Software API for Switch Partitioning

Application Note AN-709

Notes

By Alex Chang

Introduction

The IDT PCIe Gen2 16-port 64-lane switch (PES64H16G2) supports multiple (up to 16) active switch partitions with IDs 0 to 15. A switch partition represents an independent PCIe hierarchy whose operation is independent of other switch partitions. The state of a given switch partition may be set up at any time via the switch partition control register, which means a switch partition may be enabled or disabled dynamically.

Each switch partition serves as a logical container to which switch ports are attached. When a given switch partition is disabled, all the associated ports are also disabled, regardless of the current mode of the ports. The PES64H16G2 supports up to 16 ports and each port may be dynamically reconfigured to operate in one of the following modes:

- Disabled
- Unattached
- Downstream switch port
- Upstream switch port.

Figure 1 illustrates the logical view of two separate switch partitions configured within the PES64H16G2. The remaining ports not shown in Figure 1 are configured in non-operational modes (i.e., Disabled or Unattached) and they are not associated with any switch partition.

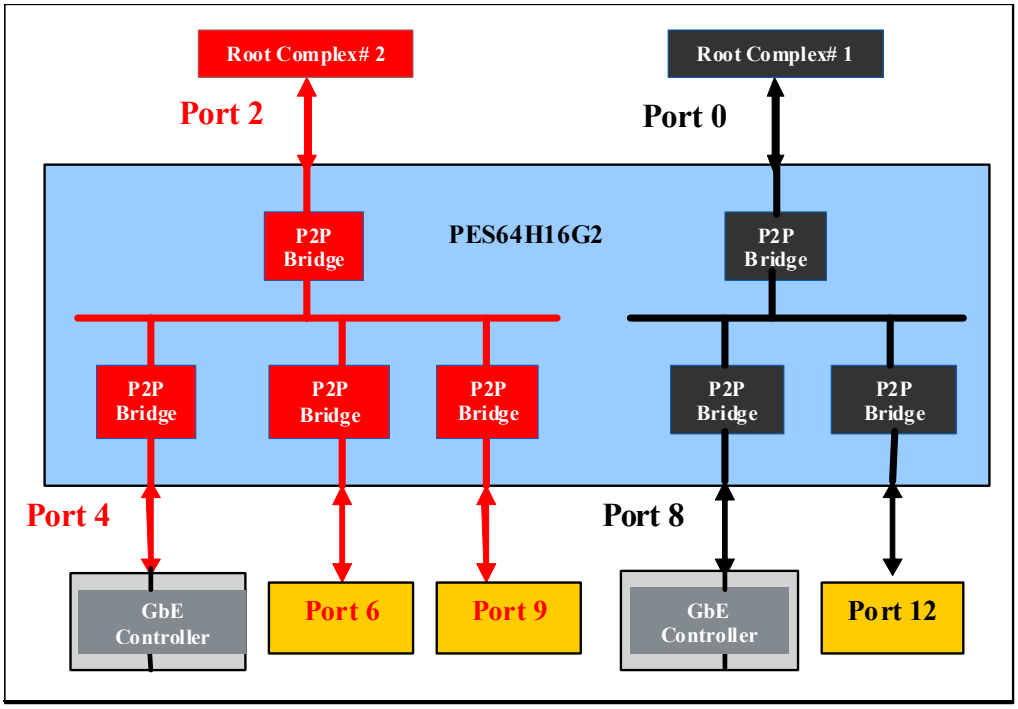

Figure 1 Logical View of Two Switch Partitions

This application note describes the basics of switch partitioning and the software API that can be used to achieve dynamic switch partitioning.

Notes

Switch Partitioning

Before configuring switch partition within the PES64H16G2, please note that any configurations other than the valid switch partitioning listed below may generate undefined behavior.

- A switch partition with no associated ports
- A switch partition with one upstream port and zero or more downstream ports
- A switch partition with no upstream port and one or more downstream ports.

