Software API for Upstream Port Failover in Inter-domain PCIe Switches

Software API

Notes

By Kwok Kong, Alex Chang

Introduction

The IDT Inter-domain PCIe switches (PES24NT3 and PES12NT3) supports an NTB upstream port failover mechanism that enables the construction of fault tolerant systems. In the document, the PES24NT3 is used as examples but the all the failover descriptions apply to the PES12NT3 as well.

The NTB upstream port failover usage model is illustrated in Figure 1. In this usage there is a primary root and a secondary root. Both roots are active and may communicate using transactions flowing through the NTB, mechanisms provided by the NTB for interprocessor communications, or an out-of-band communications channel. In normal mode, the primary root is responsible for configuring and managing the internal PCIe hierarchy (i.e., the PCIe hierarchy consisting of upstream port A, downstream port B, P2P bridges, and the internal NTB endpoint).

NTB upstream port failover enables the swapping of the upstream port (i.e., port A) with the NTB port (i.e., port C). When a hardware or software failure is detected in the primary root, the PES24NT3 may be directed to operate in a failover mode. In failure mode, the secondary root (i.e., the root associated with port C) becomes the root responsible for configuring and managing the internal PCIe hierarchy and the primary root becomes the root of the external NTB hierarchy.

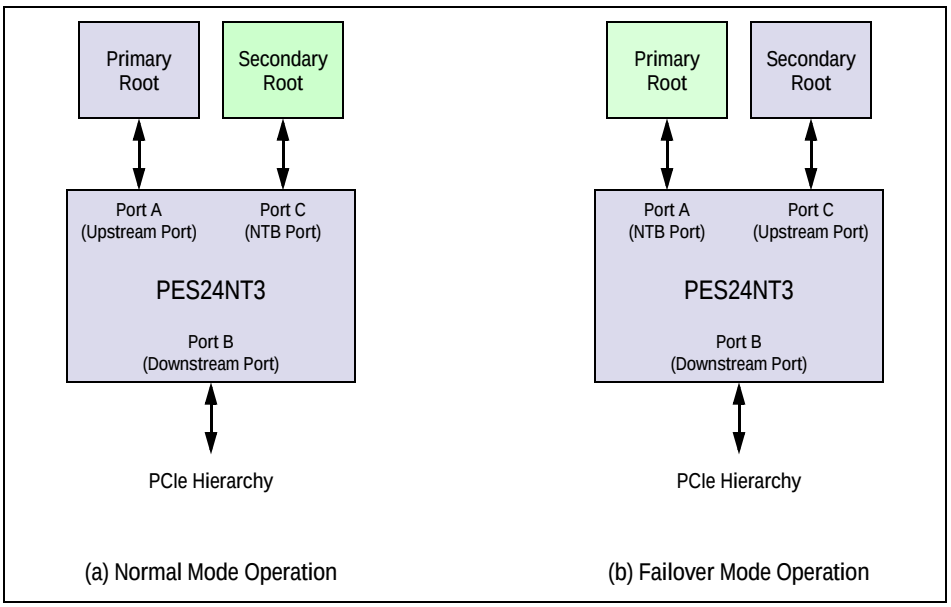

Figure 1 NTB Upstream Port Failover Usage Model

The PES24NT3 NTB upstream port failover architecture is shown in Figure 2. The two main components of this device are switch logic and a SerDes switch. The PES24NT3 switch logic implements a three port non-transparent switch that does not support NTB upstream port failover. It consists of PCIe stacks, a switch core and NTB logic. The SerDes switch enables SerDes lanes associated with port A and C to be passed through unmodified or swapped.

© 2019 Renesas Electronics Corporation

Notes

In normal mode, the SerDes switch operates in a pass-through configuration. This connects the external SerDes lanes associated with port A with the internal port A upstream port of the switch logic and the SerDes lanes associated with port C with the internal port C NTB port. In failover mode, the SerDes switch operates in a swapped configuration. This connects the SerDes lanes associated with port A to the internal port C NTB port and the external SerDes lanes associated with port C to the internal port A upstream port.

In failover mode the device associated with port C SerDes assumes all of the resources and responsibilities of the internal port A upstream port and visa versa. This means that the root associated with port C has direct access to all port A upstream port registers, receives internal PCIe hierarchy messages (e.g., error and INTx), and so on.

