

RapidIO Subsystem Guide

Matt Porter <mporter@kernel.crashing.org>

RapidIO Subsystem Guide

by Matt Porter

Copyright © 2005 MontaVista Software, Inc.

This documentation is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

For more details see the file COPYING in the source distribution of Linux.

Table of Contents

| | |
|-------------------------------------|-----|
| 1. Introduction | 1 |
| 2. Known Bugs and Limitations | 2 |
| Bugs | 2 |
| Limitations | 2 |
| 3. RapidIO driver interface | 3 |
| Functions | 3 |
| 4. Internals | 74 |
| Structures | 74 |
| Enumeration and Discovery | 89 |
| Driver functionality | 114 |
| Device model support | 128 |
| Sysfs support | 133 |
| PPC32 support | 135 |
| 5. Credits | 141 |

Chapter 1. Introduction

RapidIO is a high speed switched fabric interconnect with features aimed at the embedded market. RapidIO provides support for memory-mapped I/O as well as message-based transactions over the switched fabric network. RapidIO has a standardized discovery mechanism not unlike the PCI bus standard that allows simple detection of devices in a network.

This documentation is provided for developers intending to support RapidIO on new architectures, write new drivers, or to understand the subsystem internals.

Chapter 2. Known Bugs and Limitations

Bugs

None. ;)

Limitations

1. Access/management of RapidIO memory regions is not supported
2. Multiple host enumeration is not supported

Chapter 3. RapidIO driver interface

Drivers are provided a set of calls in order to interface with the subsystem to gather info on devices, request/map memory region resources, and manage mailboxes/doorbells.

Functions

Name

`rio_local_read_config_32` — Read 32 bits from local configuration space

Synopsis

```
int rio_local_read_config_32 (struct rio_mport * port, u32 offset, u32  
* data);
```

Arguments

| | |
|---------------|---------------------------------------|
| <i>port</i> | Master port |
| <i>offset</i> | Offset into local configuration space |
| <i>data</i> | Pointer to read data into |

Description

Reads 32 bits of data from the specified offset within the local device's configuration space.

Name

`rio_local_write_config_32` — Write 32 bits to local configuration space

Synopsis

```
int rio_local_write_config_32 (struct rio_mport * port, u32 offset, u32  
data);
```

Arguments

| | |
|---------------|---------------------------------------|
| <i>port</i> | Master port |
| <i>offset</i> | Offset into local configuration space |
| <i>data</i> | Data to be written |

Description

Writes 32 bits of data to the specified offset within the local device's configuration space.

Name

`rio_local_read_config_16` — Read 16 bits from local configuration space

Synopsis

```
int rio_local_read_config_16 (struct rio_mport * port, u32 offset, u16  
* data);
```

Arguments

| | |
|---------------|---------------------------------------|
| <i>port</i> | Master port |
| <i>offset</i> | Offset into local configuration space |
| <i>data</i> | Pointer to read data into |

Description

Reads 16 bits of data from the specified offset within the local device's configuration space.

Name

`rio_local_write_config_16` — Write 16 bits to local configuration space

Synopsis

```
int rio_local_write_config_16 (struct rio_mport * port, u32 offset, u16  
data);
```

Arguments

| | |
|---------------|---------------------------------------|
| <i>port</i> | Master port |
| <i>offset</i> | Offset into local configuration space |
| <i>data</i> | Data to be written |

Description

Writes 16 bits of data to the specified offset within the local device's configuration space.

Name

`rio_local_read_config_8` — Read 8 bits from local configuration space

Synopsis

```
int rio_local_read_config_8 (struct rio_mport * port, u32 offset, u8  
* data);
```

Arguments

| | |
|---------------|---------------------------------------|
| <i>port</i> | Master port |
| <i>offset</i> | Offset into local configuration space |
| <i>data</i> | Pointer to read data into |

Description

Reads 8 bits of data from the specified offset within the local device's configuration space.

Name

`rio_local_write_config_8` — Write 8 bits to local configuration space

Synopsis

```
int rio_local_write_config_8 (struct rio_mport * port, u32 offset, u8  
data);
```

Arguments

| | |
|---------------|---------------------------------------|
| <i>port</i> | Master port |
| <i>offset</i> | Offset into local configuration space |
| <i>data</i> | Data to be written |

Description

Writes 8 bits of data to the specified offset within the local device's configuration space.

Name

`rio_read_config_32` — Read 32 bits from configuration space

Synopsis

```
int rio_read_config_32 (struct rio_dev * rdev, u32 offset, u32 * data);
```

Arguments

| | |
|---------------|--|
| <i>rdev</i> | RIO device |
| <i>offset</i> | Offset into device configuration space |
| <i>data</i> | Pointer to read data into |

Description

Reads 32 bits of data from the specified offset within the RIO device's configuration space.

Name

`rio_write_config_32` — Write 32 bits to configuration space

Synopsis

```
int rio_write_config_32 (struct rio_dev * rdev, u32 offset, u32 data);
```

Arguments

rdev RIO device

offset Offset into device configuration space

data Data to be written

Description

Writes 32 bits of data to the specified offset within the RIO device's configuration space.

Name

`rio_read_config_16` — Read 16 bits from configuration space

Synopsis

```
int rio_read_config_16 (struct rio_dev * rdev, u32 offset, u16 * data);
```

Arguments

| | |
|---------------|--|
| <i>rdev</i> | RIO device |
| <i>offset</i> | Offset into device configuration space |
| <i>data</i> | Pointer to read data into |

Description

Reads 16 bits of data from the specified offset within the RIO device's configuration space.

Name

`rio_write_config_16` — Write 16 bits to configuration space

Synopsis

```
int rio_write_config_16 (struct rio_dev * rdev, u32 offset, u16 data);
```

Arguments

| | |
|---------------|--|
| <i>rdev</i> | RIO device |
| <i>offset</i> | Offset into device configuration space |
| <i>data</i> | Data to be written |

Description

Writes 16 bits of data to the specified offset within the RIO device's configuration space.

Name

`rio_read_config_8` — Read 8 bits from configuration space

Synopsis

```
int rio_read_config_8 (struct rio_dev * rdev, u32 offset, u8 * data);
```

Arguments

| | |
|---------------|--|
| <i>rdev</i> | RIO device |
| <i>offset</i> | Offset into device configuration space |
| <i>data</i> | Pointer to read data into |

Description

Reads 8 bits of data from the specified offset within the RIO device's configuration space.

Name

`rio_write_config_8` — Write 8 bits to configuration space

Synopsis

```
int rio_write_config_8 (struct rio_dev * rdev, u32 offset, u8 data);
```

Arguments

rdev RIO device

offset Offset into device configuration space

data Data to be written

Description

Writes 8 bits of data to the specified offset within the RIO device's configuration space.

Name

`rio_send_doorbell` — Send a doorbell message to a device

Synopsis

```
int rio_send_doorbell (struct rio_dev * rdev, u16 data);
```

Arguments

rdev RIO device

data Doorbell message data

Description

Send a doorbell message to a RIO device. The doorbell message has a 16-bit info field provided by the *data* argument.