In order to support dynamic switch partitioning, the switch mode needs to be configured in multi-partition mode with EEPROM initialization after fundamental reset. The PES64H16G2 provides a set of partition and port-related registers. Each partition or port has it own control and status registers that are accessible via the SMBus or PCI Bus.

Static Switch Partitioning

Without violating the valid switch partitioning principle mentioned above, multiple switch partitions may be set up via EEPROM, which is called Static Switch Partitioning. For example, the switch partitioning shown in Figure 1 enables two separate partitions after the systems power up. In this case, the partitions are associated with different Root Complexes, and power cycling either Root Complex does not affect the operation of the switch partition residing in the other Root Complex.

Dynamic Switch Partitioning

The PES64H16G2 also supports dynamic switch partitioning at any desired timing. When powering up, the switch mode is required to be in multiple partition mode. After a fundamental reset, all the partitions are in the disable state and all the ports are in unattached mode. The following switch partitioning can be done through programming the related partition and port control registers dynamically:

- Activate or disable a given partition
- Enable a given port as an upstream or a downstream port
- Disable a given port
- Unattach a given port
- Attach a given port to a given partition as a downstream port
- Attach a given port to a given partition as an upstream port if the partition doesn't have any upstream port attached.

After successfully programming the desired control register(s), the current status of a given partition or port is reflected in its own status register. IDT not only provides switch partitioning API to achieve the above tasks, but also provides a Hot Swap driver to help the system on re-enumerating PCI buses and re-allocating required resources after dynamic switch partitioning without power cycling.

Taking the switch partitioning shown in Figure 1 as example, assume an application wants to move Port 8 of "black" partition to "red" partition and ensure the Gigabit Ethernet Controller plugged into the port can still function properly after the move. Before triggering the switch partitioning, the following preparations are required:

- Re-compile Linux kernel after applying IDT Hot Swap patches
- Boot both Root Complexes with the newly re-built kernel
- Build IDT Hot Swap driver
- Load IDT Hot Swap driver on both Root Complexes
- Re-build libPCI and copy libpci.a to /usr/lib directory
- Build switch partitioning API.

The switch partitioning API includes a command-line executable called idtsp, which is a sample application used to trigger any desired switch partitioning task. In this case, idtsp can be executed from either Root Complex to move Port 8 from "black" partition to "red" partition. After the IDT Hot Swap driver detects the change, it engages a new round of bus re-enumeration and resource allocation on both Root Complexes to reflect the new topology. Now, the Gigabit Ethernet Controller is back to a functional state after its associated software has been re-loaded.

Notes

For more details regarding IDT's Hot Swap driver or other switch partitioning issues, please contact IDT at ssdhelp@idt.com.

Software API

A Linux-based software API package is implemented to make it easier to dynamically configure switch partitions within the PES64H16G2. The API package is developed in C language and can be easily ported to other operating systems. It supports Fedora Core 6, 7, 8, and 9 releases with both 32 and 64-bit kernels. The software API package is divided into three different layers as shown in Figure 2.

The top layer is the User Application that runs in the user space or user device driver that runs in the kernel space. This layer is provided by the customers per their specific applications. The middle layer is the Switch Partition APIs that is provided by IDT. It provides APIs to the user level program for switch partitioning. The bottom layer is I/O Access APIs that is system and OS dependent. This layer accesses PCIe configuration space registers via LibPCI library to finalize switch partitioning requests from User Application layer.

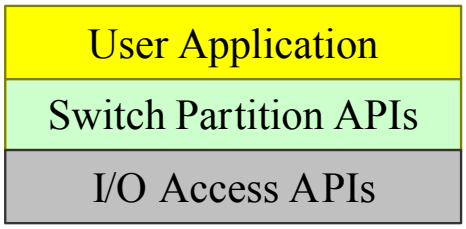

Figure 2 Switch Partition API Architecture

User Application

IDT provides a sample application called idtsp included in the Switch Partition API package to demonstrate how switch partitioning works. The use of idtsp is detailed in Table 1.