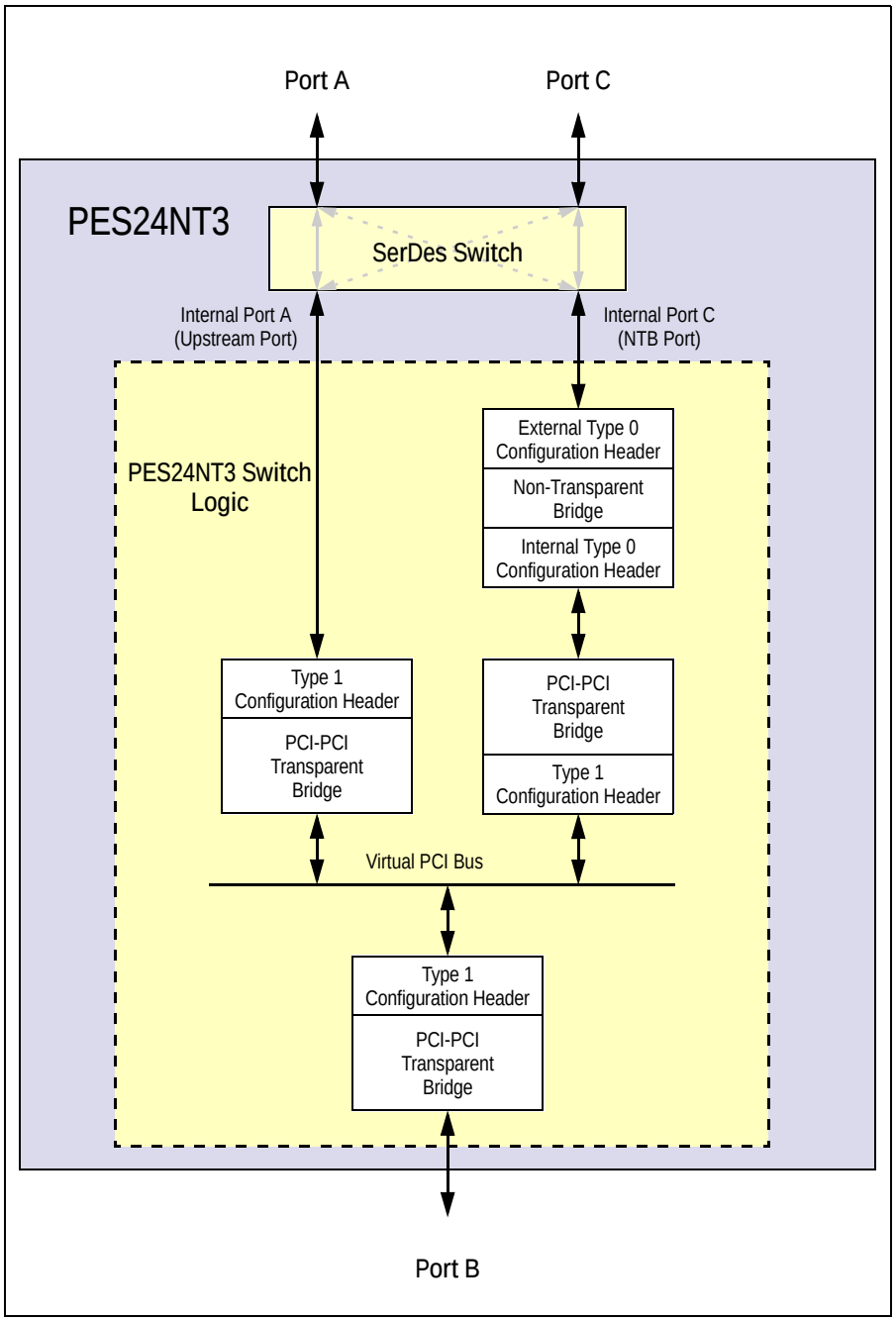

Figure 2 PES24NT3 NTB Upstream Port Failover Architecture

© 2019 Renesas Electronics Corporation

Notes

This document describes the basic configuration of upstream port failover and the software API to configure it.

Failover

Failover may be initiated statically or dynamically. A static upstream port failover requires a fundamental reset to be initiated whenever a failover mode change is required. An static upstream port failover consists of the following steps:

- Assert the PCIe fundamental reset signal
- Modify the switch mode signals to the selected failover mode (i.e., normal mode or failover mode).
- Negate the PCIe fundamental reset signal

Since the static upstream port failover requires external signals to be modified and is system dependent, software API does not support the static upstream port failover.

Dynamic upstream port failover allows a failover to occur while the system is live and in a manner that preserves the system state. There are three methods to initiates a dynamic upstream port failover:

- Software initiated failover - A failover may be initiated by modifying the failover control register.
- Signal initiated failover - An upstream port failover may be initiated by a change in the state of the NTB Upstream Port Failover signal.
- Watchdog timer initiated failover - An NTB upstream port failover may be initiated as the result of an expiration of a watchdog timer.

Software API supports all three methods of dynamic upstream port failover.

Failover Examples

The PCI topology changes after the NTB upstream port failover. The system software has to update its PCI topology after the NTB upstream port failover. A total of four failover examples are described in this section to show the failover procedure.

- Managed Failover - There is no failure in the system. For software maintenance or management purpose, user initiates a failover via software.
- Recovery Failover - The active root complex has crashed or removed from the upstream port of the PES24NT3 device. The Standby root complex forces a failover such that it becomes the active root complex.
- Watchdog Timer Failover - The watchdog timer has expired and initiates a failover automatically.

It is assumed that the primary root is the active root complex and the PES24NT3 device is used in all the examples.