Name

`rio_init_mbox_res` — Initialize a RIO mailbox resource

Synopsis

```
void rio_init_mbox_res (struct resource * res, int start, int end);
```

Arguments

res resource struct

start start of mailbox range

end end of mailbox range

Description

This function is used to initialize the fields of a resource for use as a mailbox resource. It initializes a range of mailboxes using the start and end arguments.

Name

`rio_init_dbell_res` — Initialize a RIO doorbell resource

Synopsis

```
void rio_init_dbell_res (struct resource * res, u16 start, u16 end);
```

Arguments

res resource struct

start start of doorbell range

end end of doorbell range

Description

This function is used to initialize the fields of a resource for use as a doorbell resource. It initializes a range of doorbell messages using the start and end arguments.

Name

`RIO_DEVICE` — macro used to describe a specific RIO device

Synopsis

```
RIO_DEVICE ( dev, ven );
```

Arguments

dev the 16 bit RIO device ID

ven the 16 bit RIO vendor ID

Description

This macro is used to create a struct `rio_device_id` that matches a specific device. The assembly vendor and assembly device fields will be set to `RIO_ANY_ID`.

Name

`rio_add_outb_message` — Add RIO message to an outbound mailbox queue

Synopsis

```
int rio_add_outb_message (struct rio_mport * mport, struct rio_dev *  
rdev, int mbox, void * buffer, size_t len);
```

Arguments

| | |
|---------------|---|
| <i>mport</i> | RIO master port containing the outbound queue |
| <i>rdev</i> | RIO device the message is be sent to |
| <i>mbox</i> | The outbound mailbox queue |
| <i>buffer</i> | Pointer to the message buffer |
| <i>len</i> | Length of the message buffer |

Description

Adds a RIO message buffer to an outbound mailbox queue for transmission. Returns 0 on success.

Name

`rio_add_inb_buffer` — Add buffer to an inbound mailbox queue

Synopsis

```
int rio_add_inb_buffer (struct rio_mport * mport, int mbox, void *  
buffer);
```

Arguments

mport Master port containing the inbound mailbox

mbox The inbound mailbox number

buffer Pointer to the message buffer

Description

Adds a buffer to an inbound mailbox queue for reception. Returns 0 on success.

Name

`rio_get_inb_message` — Get A RIO message from an inbound mailbox queue

Synopsis

```
void * rio_get_inb_message (struct rio_mport * mport, int mbox);
```

Arguments

mport Master port containing the inbound mailbox

mbox The inbound mailbox number

Description

Get a RIO message from an inbound mailbox queue. Returns 0 on success.

Name

`rio_name` — Get the unique RIO device identifier

Synopsis

```
const char * rio_name (struct rio_dev * rdev);
```

Arguments

rdev RIO device

Description

Get the unique RIO device identifier. Returns the device identifier string.

Name

`rio_get_drvdata` — Get RIO driver specific data

Synopsis

```
void * rio_get_drvdata (struct rio_dev * rdev);
```

Arguments

rdev RIO device

Description

Get RIO driver specific data. Returns a pointer to the driver specific data.

Name

`rio_set_drvdata` — Set RIO driver specific data

Synopsis

```
void rio_set_drvdata (struct rio_dev * rdev, void * data);
```

Arguments

rdev RIO device

data Pointer to driver specific data

Description

Set RIO driver specific data. device struct driver data pointer is set to the *data* argument.

Name

`rio_dev_get` — Increments the reference count of the RIO device structure

Synopsis

```
struct rio_dev * rio_dev_get (struct rio_dev * rdev);
```

Arguments

rdev RIO device being referenced

Description

Each live reference to a device should be refcounted.

Drivers for RIO devices should normally record such references in their probe methods, when they bind to a device, and release them by calling `rio_dev_put`, in their disconnect methods.

Name

`rio_dev_put` — Release a use of the RIO device structure

Synopsis

```
void rio_dev_put (struct rio_dev * rdev);
```

Arguments

rdev RIO device being disconnected

Description

Must be called when a user of a device is finished with it. When the last user of the device calls this function, the memory of the device is freed.

Name

`rio_register_driver` — register a new RIO driver

Synopsis

```
int rio_register_driver (struct rio_driver * rdrv);
```

Arguments

rdrv the RIO driver structure to register

Description

Adds a struct `rio_driver` to the list of registered drivers. Returns a negative value on error, otherwise 0. If no error occurred, the driver remains registered even if no device was claimed during registration.

Name

`rio_unregister_driver` — unregister a RIO driver

Synopsis

```
void rio_unregister_driver (struct rio_driver * rdrv);
```

Arguments

rdrv the RIO driver structure to unregister

Description

Deletes the struct `rio_driver` from the list of registered RIO drivers, gives it a chance to clean up by calling its `remove` function for each device it was responsible for, and marks those devices as `driverless`.

Name

`rio_local_get_device_id` — Get the base/extended device id for a port

Synopsis

```
ul64 rio_local_get_device_id (struct rio_mport * port);
```

Arguments

port RIO master port from which to get the deviceid

Description

Reads the base/extended device id from the local device implementing the master port. Returns the 8/16-bit device id.

Name

`rio_query_mport` — Query mport device attributes

Synopsis

```
int rio_query_mport (struct rio_mport * port, struct rio_mport_attr *  
mport_attr);
```

Arguments

port mport device to query

mport_attr mport attributes data structure

Description

Returns attributes of specified mport through the pointer to attributes data structure.

Name

`rio_alloc_net` — Allocate and initialize a new RIO network data structure

Synopsis

```
struct rio_net * rio_alloc_net (struct rio_mport * mport);
```

Arguments

mport Master port associated with the RIO network

Description

Allocates a RIO network structure, initializes per-network list heads, and adds the associated master port to the network list of associated master ports. Returns a RIO network pointer on success or NULL on failure.

Name

`rio_local_set_device_id` — Set the base/extended device id for a port

Synopsis

```
void rio_local_set_device_id (struct rio_mport * port, u16 did);
```

Arguments

port RIO master port

did Device ID value to be written

Description

Writes the base/extended device id from a device.

Name

`rio_add_device` — Adds a RIO device to the device model

Synopsis

```
int rio_add_device (struct rio_dev * rdev);
```

Arguments

rdev RIO device

Description

Adds the RIO device to the global device list and adds the RIO device to the RIO device list. Creates the generic sysfs nodes for an RIO device.

Name

`rio_request_inb_mbox` — request inbound mailbox service

Synopsis

```
int rio_request_inb_mbox (struct rio_mport * mport, void * dev_id, int
mbox, int entries, void (*minb) (struct rio_mport * mport, void *dev_id,
int mbox, int slot));
```

Arguments

mport RIO master port from which to allocate the mailbox resource

dev_id Device specific pointer to pass on event

mbox Mailbox number to claim

entries Number of entries in inbound mailbox queue

minb Callback to execute when inbound message is received

Description

Requests ownership of an inbound mailbox resource and binds a callback function to the resource. Returns 0 on success.