The syntax of idtsp is: **idtsp bus# device# function# -aspcrudxn value#1 value#2**

Options	Descriptions
none	List usages of idtsp.
bus# dev# fun#	List the current partition/port information of the port associated with the specified bus#, dev# and fun#.
-a	List all current switch partitions with their associated ports.
-s	Specify a target partition.
-p	Specify a target port.
-c	Enable a partition or enable a port and attach it to a partition.
-r	Disable a partition or remove a port from a partition.
-d	Configured as a downstream port.
-u	Configured as an upstream port.
-n	Set device number for a specified port with option 'p'.
-x	Disable OMA for port(s).

Table 1 Use of idtsp (Page 1 of 2)

Notes

Examples	Descriptions
idtsp 4 0 0 –s 2	Display current status and information of partition# 2.
idtsp 4 0 0 –p 1	Display current status and information of port# 1.
idtsp 4 0 0 –pn 5 0	Set device number of port# 5 to be 0.
idtsp 4 0 0 –px 4 0	Disable OMA of port# 4.
idtsp 4 0 0 –sc 4	Enable partition# 4.
idtsp 5 0 0 –sr 3	Disable partition# 3.
idtsp 3 0 0 –spcd 0 5	Add port# 5 as a downstream port of partition# 0.
idtsp 4 0 0 –spr 2 4	Remove port# 4 from partition# 2.
idtsp 5 0 0 –spcu 3 8	Add port# 8 as upstream port of partition# 3.

Table 1 Use of idtsp (Page 2 of 2)

Switch Partition APIs

Switch Partition APIs are provided to satisfy requests from the User Application layer. Switch Partition and I/O Access API share the following return codes, detailed in Table 2, when return type is an unsigned integer:

Value	Code	Meaning
-1	RC_NO_DEVICE	No device found associated with the BDF.
0	RC_SUCCESS	Function completes successfully.
1	RC_INV_PARA	Invalid parameter(s) found.
2	RC_NOT_SUPPORT	Request can not be fulfilled.
3	RC_REG_FAILURE	Fails on accessing register(s).
0xFF	RC_UNKNOWN_ERR	Unknown error.

Table 2 API Return Codes

The major APIs are listed below:

Notes

unsigned idtspFormBDF(unsigned bus, unsigned dev, unsigned fun)

Description	Prepare an unsigned value containing Bus, Device and Function numbers.
Input(s)	
bus	Bus number of a PCI-to-PCI bridge.
dev	Device number of a PCI-to-PCI bridge.
fun	Function number of a PCI-to-PCI bridge.
Output(s)	
none	
Return	
unsigned	An unsigned value composed of Bus# (bit[15..8]), Device#(bit[7..3]) and Function#(bit[2..0]).

unsigned idtspSetPartState(unsigned bdf, unsigned part_num, unsigned state)

Description	Set the given partition as specified state.
Input(s)	
bdf	The Bus, Device and Function numbers of any enabled PCI-to-PCI bridge in the PES64H16G2. API idtspFormBDF may be used to generate the value.
part_num	Partition number [15..0].
state	Set current state as disabled (0), active (1), hot reset (2) or reset (3).
Output(s)	
none	
Return	
unsigned	Refer to Return Codes in Table 2.