Managed Failover

In this example, there is no failure in the system. Both the primary and secondary roots are active. The primary root is the active root complex. A failover is initiated by management for the purpose of software maintenance or management.

Primary Root Change

Before the NTB upstream port failover, the primary root is the root complex of the PES24NT3 device. The primary root sees three PCI-PCI bridges and the Internal Endpoint inside the PES24NT3 device. There may also be a PCI tree below Port B of the PES24NT3 device. The PCI tree may consists of zero, one or multiple PCIe switches and zero, one of multiple endpoints. The primary root configures the Internal Endpoint registers to initiate the NTB upstream port failover.

After the NTB upstream port failover, the primary root is no longer the root complex of the PES24NT3 device. The primary root sees a single External Endpoint inside the PES24NT3. The multiple PCI-PCI bridges, Internal Endpoint and the PCI tree below Port B of the PES24NT4 device disappear. The primary root has no access to the Internal Endpoint anymore before recognizing the External Endpoint first. The topology as viewed by the Primary root before and after the failover is shown in Figure 3.

© 2019 Renesas Electronics Corporation

Notes

The procedure to initiate the NTB upstream port failover are:

1. Stop communication to External Endpoint
2. Initiate a “virtual” unplug of all the devices that are connected to Port B of the PES24NT3 device. This uninitializes all the device drivers that are associated with the devices that are below Port B of the PES24NT3 device.
3. Initiate the NTB Upstream port failover. At this point, software has to assume that it has no access to the Internal Endpoint anymore.
4. Initiate a “virtual” unplug of the PES24NT3 device
5. Look for a new device “External Endpoint of the PES24NT3”
6. When the External Endpoint is discovered, run the External Endpoint device driver to initialize the External Endpoint.
7. Re-establish communication with the Internal Endpoint
8. NTB Upstream Port Failover is completed.

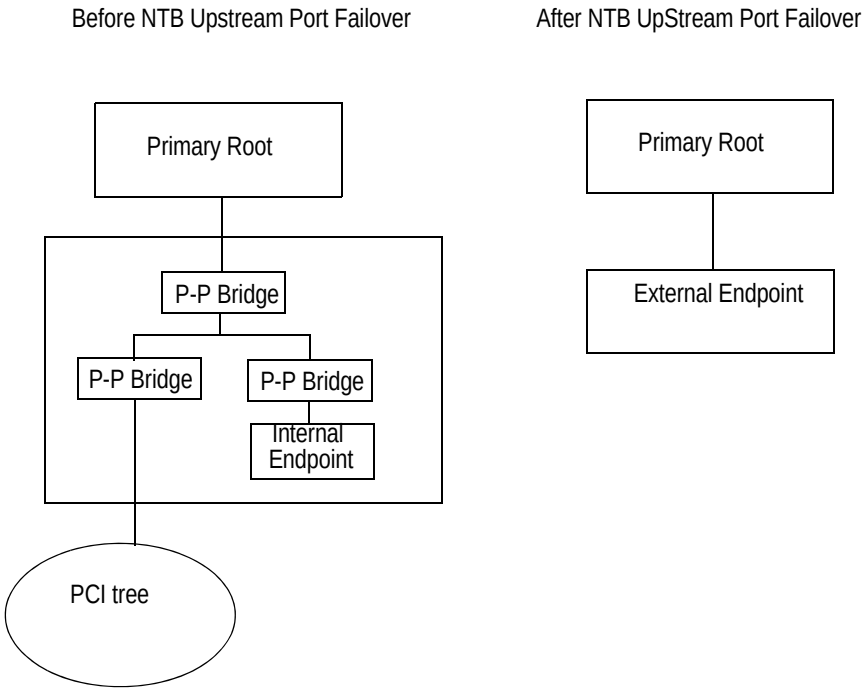

Figure 3 PCI tree for Primary root before and after NTB Upstream Port Failover

Secondary Root Change

Before the NTB upstream port failover, the secondary root is not the root complex of the PES24NT3 device. The secondary root sees a single External Endpoint inside the PES24NT3 device.

After the NTB upstream port failover, the secondary root is the root complex of the PES24NT3 device. The secondary root now sees three PCI-PCI bridges and the Internal Endpoint inside the PES24NT3 device. There may also be a PCI tree below Port B of the PES24NT3 device. The PCI tree may consists of zero, one or multiple PCIe switches and zero, one of multiple endpoints. The External Endpoints disap-

© 2019 Renesas Electronics Corporation

Notes

pears from the secondary root. The topology as viewed by the Secondary root before and after the failover is shown in Figure 4. In this example, the primary root is responsible for initiating the NTB Upstream Port Failover. The secondary root never initiates the NTB Upstream Port Failover.