Name

`rio_release_inb_mbox` — release inbound mailbox message service

Synopsis

```
int rio_release_inb_mbox (struct rio_mport * mport, int mbox);
```

Arguments

mport RIO master port from which to release the mailbox resource

mbox Mailbox number to release

Description

Releases ownership of an inbound mailbox resource. Returns 0 if the request has been satisfied.

Name

`rio_request_outb_mbox` — request outbound mailbox service

Synopsis

```
int rio_request_outb_mbox (struct rio_mport * mport, void * dev_id,
int mbox, int entries, void (*moutb) (struct rio_mport * mport, void
*dev_id, int mbox, int slot));
```

Arguments

mport RIO master port from which to allocate the mailbox resource

dev_id Device specific pointer to pass on event

mbox Mailbox number to claim

entries Number of entries in outbound mailbox queue

moutb Callback to execute when outbound message is sent

Description

Requests ownership of an outbound mailbox resource and binds a callback function to the resource. Returns 0 on success.

Name

`rio_release_outb_mbox` — release outbound mailbox message service

Synopsis

```
int rio_release_outb_mbox (struct rio_mport * mport, int mbox);
```

Arguments

mport RIO master port from which to release the mailbox resource

mbox Mailbox number to release

Description

Releases ownership of an inbound mailbox resource. Returns 0 if the request has been satisfied.

Name

`rio_request_inb_dbell` — request inbound doorbell message service

Synopsis

```
int rio_request_inb_dbell (struct rio_mport * mport, void * dev_id, u16
start, u16 end, void (*dinb) (struct rio_mport * mport, void *dev_id,
u16 src, u16 dst, u16 info));
```

Arguments

mport RIO master port from which to allocate the doorbell resource

dev_id Device specific pointer to pass on event

start Doorbell info range start

end Doorbell info range end

dinb Callback to execute when doorbell is received

Description

Requests ownership of an inbound doorbell resource and binds a callback function to the resource. Returns 0 if the request has been satisfied.

Name

`rio_release_inb_dbell` — release inbound doorbell message service

Synopsis

```
int rio_release_inb_dbell (struct rio_mport * mport, u16 start, u16
end);
```

Arguments

mport RIO master port from which to release the doorbell resource

start Doorbell info range start

end Doorbell info range end

Description

Releases ownership of an inbound doorbell resource and removes callback from the doorbell event list. Returns 0 if the request has been satisfied.

Name

`rio_request_outb_dbell` — request outbound doorbell message range

Synopsis

```
struct resource * rio_request_outb_dbell (struct rio_dev * rdev, u16
start, u16 end);
```

Arguments

rdev RIO device from which to allocate the doorbell resource

start Doorbell message range start

end Doorbell message range end

Description

Requests ownership of a doorbell message range. Returns a resource if the request has been satisfied or NULL on failure.

Name

`rio_release_outb_dbell` — release outbound doorbell message range

Synopsis

```
int rio_release_outb_dbell (struct rio_dev * rdev, struct resource *  
res);
```

Arguments

rdev RIO device from which to release the doorbell resource

res Doorbell resource to be freed

Description

Releases ownership of a doorbell message range. Returns 0 if the request has been satisfied.

Name

`rio_add_mport_pw_handler` — add port-write message handler into the list of mport specific pw handlers

Synopsis

```
int rio_add_mport_pw_handler (struct rio_mport * mport, void * context,  
int (*pwcallback) (struct rio_mport *mport, void *context, union rio_pw_msg  
*msg, int step));
```

Arguments

mport RIO master port to bind the portwrite callback

context Handler specific context to pass on event

pwcallback Callback to execute when portwrite is received

Description

Returns 0 if the request has been satisfied.

Name

`rio_del_mport_pw_handler` — remove port-write message handler from the list of mport specific pw handlers

Synopsis

```
int rio_del_mport_pw_handler (struct rio_mport * mport, void * context,
int (*pwcallback) (struct rio_mport *mport, void *context, union rio_pw_msg
*msg, int step));
```

Arguments

mport RIO master port to bind the portwrite callback

context Registered handler specific context to pass on event

pwcallback Registered callback function

Description

Returns 0 if the request has been satisfied.

Name

`rio_request_inb_pwrite` — request inbound port-write message service for specific RapidIO device

Synopsis

```
int rio_request_inb_pwrite (struct rio_dev * rdev, int (*pwcallback)
(struct rio_dev *rdev, union rio_pw_msg *msg, int step));
```

Arguments

rdev RIO device to which register inbound port-write callback routine

pwcallback Callback routine to execute when port-write is received

Description

Binds a port-write callback function to the RapidIO device. Returns 0 if the request has been satisfied.

Name

`rio_release_inb_pwrite` — release inbound port-write message service associated with specific RapidIO device

Synopsis

```
int rio_release_inb_pwrite (struct rio_dev * rdev);
```

Arguments

rdev RIO device which registered for inbound port-write callback

Description

Removes callback from the `rio_dev` structure. Returns 0 if the request has been satisfied.

Name

`rio_pw_enable` — Enables/disables port-write handling by a master port

Synopsis

```
void rio_pw_enable (struct rio_mport * mport, int enable);
```

Arguments

mport Master port associated with port-write handling

enable 1=enable, 0=disable

Name

`rio_map_inb_region` — - Map inbound memory region.

Synopsis

```
int rio_map_inb_region (struct rio_mport * mport, dma_addr_t local, u64  
rbase, u32 size, u32 rflags);
```

Arguments

| | |
|---------------|--|
| <i>mport</i> | Master port. |
| <i>local</i> | physical address of memory region to be mapped |
| <i>rbase</i> | RIO base address assigned to this window |
| <i>size</i> | Size of the memory region |
| <i>rflags</i> | Flags for mapping. |

Return

0 -- Success.

This function will create the mapping from RIO space to local memory.

Name

`rio_unmap_inb_region` — - Unmap the inbound memory region

Synopsis

```
void rio_unmap_inb_region (struct rio_mport * mport, dma_addr_t lstart);
```

Arguments

mport Master port

lstart physical address of memory region to be unmapped

Name

`rio_map_outb_region` — - Map outbound memory region.

Synopsis

```
int rio_map_outb_region (struct rio_mport * mport, u16 destid, u64  
rbase, u32 size, u32 rflags, dma_addr_t * local);
```

Arguments

mport Master port.

destid destination id window points to

rbase RIO base address window translates to

size Size of the memory region

rflags Flags for mapping.

local physical address of memory region mapped

Return

0 -- Success.

This function will create the mapping from RIO space to local memory.

Name

`rio_unmap_outb_region` — - Unmap the inbound memory region

Synopsis

```
void rio_unmap_outb_region (struct rio_mport * mport, u16 destid, u64  
rstart);
```

Arguments

mport Master port

destid destination id mapping points to

rstart RIO base address window translates to

Name

`rio_mport_get_physefb` — Helper function that returns register offset for Physical Layer Extended Features Block.