Notes

unsigned idtspGetPartState(unsigned bdf, unsigned part_num, unsigned* state)

Description	Retrieve current state of the given partition.
Input(s)	
bdf	The Bus, Device, and Function numbers of any enabled PCI-to-PCI bridge in the PES64H16G2. API idtspFormBDF may be used to generate the value.
part_num	Partition number [15..0].
state	Pointer to an unsigned value.
Output(s)	
*state	Current state as disabled (0), active (1), hot reset (2) or reset (3).
Return	
unsigned	Refer to Return Codes in Table 2.

unsigned idtspSetDeviceNum(unsigned bdf, unsigned port_num, unsigned dev_num)

Description	Set the specified device number for the given port.
Input(s)	
bdf	The Bus, Device, and Function numbers of any enabled PCI-to-PCI bridge in the PES64H16G2. API idtspFormBDF may be used to generate the value.
port_num	Port number [15..0].
dev_num	Set device number [15..0] of the specified port.
Output(s)	
none	
Return	
unsigned	Refer to Return Codes in Table 2.

Notes

unsigned idtspGetDeviceNum(unsigned bdf, unsigned port_num, unsigned* dev_num)

Description	Retrieve device number of the given port.
Input(s)	
bdf	The Bus, Device, and Function numbers of any enabled PCI-to-PCI bridge in the PES64H16G2. API idtspFormBDF may be used to generate the value.
port_num	Partition number [15..0].
dev_num	Pointer to an unsigned value.
Output(s)	
*dev_num	Current device number of the specified port.
Return	
unsigned	Refer to Return Codes in Table 2.

unsigned idtspAddPort(unsigned bdf, unsigned part_num, unsigned port_num, unsigned upstream)

Description	Attach the specified port to the given partition, variable upstream used to indicate it's an upstream or downstream port.
Input(s)	
bdf	The Bus, Device, and Function numbers of any enabled PCI-to-PCI bridge in the PES64H16G2. API idtspFormBDF may be used to generate the value.
part_num	Partition number [15..0].
port_num	Port numer [15..0].
upstream	Non-zero means adding as upstream port, zero means downstream port.
Output(s)	
none	
Return	
unsigned	Refer to Return Codes in Table 2.

Notes

unsigned idtspRemovePort(unsigned bdf, unsigned part_num, unsigned port_num, unsigned upstream)

Description	Remove the specified port from the given partition and set the port mode as disabled.
Input(s)	
bdf	The Bus, Device, and Function numbers of any enabled PCI-to-PCI bridge in the PES64H16G2. API idtspFormBDF may be used to generate the value.
part_num	Partition number [15..0].
port_num	Port numer [15..0].
Output(s)	
none	
Return	
unsigned	Refer to Return Codes in Table 2.

unsigned idtspFindPorts(unsigned bdf, unsigned part_num, unsigned* port, unsigned upstream)

Description	Find out the associated port(s) with the given partition, variable upstream used to indicate looking for downstream or upstream port(s).
Input(s)	
bdf	The Bus, Device, and Function numbers of any enabled PCI-to-PCI bridge in the PES64H16G2. API idtspFormBDF may be used to generate the value.
part_num	Partition number [15..0].
port	Pointer to an unsigned value.
upstream	Non-zero means finding upstream port, zero means downstream port(s).
Output(s)	
*port	Bit value as 1 indicates the corresponding port is associated with the partition.
Return	
unsigned	Refer to Return Codes in Table 2.

Notes

unsigned idtspSetPortMode(unsigned bdf, unsigned port_num, unsigned mode)

Description	Set the specified mode for the given port.
Input(s)	
bdf	The Bus, Device, and Function numbers of any enabled PCI-to-PCI bridge in the PES64H16G2. API idtspFormBDF may be used to generate the value.
port_num	Port number [15..0].
mode	Configure the port as disabled (0), a downstream port (1), an upstream port (2) or unattached (5).
Output(s)	
none	
Return	
unsigned	Refer to Return Codes in Table 2.

unsigned idtspGetPortMode(unsigned bdf, unsigned port_num, unsigned* mode)

Description	Retrieve current mode of the given port.
Input(s)	
bdf	The Bus, Device, and Function numbers of any enabled PCI-to-PCI bridge in the PES64H16G2. API idtspFormBDF may be used to generate the value.
port_num	Port number [15..0].
mode	Pointer to an unsigned value.
Output(s)	
*mode	Current mode.
Return	
unsigned	Refer to Return Codes in Table 2.