The procedure to initiate the NTB upstream port failover are:

1. Get notification from the primary root that the NTB upstream port failover is going to happen
2. Initiate a “virtual” unplug of External Endpoint
3. Look for a new PCI-PCI bridge device of the PES24NT3 device
4. When the PCI-PCI bridge device is discovered, run PCI enumeration procedure to discover and configure the three PCI-PCI bridges, Internal Endpoint and the devices in the PCI tree that is connected to port B of the PES24NT3 device.
5. Re-establish communication with the Internal Endpoint

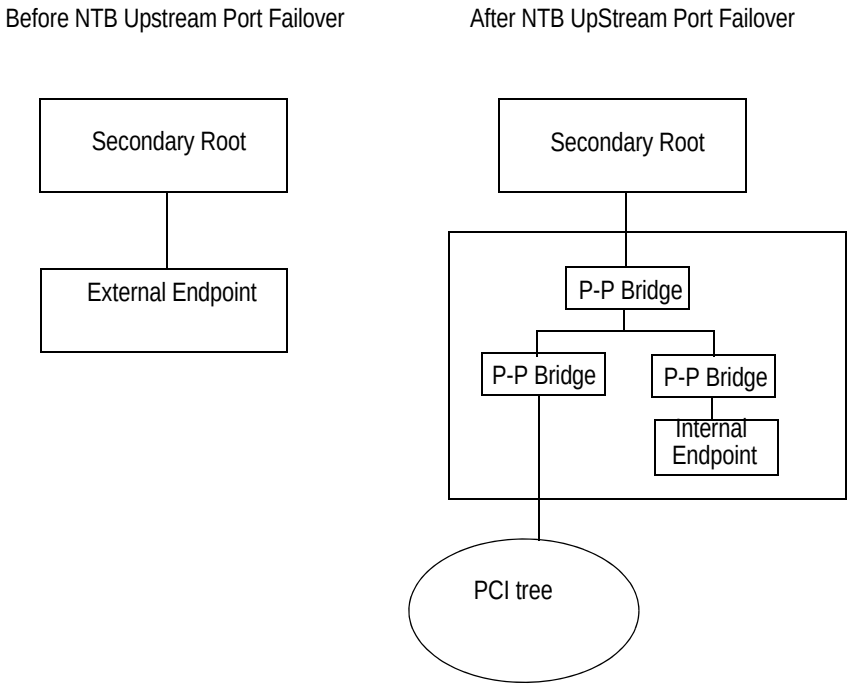

Figure 4 PCI tree for Secondary Root before and after NTB Upstream Port Failover

Recovery Failover

The primary root has failed. The secondary root detects that the primary root has failed and it initiates the recovery failover automatically. After the failover, the secondary root becomes the active root complex of the PES24NT3 device.

Primary Root Change

Before the NTB upstream port failover, the primary root is the root complex of the PES24NT3 device. The primary root sees three PCI-PCI bridges and the Internal Endpoint inside the PES24NT3 device. There may also be a PCI tree below Port B of the PES24NT3 device. The PCI tree may consists of zero, one or multiple PCIe switches and zero, one of multiple endpoints.

© 2019 Renesas Electronics Corporation

Notes

After the NTB upstream port failover, the primary root is no longer the root complex of the PES24NT3 device. The primary root sees a single External Endpoint inside the PES24NT3. The multiple PCI-PCI bridges, Internal Endpoint and the PCI tree below Port B of the PES24NT4 device disappear. The topology as viewed by the Primary root before and after the failover is shown in Figure 3.

As the primary root has failed, there is no need to recover from the failover.

Secondary Root Change

Before the NTB upstream port failover, the secondary root is not the root complex of the PES24NT3 device. The secondary root sees a single External Endpoint inside the PES24NT3 device.

After the NTB upstream port failover, the secondary root is the root complex of the PES24NT3 device. The secondary root now sees three PCI-PCI bridges and the Internal Endpoint inside the PES24NT3 device. There may also be a PCI tree below Port B of the PES24NT3 device. The PCI tree may consists of zero, one or multiple PCIe switches and zero, one of multiple endpoints. The External Endpoints disappears from the secondary root. The topology as viewed by the Secondary root before and after the failover is shown in Figure 4.

In this example, the Primary root has failed and hence the Secondary root is responsible for initiating the NTB Upstream Port Failover. How the secondary root determines if the Primary root fails is system dependent. A few examples are heartbeats stops or the link that is connected to the Primary root is down.

The procedure to initiate the NTB upstream port failover are:

1. Reach a decision that the Primary root has failed and a NTB upstream port failover is necessary to recover from the failure
2. Initiate the NTB Upstream port failover. At this point, software has to assume that it has no access to the External Endpoint anymore.
3. Initiate a “virtual” unplug of External Endpoint
4. Look for a new PCI-PCI bridge device of the PES24NT3 device
5. When the PCI-PCI bridge device is discovered, run PCI enumeration procedure to discover and configure the three PCI-PCI bridges, Internal Endpoint and the devices in the PCI tree that is connected to port B of the PES24NT3 device.

Watchdog Timer Failover

The primary root has failed and the Watchdog timer has expired. A failover is initiated by hardware automatically because of the Watchdog Timer expiration. The secondary root detects that a failover has happened and it starts the recovery procedure. After the failover, the secondary root becomes the active root complex of the PES24NT3 device.