Synopsis

```
u32 rio_mport_get_physefb (struct rio_mport * port, int local, u16  
destid, u8 hopcount);
```

Arguments

| | |
|-----------------|--|
| <i>port</i> | Master port to issue transaction |
| <i>local</i> | Indicate a local master port or remote device access |
| <i>destid</i> | Destination ID of the device |
| <i>hopcount</i> | Number of switch hops to the device |

Name

`rio_get_comptag` — Begin or continue searching for a RIO device by component tag

Synopsis

```
struct rio_dev * rio_get_comptag (u32 comp_tag, struct rio_dev * from);
```

Arguments

comp_tag RIO component tag to match

from Previous RIO device found in search, or NULL for new search

Description

Iterates through the list of known RIO devices. If a RIO device is found with a matching *comp_tag*, a pointer to its device structure is returned. Otherwise, NULL is returned. A new search is initiated by passing NULL to the *from* argument. Otherwise, if *from* is not NULL, searches continue from next device on the global list.

Name

`rio_set_port_lockout` — Sets/clears LOCKOUT bit (RIO EM 1.3) for a switch port.

Synopsis

```
int rio_set_port_lockout (struct rio_dev * rdev, u32 pnum, int lock);
```

Arguments

rdev Pointer to RIO device control structure

pnum Switch port number to set LOCKOUT bit

lock Operation : set (=1) or clear (=0)

Name

`rio_enable_rx_tx_port` — enable input receiver and output transmitter of given port

Synopsis

```
int rio_enable_rx_tx_port (struct rio_mport * port, int local, u16  
destid, u8 hopcount, u8 port_num);
```

Arguments

| | |
|-----------------|--|
| <i>port</i> | Master port associated with the RIO network |
| <i>local</i> | <code>local=1</code> select local port otherwise a far device is reached |
| <i>destid</i> | Destination ID of the device to check host bit |
| <i>hopcount</i> | Number of hops to reach the target |
| <i>port_num</i> | Port (-number on switch) to enable on a far end device |

Description

Returns 0 or 1 from on General Control Command and Status Register (EXT_PTR+0x3C)

Name

`rio_mport_chk_dev_access` — Validate access to the specified device.

Synopsis

```
int rio_mport_chk_dev_access (struct rio_mport * mport, u16 destid, u8  
hopcount);
```

Arguments

| | |
|-----------------|----------------------------------|
| <i>mport</i> | Master port to send transactions |
| <i>destid</i> | Device destination ID in network |
| <i>hopcount</i> | Number of hops into the network |

Name

`rio_inb_pwrite_handler` — inbound port-write message handler

Synopsis

```
int rio_inb_pwrite_handler (struct rio_mport * mport, union rio_pw_msg  
* pw_msg);
```

Arguments

mport mport device associated with port-write

pw_msg pointer to inbound port-write message

Description

Processes an inbound port-write message. Returns 0 if the request has been satisfied.

Name

`rio_mport_get_efb` — get pointer to next extended features block

Synopsis

```
u32 rio_mport_get_efb (struct rio_mport * port, int local, u16 destid,  
u8 hopcount, u32 from);
```

Arguments

| | |
|-----------------|--|
| <i>port</i> | Master port to issue transaction |
| <i>local</i> | Indicate a local master port or remote device access |
| <i>destid</i> | Destination ID of the device |
| <i>hopcount</i> | Number of switch hops to the device |
| <i>from</i> | Offset of current Extended Feature block header (if 0 starts from ExtFeaturePtr) |

Name

`rio_mport_get_feature` — query for devices' extended features

Synopsis

```
u32 rio_mport_get_feature (struct rio_mport * port, int local, u16  
destid, u8 hopcount, int ftr);
```

Arguments

| | |
|-----------------|--|
| <i>port</i> | Master port to issue transaction |
| <i>local</i> | Indicate a local master port or remote device access |
| <i>destid</i> | Destination ID of the device |
| <i>hopcount</i> | Number of switch hops to the device |
| <i>ftr</i> | Extended feature code |

Description

Tell if a device supports a given RapidIO capability. Returns the offset of the requested extended feature block within the device's RIO configuration space or 0 in case the device does not support it. Possible values for *ftr*:

`RIO_EFB_PAR_EP_ID` LP/LVDS EP Devices

`RIO_EFB_PAR_EP_REC_ID` LP/LVDS EP Recovery Devices

`RIO_EFB_PAR_EP_FREE_ID` LP/LVDS EP Free Devices

`RIO_EFB_SER_EP_ID` LP/Serial EP Devices

`RIO_EFB_SER_EP_REC_ID` LP/Serial EP Recovery Devices

`RIO_EFB_SER_EP_FREE_ID` LP/Serial EP Free Devices

Name

`rio_get_asm` — Begin or continue searching for a RIO device by `vid/did/asm_vid/asm_did`

Synopsis

```
struct rio_dev * rio_get_asm (u16 vid, u16 did, u16 asm_vid, u16 asm_did,  
struct rio_dev * from);
```

Arguments

| | |
|----------------|---|
| <i>vid</i> | RIO vid to match or <code>RIO_ANY_ID</code> to match all vids |
| <i>did</i> | RIO did to match or <code>RIO_ANY_ID</code> to match all dids |
| <i>asm_vid</i> | RIO <code>asm_vid</code> to match or <code>RIO_ANY_ID</code> to match all <code>asm_vids</code> |
| <i>asm_did</i> | RIO <code>asm_did</code> to match or <code>RIO_ANY_ID</code> to match all <code>asm_dids</code> |
| <i>from</i> | Previous RIO device found in search, or <code>NULL</code> for new search |

Description

Iterates through the list of known RIO devices. If a RIO device is found with a matching *vid*, *did*, *asm_vid*, *asm_did*, the reference count to the device is incremented and a pointer to its device structure is returned. Otherwise, `NULL` is returned. A new search is initiated by passing `NULL` to the *from* argument. Otherwise, if *from* is not `NULL`, searches continue from next device on the global list. The reference count for *from* is always decremented if it is not `NULL`.

Name

`rio_get_device` — Begin or continue searching for a RIO device by vid/did

Synopsis

```
struct rio_dev * rio_get_device (ul6 vid, ul6 did, struct rio_dev *  
from);
```

Arguments

vid RIO vid to match or `RIO_ANY_ID` to match all vids

did RIO did to match or `RIO_ANY_ID` to match all dids

from Previous RIO device found in search, or `NULL` for new search

Description

Iterates through the list of known RIO devices. If a RIO device is found with a matching *vid* and *did*, the reference count to the device is incremented and a pointer to its device structure is returned. Otherwise, `NULL` is returned. A new search is initiated by passing `NULL` to the *from* argument. Otherwise, if *from* is not `NULL`, searches continue from next device on the global list. The reference count for *from* is always decremented if it is not `NULL`.

Name

`rio_lock_device` — Acquires host device lock for specified device

Synopsis

```
int rio_lock_device (struct rio_mport * port, u16 destid, u8 hopcount,  
int wait_ms);
```

Arguments

| | |
|-----------------|--|
| <i>port</i> | Master port to send transaction |
| <i>destid</i> | Destination ID for device/switch |
| <i>hopcount</i> | Hopcount to reach switch |
| <i>wait_ms</i> | Max wait time in msec (0 = no timeout) |

Description

Attempts to acquire host device lock for specified device Returns 0 if device lock acquired or EINVAL if timeout expires.