Notes

unsigned idtspSetOMA(unsigned bdf, unsigned port_num, unsigned oma)

Description	Set the specified operating mode change action for the given port.
Input(s)	
bdf	The Bus, Device, and Function numbers of any enabled PCI-to-PCI bridge in the PES64H16G2. API idtspFormBDF may be used to generate the value.
port_num	Port number [15..0].
oma	Set Operating Mode Change Action (OMA) as: No action - preserve state (0). Port fundamental reset (1). Port hot reset (2). Reserved (others).
Output(s)	
none	
Return	
unsigned	Refer to Return Codes in Table 2.

unsigned idtspGetOMA(unsigned bdf, unsigned port_num, unsigned* oma)

Description	Retrieve the current operating mode change action of the given port.
Input(s)	
bdf	The Bus, Device, and Function numbers of any enabled PCI-to-PCI bridge in the PES64H16G2. API idtspFormBDF may be used to generate the value.
port_num	Port number [15..0].
oma	Pointer to an unsigned value.
Output(s)	
*oma	Current OMA value.
Return	
unsigned	Refer to Return Codes in Table 2.

Notes

unsigned idtspGetPortLink(unsigned bdf, unsigned port_num, unsigned* link)

Description	Retrieve current link status of the given port.
Input(s)	
bdf	The Bus, Device, and Function numbers of any enabled PCI-to-PCI bridge in the PES64H16G2. API idtspFormBDF may be used to generate the value.
port_num	Port number [15..0].
link	Pointer to an unsigned value.
Output(s)	
*link	Current link status, 1 means up and 0 means down.
Return	
unsigned	Refer to Return Codes in Table 2.

unsigned idtspGetPortPartID(unsigned bdf, unsigned port_num, unsigned* part_id)

Description	Retrieve current partition ID the given port is associated with.
Input(s)	
bdf	The Bus, Device, and Function numbers of any enabled PCI-to-PCI bridge in the PES64H16G2. API idtspFormBDF may be used to generate the value.
port_num	Port number [15..0].
part_id	Pointer to an unsigned value.
Output(s)	
*part_id	Partition ID[15..0].
Return	
unsigned	Refer to Return Codes in Table 2.

Notes

unsigned idtspFindPortNum(unsigned bdf, unsigned* port_num)

Description	Retrieve the port number associated with the specified bus, device and function numbers.
Input(s)	
bdf	The Bus, Device, and Function numbers of any enabled PCI-to-PCI bridge in the PES64H16G2. API idtspFormBDF may be used to generate the value.
port_num	Pointer to an unsigned value.
Output(s)	
*port_num	Port number [15..0].
Return	
unsigned	Refer to Return Codes in Table 2.

IO Access APIs

I/O Access APIs are designed to access configuration space via LibPCI. They all share the same return codes in Table 2.

unsigned ioaPCIReadConfig(unsigned bdf, unsigned offset, unsigned* value)

Description	Retrieve the current value at specified offset in configuration space associated with the bus, device and function numbers.
Input(s)	
bdf	The Bus, Device, and Function numbers associated with a PCI-to-PCI bridge. API idtsp-FormBDF may be used to generate the value.
offset	Where to read from.
value	Pointer to an unsigned value.
Output(s)	
*value	Current read back value.
Return	
unsigned	Refer to Return Codes in Table 2.

Notes

unsigned ioaPCIWriteConfig(unsigned bdf, unsigned offset, unsigned value)

Description	Write the given value at the specified offset in configuration space associated with the bus, device and function numbers.
Input(s)	
bdf	The Bus, Device, and Function numbers associated with a PCI-to-PCI bridge. API idtsp-FormBDF may be used to generate the value.
offset	Where to write to.
value	Value to write.
Output(s)	
none	
Return	
unsigned	Refer to Return Codes in Table 2.

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.