Primary Root Change

Before the NTB upstream port failover, the primary root is the root complex of the PES24NT3 device. The primary root sees three PCI-PCI bridges and the Internal Endpoint inside the PES24NT3 device. There may also be a PCI tree below Port B of the PES24NT3 device. The PCI tree may consists of zero, one or multiple PCIe switches and zero, one of multiple endpoints.

After the NTB upstream port failover, the primary root is no longer the root complex of the PES24NT3 device. The primary root sees a single External Endpoint inside the PES24NT3. The multiple PCI-PCI bridges, Internal Endpoint and the PCI tree below Port B of the PES24NT4 device disappear. The topology as viewed by the Primary root before and after the failover is shown in Figure 3.

As the primary root has failed, there is no need to recover from the failover.

© 2019 Renesas Electronics Corporation

Notes

Secondary Root Change

Before the NTB upstream port failover, the secondary root is not the root complex of the PES24NT3 device. The secondary root sees a single External Endpoint inside the PES24NT3 device.

After the NTB upstream port failover, the secondary root is the root complex of the PES24NT3 device. The secondary root now sees three PCI-PCI bridges and the Internal Endpoint inside the PES24NT3 device. There may also be a PCI tree below Port B of the PES24NT3 device. The PCI tree may consists of zero, one or multiple PCIe switches and zero, one of multiple endpoints. The External Endpoints disappears from the secondary root. The topology as viewed by the Secondary root before and after the failover is shown in Figure 4.

In this example, the primary root has failed and the watchdog timer causes the NTB Upstream Port Failover. Where there is a failover, the link is down on Port C as well as on Port A. Secondary root gets a link down interrupt on the downstream port that is connected to Port C of PES24NT3.

The procedure to initiate the NTB upstream port failover are:

1. Get a link down interrupt on the downstream port that is connect to Port C of PES24NT3.
2. Look for a new PCI-PCI bridge device of the PES24NT3 device
3. Initiate a “virtual” unplug of External Endpoint
4. When the PCI-PCI bridge device is discovered, run PCI enumeration procedure to discover and configure the three PCI-PCI bridges, Internal Endpoint and the devices in the PCI tree that is connected to port B of the PES24NT3 device.

Software API

A software API is created to make it easier to configure the NTB Upstream Port Failover. The software API is to be delivered as a “C” library and is OS independent. The customer is expected to build their own configuration software or device driver to configure the NTB Upstream Port Failover. The software architecture is shown in Figure 5. At 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 customer. The middle layer is the NTB Upstream Port Failover API that is provided by IDT. It provides API to the user level program for NTB Upstream Port Failover. The NTB Upstream Port Failover API relies on the I/O Access layer to access the PCIe Configuration Registers. The I/O Access layer is system and OS dependent. It makes the appropriate system or OS services to access the PCIe Configuration Registers. A few examples are provided in the I/O Access layer. The examples are Linux user level and Linux kernel level.

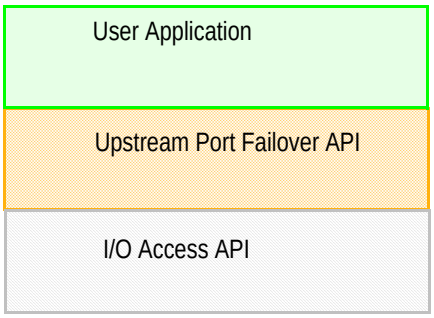

Figure 5 Software Architecture

© 2019 Renesas Electronics Corporation

Notes

API Description

Multiple APIs are provided to support the user application. The NTB Upstream Port Failover layer API uses the I/O Access layer API to read/write from/to the device's configuration registers. I/O Access API is system and OS dependent.

© 2019 Renesas Electronics Corporation

Notes

IdtPesAmIRootComplex()

```
int IdtPesAmIRootComplex
unsigned                bdf,
unsigned               *is_rc;
)
```

Parameters

bdf	The bus, function and device numbers to identify the device to be configured. The function number occupies the lower 3 bits, the device number occupies the next 5 bits and the bus number occupies the next 8 bits. This is the bdf of the NTB endpoint.
is_rc	0 means no. 1 means yes. Others mean error.

Description

It checks if I am the root complex for the Inter-domain switch. I am the root complex if the bdf is the Internal endpoint of the Inter-domain switch.

Returns

PES_OK	bdf is valid. The result is saved in "*is_rc".
PES_INVALID_BDF	bdf is invalided. Could not locate any IDT endpoints of the Inter-domain switch in the system.