Name

`rio_unlock_device` — Releases host device lock for specified device

Synopsis

```
int rio_unlock_device (struct rio_mport * port, u16 destid, u8 hopcount);
```

Arguments

| | |
|-----------------|----------------------------------|
| <i>port</i> | Master port to send transaction |
| <i>destid</i> | Destination ID for device/switch |
| <i>hopcount</i> | Hopcount to reach switch |

Description

Returns 0 if device lock released or EINVAL if fails.

Name

`rio_route_add_entry` — Add a route entry to a switch routing table

Synopsis

```
int rio_route_add_entry (struct rio_dev * rdev, u16 table, u16
route_destid, u8 route_port, int lock);
```

Arguments

| | |
|---------------------|---|
| <i>rdev</i> | RIO device |
| <i>table</i> | Routing table ID |
| <i>route_destid</i> | Destination ID to be routed |
| <i>route_port</i> | Port number to be routed |
| <i>lock</i> | apply a hardware lock on switch device flag (1=lock, 0=no_lock) |

Description

If available calls the switch specific `add_entry` method to add a route entry into a switch routing table. Otherwise uses standard RT update method as defined by RapidIO specification. A specific routing table can be selected using the *table* argument if a switch has per port routing tables or the standard (or global) table may be used by passing `RIO_GLOBAL_TABLE` in *table*.

Returns 0 on success or `-EINVAL` on failure.

Name

`rio_route_get_entry` — Read an entry from a switch routing table

Synopsis

```
int rio_route_get_entry (struct rio_dev * rdev, u16 table, u16
route_destid, u8 * route_port, int lock);
```

Arguments

| | |
|---------------------|---|
| <i>rdev</i> | RIO device |
| <i>table</i> | Routing table ID |
| <i>route_destid</i> | Destination ID to be routed |
| <i>route_port</i> | Pointer to read port number into |
| <i>lock</i> | apply a hardware lock on switch device flag (1=lock, 0=no_lock) |

Description

If available calls the switch specific `get_entry` method to fetch a route entry from a switch routing table. Otherwise uses standard RT read method as defined by RapidIO specification. A specific routing table can be selected using the *table* argument if a switch has per port routing tables or the standard (or global) table may be used by passing `RIO_GLOBAL_TABLE` in *table*.

Returns 0 on success or `-EINVAL` on failure.

Name

`rio_route_clr_table` — Clear a switch routing table

Synopsis

```
int rio_route_clr_table (struct rio_dev * rdev, u16 table, int lock);
```

Arguments

rdev RIO device

table Routing table ID

lock apply a hardware lock on switch device flag (1=lock, 0=no_lock)

Description

If available calls the switch specific `clr_table` method to clear a switch routing table. Otherwise uses standard RT write method as defined by RapidIO specification. A specific routing table can be selected using the *table* argument if a switch has per port routing tables or the standard (or global) table may be used by passing `RIO_GLOBAL_TABLE` in *table*.

Returns 0 on success or `-EINVAL` on failure.

Name

`rio_request_mport_dma` — request RapidIO capable DMA channel associated with specified local RapidIO mport device.

Synopsis

```
struct dma_chan * rio_request_mport_dma (struct rio_mport * mport);
```

Arguments

mport RIO mport to perform DMA data transfers

Description

Returns pointer to allocated DMA channel or NULL if failed.

Name

`rio_request_dma` — request RapidIO capable DMA channel that supports specified target RapidIO device.

Synopsis

```
struct dma_chan * rio_request_dma (struct rio_dev * rdev);
```

Arguments

rdev RIO device associated with DMA transfer

Description

Returns pointer to allocated DMA channel or NULL if failed.

Name

`rio_release_dma` — release specified DMA channel

Synopsis

```
void rio_release_dma (struct dma_chan * dchan);
```

Arguments

dchan DMA channel to release

Name

`rio_dma_prep_xfer` — RapidIO specific wrapper for `device_prep_slave_sg` callback defined by DMAENGINE.

Synopsis

```
struct dma_async_tx_descriptor * rio_dma_prep_xfer (struct dma_chan
* dchan, ul64 destid, struct rio_dma_data * data, enum
dma_transfer_direction direction, unsigned long flags);
```

Arguments

| | |
|------------------|---|
| <i>dchan</i> | DMA channel to configure |
| <i>destid</i> | target RapidIO device destination ID |
| <i>data</i> | RIO specific data descriptor |
| <i>direction</i> | DMA data transfer direction (TO or FROM the device) |
| <i>flags</i> | dmaengine defined flags |

Description

Initializes RapidIO capable DMA channel for the specified data transfer. Uses DMA channel private extension to pass information related to remote target RIO device. Returns pointer to DMA transaction descriptor or NULL if failed.

Name

`rio_dma_prep_slave_sg` — RapidIO specific wrapper for `device_prep_slave_sg` callback defined by DMAENGINE.

Synopsis

```
struct dma_async_tx_descriptor * rio_dma_prep_slave_sg (struct rio_dev
* rdev, struct dma_chan * dchan, struct rio_dma_data * data, enum
dma_transfer_direction direction, unsigned long flags);
```

Arguments

| | |
|------------------|---|
| <i>rdev</i> | RIO device control structure |
| <i>dchan</i> | DMA channel to configure |
| <i>data</i> | RIO specific data descriptor |
| <i>direction</i> | DMA data transfer direction (TO or FROM the device) |
| <i>flags</i> | dmaengine defined flags |

Description

Initializes RapidIO capable DMA channel for the specified data transfer. Uses DMA channel private extension to pass information related to remote target RIO device. Returns pointer to DMA transaction descriptor or NULL if failed.

Name

`rio_register_scan` — enumeration/discovery method registration interface

Synopsis

```
int rio_register_scan (int mport_id, struct rio_scan * scan_ops);
```

Arguments

mport_id mport device ID for which fabric scan routine has to be set (RIO_MPORT_ANY = set for all available mports)

scan_ops enumeration/discovery operations structure

Description

Registers enumeration/discovery operations with RapidIO subsystem and attaches it to the specified mport device (or all available mports if RIO_MPORT_ANY is specified).

Returns error if the mport already has an enumerator attached to it. In case of RIO_MPORT_ANY skips mports with valid scan routines (no error).

Name

`rio_unregister_scan` — removes enumeration/discovery method from mport

Synopsis

```
int rio_unregister_scan (int mport_id, struct rio_scan * scan_ops);
```

Arguments

mport_id mport device ID for which fabric scan routine has to be unregistered (RIO_MPORT_ANY = apply to all mports that use the specified scan_ops)

scan_ops enumeration/discovery operations structure

Description

Removes enumeration or discovery method assigned to the specified mport device. If RIO_MPORT_ANY is specified, removes the specified operations from all mports that have them attached.

Chapter 4. Internals

This chapter contains the autogenerated documentation of the RapidIO subsystem.

