating the PCIe buses. If set to zero, the driver will use the next available bus number.
debug	0	Set to 1 to enable diagnostic messages.

Table 1 Hot-Swap Driver Module Parameters

References

[1] Enabling Multi-peer Support with a Standard-Based PCI Express multi-port Switch White Paper, Kwok Kong, IDT.

[2] Application Note 571, PCI Express® System Interconnect Software Architecture for x86-based Systems, Kwok Kong & Alex Chang, IDT.

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01 Jan 2024)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit www.renesas.com/contact-us/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.