© 2019 Renesas Electronics Corporation

Notes

IdtPesGetCurrentFailoverMode()

```
int IdtPesGetCurrentFailoverMode(
unsigned                bdf,
int                     *mode;
)
```

Parameters

bdf	The bus, function and device numbers to identify the device of which port arbitration setup is to be returned. The function number occupies the lower 3 bits, the device number occupies the next 8 bits. This is the bdf of the NTB endpoint.
mode	current failover mode is returned here if there is no error.
PES_FAILOVER_NORMAL	External port A associated with internal port A and external port C associated with internal port C.
PES_FAILOVER_FAILOVER	External port A associated with internal Port C and external port C associated with internal port A.
PES_UNKNOWN_ERROR	Certain unknown error encountered when identifying current failover mode.

Description

It returns the current failover mode.

Returns

PES_OK	failover mode is returned in “*mode”
PES_INVALID_BDF	bdf is invalided. Failover mode is not returned.

© 2019 Renesas Electronics Corporation

Notes

IdtPesGetFailoverControl()

```
int IdtPesGetFailoverControl(
unsigned                bdf
unsigned                *control;
)
```

Parameters

bdf	The bus, function and device numbers to identify the device of which port arbitration setup is to be returned. The function number occupies the lower 3 bits, the device number occupies the next 5 bits and the bus number occupies the next 8 bits. This is the bdf of the NTB endpoint.
control	Failover control register value. Possible bit fields are: FLD_FAILOVER_MODE_SELECT (bit 0) FLD_SIGNAL_FAILOVER_ENABLE (bit 1) FLD_TIMER_FAILOVER_ENABLE (bit 2) FLD_FAILOVER_HOT_RESET_DISABLE (bit 3) FLD_FAILOVER_IHLDHR_DISABLE (bit 4) FLD_FAILOVER_EHLDHR_DISABLE (bit 5) FLD_FAILOVER_IHHRP_DISABLE (bit 6) FLD_FAILOVER_EHHRP_DISABLE (bit 7)

Description

It returns the content of the failover control register in “control”. Please refer to the PES24NT3 User Manual Failover Control (0x22C) register for detailed description of the control bits.

Returns

PES_OK	failover control is returned in “*control”
PES_INVALID_BDF	bdf is invalided. Failover control is not returned.

© 2019 Renesas Electronics Corporation

Notes

IdtPesSetFailoverControl()

```
int IdtPesSetFailoverControl(
unsigned                bdf
unsigned                control
)
```

Parameters

bdf	The bus, function and device numbers to identify the device of which port arbitration setup is to be returned. The function number occupies the lower 3 bits, the device number occupies the next 5 bits and the bus number occupies the next 8 bits. This is the bdf of the NTB endpoint.
control	Failover control. This is a bit field and each field can be “or” to set multiple modes. Possible fields are: FLD_FAILOVER_MODE_SELECT (bit 0) FLD_SIGNAL_FAILOVER_ENABLE (bit 1) FLD_TIMER_FAILOVER_ENABLE (bit 2) FLD_FAILOVER_HOT_RESET_DISABLE (bit 3) FLD_FAILOVER_IHLDHR_DISABLE (bit 4) FLD_FAILOVER_EHLDHR_DISABLE (bit 5) FLD_FAILOVER_IHHRP_DISABLE (bit 6) FLD_FAILOVER_EHHRP_DISABLE (bit 7) Unused bit fields are ignored by this function.

Description

It sets the failover control register. Please refer to the PES24NT3 user manual Failover Control (0x22C) register for detailed description of the control bits

Returns

PES_OK	failover control has been written successful into the failover control register.
PES_INVALID_BDF	bdf is invalided.

© 2019 Renesas Electronics Corporation

Notes

IdtPesGetWatchdogTimer()

```
int IdtPesGetWatchdogTimer(
unsigned                bdf,
unsigned                *value
)
```

Parameters

bdf	The bus, function and device numbers to identify the device of which port arbitration setup is to be returned. The function number occupies the lower 3 bits, the device number occupies the next 5 bits and the bus number occupies the next 8 bits. This is the bdf of the NTB endpoint.
value	Watchdog timer value is returned here. The value is in microsecond.

Description

It returns the value of the watchdog timer in “value”.

Returns

PES_OK	content of the watchdog timer is returned in “*value”.
PES_INVALID_BDF	bdf is invalided.

© 2019 Renesas Electronics Corporation

Notes

IdtPesSetWatchdogTimer()

```
int IdtPesSetWatchdogTimer(
unsigned                bdf,
unsigned                value
)
```

Parameters

bdf	The bus, function and device numbers to identify the device of which port arbitration setup is to be returned. The function number occupies the lower 3 bits, the device number occupies the next 8 bits. This is the bdf of the NTB endpoint.
value	Watchdog timer value to be set. The value is in micro-second.

Description

It sets the “value” to the watchdog timer.

Returns

PES_OK	value has been written to the watchdog time successfully.
PES_INVALID_BDF	bdf is invalided.

© 2019 Renesas Electronics Corporation

Notes

IdtPesGetFailoverSignal()

```
int IdtPesGetFailoverSignal(
unsigned                bdf,
unsigned                *level
)
```

Parameters

bdf	The bus, function and device numbers to identify the device of which port arbitration setup is to be returned. The function number occupies the lower 3 bits, the device number occupies the next 5 bits and the bus number occupies the next 8 bits. This is the bdf of the NTB endpoint.
level	Current signal level. 0: low level 1: high level Otherwise: error

Description

It returns the current failover signal level.