Structures

Name

struct rio_switch — RIO switch info

Synopsis

```
struct rio_switch {
    struct list_head node;
    u8 * route_table;
    u32 port_ok;
    struct rio_switch_ops * ops;
    spinlock_t lock;
    struct rio_dev * nextdev[0];
};
```

Members

| | |
|-------------|---|
| node | Node in global list of switches |
| route_table | Copy of switch routing table |
| port_ok | Status of each port (one bit per port) - OK=1 or UNINIT=0 |
| ops | pointer to switch-specific operations |
| lock | lock to serialize operations updates |
| nextdev[0] | Array of per-port pointers to the next attached device |

Name

struct rio_switch_ops — Per-switch operations

Synopsis

```
struct rio_switch_ops {
    struct module * owner;
    int (* add_entry) (struct rio_mport *mport, u16 destid, u8 hopcount, u16 table, u8 sw_domain);
    int (* get_entry) (struct rio_mport *mport, u16 destid, u8 hopcount, u16 table, u8 sw_domain);
    int (* clr_table) (struct rio_mport *mport, u16 destid, u8 hopcount, u16 table);
    int (* set_domain) (struct rio_mport *mport, u16 destid, u8 hopcount, u8 sw_domain);
    int (* get_domain) (struct rio_mport *mport, u16 destid, u8 hopcount, u8 *sw_domain);
    int (* em_init) (struct rio_dev *dev);
    int (* em_handle) (struct rio_dev *dev, u8 swport);
};
```

Members

| | |
|------------|--|
| owner | The module owner of this structure |
| add_entry | Callback for switch-specific route add function |
| get_entry | Callback for switch-specific route get function |
| clr_table | Callback for switch-specific clear route table function |
| set_domain | Callback for switch-specific domain setting function |
| get_domain | Callback for switch-specific domain get function |
| em_init | Callback for switch-specific error management init function |
| em_handle | Callback for switch-specific error management handler function |

Description

Defines the operations that are necessary to initialize/control a particular RIO switch device.

Name

struct rio_dev — RIO device info

Synopsis

```
struct rio_dev {
    struct list_head global_list;
    struct list_head net_list;
    struct rio_net * net;
    bool do_enum;
    u16 did;
    u16 vid;
    u32 device_rev;
    u16 asm_did;
    u16 asm_vid;
    u16 asm_rev;
    u16 efptr;
    u32 pef;
    u32 swpinfo;
    u32 src_ops;
    u32 dst_ops;
    u32 comp_tag;
    u32 phys_efptr;
    u32 em_efptr;
    u64 dma_mask;
    struct rio_driver * driver;
    struct device dev;
    struct resource riores[RIO_MAX_DEV_RESOURCES];
    int (* pwcback) (struct rio_dev *rdev, union rio_pw_msg *msg, int step);
    u16 destid;
    u8 hopcount;
    struct rio_dev * prev;
    atomic_t state;
    struct rio_switch rswitch[0];
};
```

Members

| | |
|-------------|--|
| global_list | Node in list of all RIO devices |
| net_list | Node in list of RIO devices in a network |
| net | Network this device is a part of |
| do_enum | Enumeration flag |
| did | Device ID |
| vid | Vendor ID |
| device_rev | Device revision |
| asm_did | Assembly device ID |

| | |
|-------------------------------|--|
| asm_vid | Assembly vendor ID |
| asm_rev | Assembly revision |
| efptr | Extended feature pointer |
| pef | Processing element features |
| swpinfo | Switch port info |
| src_ops | Source operation capabilities |
| dst_ops | Destination operation capabilities |
| comp_tag | RIO component tag |
| phys_efptr | RIO device extended features pointer |
| em_efptr | RIO Error Management features pointer |
| dma_mask | Mask of bits of RIO address this device implements |
| driver | Driver claiming this device |
| dev | Device model device |
| riores[RIO_MAX_DEV_RESOURCES] | RIO resources this device owns |
| pwcbk | port-write callback function for this device |
| destid | Network destination ID (or associated destid for switch) |
| hopcount | Hopcount to this device |
| prev | Previous RIO device connected to the current one |
| state | device state |
| rswitch[0] | struct rio_switch (if valid for this device) |

Name

struct rio_msg — RIO message event

Synopsis

```
struct rio_msg {  
    struct resource * res;  
    void (* mcback) (struct rio_mport * mport, void *dev_id, int mbox, int slot);  
};
```

Members

| | |
|--------|------------------------|
| res | Mailbox resource |
| mcback | Message event callback |

Name

struct rio_dbell — RIO doorbell event

Synopsis

```
struct rio_dbell {
    struct list_head node;
    struct resource * res;
    void (* dinb) (struct rio_mport *mport, void *dev_id, u16 src, u16 dst, u16 info);
    void * dev_id;
};
```

Members

| | |
|--------|--|
| node | Node in list of doorbell events |
| res | Doorbell resource |
| dinb | Doorbell event callback |
| dev_id | Device specific pointer to pass on event |

Name

struct rio_mport — RIO master port info

Synopsis

```
struct rio_mport {
    struct list_head dbells;
    struct list_head pwrites;
    struct list_head node;
    struct list_head nnode;
    struct rio_net * net;
    struct mutex lock;
    struct resource iores;
    struct resource riores[RIO_MAX_MPORT_RESOURCES];
    struct rio_msg inb_msg[RIO_MAX_MBOX];
    struct rio_msg outb_msg[RIO_MAX_MBOX];
    int host_deviceid;
    struct rio_ops * ops;
    unsigned char id;
    unsigned char index;
    unsigned int sys_size;
    enum rio_phy_type phy_type;
    u32 phys_efptr;
    unsigned char name[RIO_MAX_MPORT_NAME];
    struct device dev;
    void * priv;
#ifdef CONFIG_RAPIDIO_DMA_ENGINE
    struct dma_device dma;
#endif
    struct rio_scan * nscan;
    atomic_t state;
    unsigned int pwe_refcnt;
};
```

Members

| | |
|---------------------------------|---|
| dbells | List of doorbell events |
| pwrites | List of portwrite events |
| node | Node in global list of master ports |
| nnode | Node in network list of master ports |
| net | RIO net this mport is attached to |
| lock | lock to synchronize lists manipulations |
| iores | I/O mem resource that this master port interface owns |
| riores[RIO_MAX_MPORT_RESOURCES] | resources that this master port interfaces owns |
| inb_msg[RIO_MAX_MBOX] | RIO inbound message event descriptors |

| | |
|--------------------------|--|
| outb_msg[RIO_MAX_MBOX] | RIO outbound message event descriptors |
| host_deviceid | Host device ID associated with this master port |
| ops | configuration space functions |
| id | Port ID, unique among all ports |
| index | Port index, unique among all port interfaces of the same type |
| sys_size | RapidIO common transport system size |
| phy_type | RapidIO phy type |
| phys_efptr | RIO port extended features pointer |
| name[RIO_MAX_MPORT_NAME] | Port name string |
| dev | device structure associated with an mport |
| priv | Master port private data |
| dma | DMA device associated with mport |
| nscan | RapidIO network enumeration/discovery operations |
| state | mport device state |
| pwe_refcnt | port-write enable ref counter to track enable/disable requests |