Returns

PES_OK	failover signal is returned in “*level”.
PES_INVALID_BDF	bdf is invalided.

© 2019 Renesas Electronics Corporation

Notes

IdtPesSetFailoverSignal()

```
int IdtPesSetFailoverSignal(
unsigned                bdf,
unsigned                level
)
```

Parameters

bdf	The bus, function and device numbers to identify the device of which port arbitration setup is to be returned. The function number occupies the lower 3 bits, the device number occupies the next 5 bits and the bus number occupies the next 8 bits. This is the bdf of the NTB endpoint.
level	Set the current signal level to match this level. 0: to low level. 1: to high level. Otherwise: error.

Description

It sets the failover signal to initiate a failover.

Returns

PES_OK	Failover signal has been set according to level.
PES_INVALID_BDF	bdf is invalided.

© 2019 Renesas Electronics Corporation

Notes

IdtPesEnableSignalFailover()

```
int IdtPesEnableSignalFailover
unsigned              bdf,
)
```

Parameters

bdf	The bus, function and device numbers to identify the device of which port arbitration setup is to be returned. The function number occupies the lower 3 bits, the device number occupies the next 5 bits and the bus number occupies the next 8 bits. This is the bdf of the NTB endpoint.

Description

It disables the timer failover process by writing 1 to the SIGFEN field of Failover Control register. It should be noted that the failover may not be completed when this function returns.

Returns

PES_OK	The signal failover has been enabled.
PES_INVALID_BDF	bdf is invalid.

© 2019 Renesas Electronics Corporation

Notes

IdtPesDisableSignalFailover()

```
int IdtPesDisableSignalFailover
unsigned                bdf,
)
```

Parameters

bdf	The bus, function and device numbers to identify the device of which port arbitration setup is to be returned. The function number occupies the lower 3 bits, the device number occupies the next 5 bits and the bus number occupies the next 8 bits. This is the bdf of the NTB endpoint.

Description

It disables the timer failover process by writing 0 to the SIGFEN field of Failover Control register. It should be noted that the failover may not be completed when this function returns.

Returns

PES_OK	The signal failover has been disabled.
PES_INVALID_BDF	bdf is invalid.

© 2019 Renesas Electronics Corporation

Notes

IdtPesInitiateSignalFailover()

```
int IdtPesInitiateSignalFailover
unsigned                bdf,
)
```

Parameters

bdf	The bus, function and device numbers to identify the device of which port arbitration setup is to be returned. The function number occupies the lower 3 bits, the device number occupies the next 5 bits and the bus number occupies the next 8 bits. This is the bdf of the NTB endpoint.

Description

It initiates the failover process via signal only when SIGFEN of FOVRCTL register is 1. The alternate function of GPIO[5] is used as the failover signal. It should be noted that the failover may not be completed when this function returns.

Returns

PES_OK	The failover has been initiated.
PES_INVALID_BDF	bdf is invalid.

© 2019 Renesas Electronics Corporation

Notes

IdtPesInitiateConfigFailover()

```
int IdtPesInitiateConfigFailover
unsigned                bdf,
)
```

Parameters

bdf	The bus, function and device numbers to identify the device of which port arbitration setup is to be returned. The function number occupies the lower 3 bits, the device number occupies the next 5 bits and the bus number occupies the next 8 bits. This is the bdf of the NTB endpoint.

Description

It initiates the failover process by modifying value of bit field FOVRMSEL of the Failover Control register. It should be noted that the failover may not be completed when this function returns.

Returns

PES_OK	The failover has been initiated.
PES_INVALID_BDF	bdf is invalid.

© 2019 Renesas Electronics Corporation

Notes

IdtPesEnableTimerFailover()

```
int IdtPesEnableTimerFailover
unsigned                bdf,
)
```

Parameters

bdf	The bus, function and device numbers to identify the device of which port arbitration setup is to be returned. The function number occupies the lower 3 bits, the device number occupies the next 5 bits and the bus number occupies the next 8 bits. This is the bdf of the NTB endpoint.

Description

It enables the timer failover process by writing 1 to the TIMFEN field of Failover Control register. It should be noted that the failover may not be completed when this function returns.

Returns

PES_OK	The timer failover has been enabled.
PES_INVALID_BDF	bdf is invalid.

© 2019 Renesas Electronics Corporation

Notes

IdtPesDisableTimerFailover()

```
int IdtPesDisableTimerFailover
unsigned                bdf,
)
```

Parameters

bdf	The bus, function and device numbers to identify the device of which port arbitration setup is to be returned. The function number occupies the lower 3 bits, the device number occupies the next 5 bits and the bus number occupies the next 8 bits. This is the bdf of the NTB endpoint.

Description

It disables the timer failover process by writing 0 to the TIMFEN field of Failover Control register. It should be noted that the failover may not be completed when this function returns.