Name

struct rio_net — RIO network info

Synopsis

```
struct rio_net {
    struct list_head node;
    struct list_head devices;
    struct list_head switches;
    struct list_head mports;
    struct rio_mport * hport;
    unsigned char id;
    struct device dev;
    void * enum_data;
    void (* release) (struct rio_net *net);
};
```

Members

| | |
|-----------|---|
| node | Node in global list of RIO networks |
| devices | List of devices in this network |
| switches | List of switches in this network |
| mports | List of master ports accessing this network |
| hport | Default port for accessing this network |
| id | RIO network ID |
| dev | Device object |
| enum_data | private data specific to a network enumerator |
| release | enumerator-specific release callback |

Name

struct rio_mport_attr — RIO mport device attributes

Synopsis

```
struct rio_mport_attr {  
    int flags;  
    int link_speed;  
    int link_width;  
    int dma_max_sge;  
    int dma_max_size;  
    int dma_align;  
};
```

Members

| | |
|--------------|--|
| flags | mport device capability flags |
| link_speed | SRIO link speed value (as defined by RapidIO specification) |
| link_width | SRIO link width value (as defined by RapidIO specification) |
| dma_max_sge | number of SG list entries that can be handled by DMA channel(s) |
| dma_max_size | max number of bytes in single DMA transfer (SG entry) |
| dma_align | alignment shift for DMA operations (as for other DMA operations) |

Name

struct rio_ops — Low-level RIO configuration space operations

Synopsis

```
struct rio_ops {
    int (* lcread) (struct rio_mport *mport, int index, u32 offset, int len, u32 *data);
    int (* lcwrite) (struct rio_mport *mport, int index, u32 offset, int len, u32 data);
    int (* cread) (struct rio_mport *mport, int index, u16 destid, u8 hopcount, u32 offset);
    int (* cwrite) (struct rio_mport *mport, int index, u16 destid, u8 hopcount, u32 offset, u32 data);
    int (* dsend) (struct rio_mport *mport, int index, u16 destid, u16 data);
    int (* pwenable) (struct rio_mport *mport, int enable);
    int (* open_outb_mbox) (struct rio_mport *mport, void *dev_id, int mbox, int entries);
    void (* close_outb_mbox) (struct rio_mport *mport, int mbox);
    int (* open_inb_mbox) (struct rio_mport *mport, void *dev_id, int mbox, int entries);
    void (* close_inb_mbox) (struct rio_mport *mport, int mbox);
    int (* add_outb_message) (struct rio_mport *mport, struct rio_dev *rdev, int mbox, u16 destid, u16 data);
    int (* add_inb_buffer) (struct rio_mport *mport, int mbox, void *buf);
    void (* get_inb_message) (struct rio_mport *mport, int mbox, void *buf);
    int (* map_inb) (struct rio_mport *mport, dma_addr_t lstart, u64 rstart, u32 size);
    void (* unmap_inb) (struct rio_mport *mport, dma_addr_t lstart);
    int (* query_mport) (struct rio_mport *mport, struct rio_mport_attr *attr);
    int (* map_outb) (struct rio_mport *mport, u16 destid, u64 rstart, u32 size, u32 offset);
    void (* unmap_outb) (struct rio_mport *mport, u16 destid, u64 rstart);
};
```

Members

| | |
|------------------|---|
| lcread | Callback to perform local (master port) read of config space. |
| lcwrite | Callback to perform local (master port) write of config space. |
| cread | Callback to perform network read of config space. |
| cwrite | Callback to perform network write of config space. |
| dsend | Callback to send a doorbell message. |
| pwenable | Callback to enable/disable port-write message handling. |
| open_outb_mbox | Callback to initialize outbound mailbox. |
| close_outb_mbox | Callback to shut down outbound mailbox. |
| open_inb_mbox | Callback to initialize inbound mailbox. |
| close_inb_mbox | Callback to shut down inbound mailbox. |
| add_outb_message | Callback to add a message to an outbound mailbox queue. |
| add_inb_buffer | Callback to add a buffer to an inbound mailbox queue. |
| get_inb_message | Callback to get a message from an inbound mailbox queue. |
| map_inb | Callback to map RapidIO address region into local memory space. |

| | |
|--------------------------|---|
| <code>unmap_inb</code> | Callback to unmap RapidIO address region mapped with <code>map_inb</code> . |
| <code>query_mport</code> | Callback to query mport device attributes. |
| <code>map_outb</code> | Callback to map outbound address region into local memory space. |
| <code>unmap_outb</code> | Callback to unmap outbound RapidIO address region. |

Name

struct rio_driver — RIO driver info

Synopsis

```
struct rio_driver {
    struct list_head node;
    char * name;
    const struct rio_device_id * id_table;
    int (* probe) (struct rio_dev * dev, const struct rio_device_id * id);
    void (* remove) (struct rio_dev * dev);
    void (* shutdown) (struct rio_dev * dev);
    int (* suspend) (struct rio_dev * dev, u32 state);
    int (* resume) (struct rio_dev * dev);
    int (* enable_wake) (struct rio_dev * dev, u32 state, int enable);
    struct device_driver driver;
};
```

Members

| | |
|-------------|--|
| node | Node in list of drivers |
| name | RIO driver name |
| id_table | RIO device ids to be associated with this driver |
| probe | RIO device inserted |
| remove | RIO device removed |
| shutdown | shutdown notification callback |
| suspend | RIO device suspended |
| resume | RIO device awakened |
| enable_wake | RIO device enable wake event |
| driver | LDM driver struct |

Description

Provides info on a RIO device driver for insertion/removal and power management purposes.

Name

struct rio_scan — RIO enumeration and discovery operations

Synopsis

```
struct rio_scan {  
    struct module * owner;  
    int (* enumerate) (struct rio_mport *mport, u32 flags);  
    int (* discover) (struct rio_mport *mport, u32 flags);  
};
```

Members

| | |
|-----------|---|
| owner | The module owner of this structure |
| enumerate | Callback to perform RapidIO fabric enumeration. |
| discover | Callback to perform RapidIO fabric discovery. |

Name

struct rio_scan_node — list node to register RapidIO enumeration and discovery methods with RapidIO core.

Synopsis

```
struct rio_scan_node {  
    int mport_id;  
    struct list_head node;  
    struct rio_scan * ops;  
};
```

Members

| | |
|----------|--|
| mport_id | ID of an mport (net) serviced by this enumerator |
| node | node in global list of registered enumerators |
| ops | RIO enumeration and discovery operations |

Enumeration and Discovery

Name

`rio_destid_alloc` — Allocate next available destID for given network

Synopsis

```
ul6 rio_destid_alloc (struct rio_net * net);
```

Arguments

net RIO network

Description

Returns next available device destination ID for the specified RIO network. Marks allocated ID as one in use. Returns `RIO_INVALID_DESTID` if new destID is not available.