Returns

PES_OK	The timer failover has been disabled.
PES_INVALID_BDF	bdf is invalid.

© 2019 Renesas Electronics Corporation

Notes

IdtPesMakeBDF()

```
unsigned IdtPesMakeBDF(
unsigned              bus;
unsigned              device;
unsigned              function
)
```

Parameters

bus	The bus number of a device ID. Only the lower 8 bits of the bus number is used.
device	The device number of a device ID. Only the lower 5 bits of the device number is used.
function	The function number of a device ID. Only the lower 3 bits of the function number is used.

Description

It takes the bus, device and function numbers of a device ID to form a single bdf value that can be used to call the API as specified in this document. The function number occupies the lower 3 bits, the device number occupies the next 5 bits and the bus number occupies the next 8 bits.

Returns

`bdf`	The device ID that can be used to call the API as specified in this document.

I/O Access API

The I/O access API hides the system and OS dependency from the Port Arbitration layer API to access both the PCI Configuration Space and the PCI Express Extended Configuration Space of a PCIe device. The I/O Access API is to be provided by the platform developer and not by IDT.

IDT provides sample implementation of I/O Access API for the platform that IDT uses to develop this API. The sample implementation is base on Fedora 6 Linux.

© 2019 Renesas Electronics Corporation

Notes

ioa_pci_read_config()

```
int ioa_pci_read_config(
unsigned                bdf,
unsigned                offset,
unsigned                *value
)
```

Parameters

bdf	The bus, function and device numbers to identify the device of which port arbitration setup is to be returned. The function number occupies the lower 3 bits, the device number occupies the next 5 bits and the bus number occupies the next 8 bits. This is the bdf of the NTB endpoint.
offset	To specify which register to read. The offset may be between 0 and 0xFFC and must be a multiple of 4 bytes.
value	The value of the register is returned here.

Description

It looks for the device as specified in the bdf and returns the configuration space register as specified in offset.

Returns

IOA_OK	Success.
IOA_BAD_BDF	bdf does not identify a valid device.
IOA_INVALID_OFFSET	offset is invalid.
IOA_UNKNOWN_ERROR	Unknown error.

© 2019 Renesas Electronics Corporation

Notes

ioa_pci_write_config()

```
int ioa_pci_read_config(
unsigned                bdf,
unsigned                offset,
unsigned                value
)
```

Parameters

bdf	The bus, function and device numbers to identify the device of which port arbitration setup is to be returned. The function number occupies the lower 3 bits, the device number occupies the next 5 bits and the bus number occupies the next 8 bits. This is the bdf of the NTB endpoint.
offset	To specify which register to read. The offset may be between 0 and 0xFFC and must be a multiple of 4 bytes.
value	To be written to the register

Description

It looks for the device as specified in the bdf and write the value to the configuration space register as specified in offset.

Returns

IOA_OK	Success.
IOA_BAD_BDF	bdf does not identify a valid device.
IOA_INVALID_OFFSET	offset is invalid.
IOA_UNKNOWN_ERROR	Unknown error.

© 2019 Renesas Electronics Corporation

Notes

ioa_get_failover_signal_level()

```
int ioa_get_failover_signal_level(
unsigned                bdf,
boolean                 *level
)
```

Parameters

`bdf`	The bus, function and device numbers to identify the device of which port arbitration setup is to be returned. The function number occupies the lower 3 bits, the device number occupies the next 5 bits and the bus number occupies the next 8 bits. This is the bdf of the NTB endpoint.
`level`	The current failover signal level: 0: low level(0) 1: high level(1) otherwise: error

Description

It returns the current failover signal level in “level”.

Returns

`IOA_OK`	Success.
`IOA_BAD_BDF`	bdf does not identify a valid device.

© 2019 Renesas Electronics Corporation

Notes

ioa_set_failover_signal_level()

```
int ioa_set_failover_signal_level(
unsigned                bdf,
boolean                 level
)
```

Parameters

`bdf`	The bus, function and device numbers to identify the device of which port arbitration setup is to be returned. The function number occupies the lower 3 bits, the device number occupies the next 5 bits and the bus number occupies the next 8 bits. This is the bdf of the NTB endpoint.
`level`	Set the failover signal level to this level 0: low level (0) 1: high level (1) otherwise: error

Description

It sets the failover signal level to “level”.

Returns

`IOA_OK`	Success.
`IOA_BAD_BDF`	bdf does not identify a valid device.

Linux Kernel IO Access API

When the user program is running in the Linux kernel space, the IO Access API makes use of the Linux system provided API to access the PCI configuration space:

- pci_read_config_dword()
- pci_write_config_dword()

Linux user level IO Access API

When the user program is running in the Linux user space, the IO Access API makes use of the libpci (or pciutil) library to access the PCI configuration space. The following libpci functions are used:

- pci_read_long()
- pci_write_long()

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)

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.

© 2025 Renesas Electronics Corporation. All rights reserved.