Name

`rio_destid_reserve` — Reserve the specivied destID

Synopsis

```
int rio_destid_reserve (struct rio_net * net, u16 destid);
```

Arguments

net RIO network

destid destID to reserve

Description

Tries to reserve the specified destID. Returns 0 if successful.

Name

`rio_destid_free` — free a previously allocated destID

Synopsis

```
void rio_destid_free (struct rio_net * net, u16 destid);
```

Arguments

net RIO network

destid destID to free

Description

Makes the specified destID available for use.

Name

`rio_destid_first` — return first destID in use

Synopsis

```
u16 rio_destid_first (struct rio_net * net);
```

Arguments

net RIO network

Name

`rio_destid_next` — return next destID in use

Synopsis

```
u16 rio_destid_next (struct rio_net * net, u16 from);
```

Arguments

net RIO network

from destination ID from which search shall continue

Name

`rio_get_device_id` — Get the base/extended device id for a device

Synopsis

```
u16 rio_get_device_id (struct rio_mport * port, u16 destid, u8 hopcount);
```

Arguments

| | |
|-----------------|--------------------------|
| <i>port</i> | RIO master port |
| <i>destid</i> | Destination ID of device |
| <i>hopcount</i> | Hopcount to device |

Description

Reads the base/extended device id from a device. Returns the 8/16-bit device ID.

Name

`rio_set_device_id` — Set the base/extended device id for a device

Synopsis

```
void rio_set_device_id (struct rio_mport * port, u16 destid, u8 hopcount,  
u16 did);
```

Arguments

| | |
|-----------------|-------------------------------|
| <i>port</i> | RIO master port |
| <i>destid</i> | Destination ID of device |
| <i>hopcount</i> | Hopcount to device |
| <i>did</i> | Device ID value to be written |

Description

Writes the base/extended device id from a device.

Name

`rio_clear_locks` — Release all host locks and signal enumeration complete

Synopsis

```
int rio_clear_locks (struct rio_net * net);
```

Arguments

net RIO network to run on

Description

Marks the component tag CSR on each device with the enumeration complete flag. When complete, it then release the host locks on each device. Returns 0 on success or `-EINVAL` on failure.

Name

`rio_enum_host` — Set host lock and initialize host destination ID

Synopsis

```
int rio_enum_host (struct rio_mport * port);
```

Arguments

port Master port to issue transaction

Description

Sets the local host master port lock and destination ID register with the host device ID value. The host device ID value is provided by the platform. Returns 0 on success or -1 on failure.

Name

`rio_device_has_destid` — Test if a device contains a destination ID register

Synopsis

```
int rio_device_has_destid (struct rio_mport * port, int src_ops, int  
dst_ops);
```

Arguments

port Master port to issue transaction

src_ops RIO device source operations

dst_ops RIO device destination operations

Description

Checks the provided *src_ops* and *dst_ops* for the necessary transaction capabilities that indicate whether or not a device will implement a destination ID register. Returns 1 if true or 0 if false.

Name

`rio_release_dev` — Frees a RIO device struct

Synopsis

```
void rio_release_dev (struct device * dev);
```

Arguments

dev LDM device associated with a RIO device struct

Description

Gets the RIO device struct associated a RIO device struct. The RIO device struct is freed.

Name

`rio_is_switch` — Tests if a RIO device has switch capabilities

Synopsis

```
int rio_is_switch (struct rio_dev * rdev);
```

Arguments

rdev RIO device

Description

Gets the RIO device Processing Element Features register contents and tests for switch capabilities. Returns 1 if the device is a switch or 0 if it is not a switch. The RIO device struct is freed.

Name

`rio_setup_device` — Allocates and sets up a RIO device

Synopsis

```
struct rio_dev * rio_setup_device (struct rio_net * net, struct rio_mport  
* port, u16 destid, u8 hopcount, int do_enum);
```

Arguments

| | |
|-----------------|----------------------------------|
| <i>net</i> | RIO network |
| <i>port</i> | Master port to send transactions |
| <i>destid</i> | Current destination ID |
| <i>hopcount</i> | Current hopcount |
| <i>do_enum</i> | Enumeration/Discovery mode flag |

Description

Allocates a RIO device and configures fields based on configuration space contents. If device has a destination ID register, a destination ID is either assigned in enumeration mode or read from configuration space in discovery mode. If the device has switch capabilities, then a switch is allocated and configured appropriately. Returns a pointer to a RIO device on success or NULL on failure.

Name

`rio_sport_is_active` — Tests if a switch port has an active connection.

Synopsis

```
int rio_sport_is_active (struct rio_mport * port, u16 destid, u8  
hopcount, int sport);
```

Arguments

| | |
|-----------------|--------------------------------------|
| <i>port</i> | Master port to send transaction |
| <i>destid</i> | Associated destination ID for switch |
| <i>hopcount</i> | Hopcount to reach switch |
| <i>sport</i> | Switch port number |

Description

Reads the port error status CSR for a particular switch port to determine if the port has an active link. Returns `RIO_PORT_N_ERR_STS_PORT_OK` if the port is active or 0 if it is inactive.

Name

`rio_get_host_deviceid_lock` — Reads the Host Device ID Lock CSR on a device

Synopsis

```
u16 rio_get_host_deviceid_lock (struct rio_mport * port, u8 hopcount);
```

Arguments

port Master port to send transaction

hopcount Number of hops to the device

Description

Used during enumeration to read the Host Device ID Lock CSR on a RIO device. Returns the value of the lock register.

Name

`rio_enum_peer` — Recursively enumerate a RIO network through a master port

Synopsis

```
int rio_enum_peer (struct rio_net * net, struct rio_mport * port, u8  
hopcount, struct rio_dev * prev, int prev_port);
```

Arguments

| | |
|------------------|---|
| <i>net</i> | RIO network being enumerated |
| <i>port</i> | Master port to send transactions |
| <i>hopcount</i> | Number of hops into the network |
| <i>prev</i> | Previous RIO device connected to the enumerated one |
| <i>prev_port</i> | Port on previous RIO device |

Description

Recursively enumerates a RIO network. Transactions are sent via the master port passed in *port*.

Name

`rio_enum_complete` — Tests if enumeration of a network is complete

Synopsis

```
int rio_enum_complete (struct rio_mport * port);
```

Arguments

port Master port to send transaction

Description

Tests the PGCCSR discovered bit for non-zero value (enumeration complete flag). Return 1 if enumeration is complete or 0 if enumeration is incomplete.

Name

`rio_disc_peer` — Recursively discovers a RIO network through a master port

Synopsis

```
int rio_disc_peer (struct rio_net * net, struct rio_mport * port, u16
destid, u8 hopcount, struct rio_dev * prev, int prev_port);
```

Arguments

| | |
|------------------|-----------------------------------|
| <i>net</i> | RIO network being discovered |
| <i>port</i> | Master port to send transactions |
| <i>destid</i> | Current destination ID in network |
| <i>hopcount</i> | Number of hops into the network |
| <i>prev</i> | previous <code>rio_dev</code> |
| <i>prev_port</i> | previous port number |

Description

Recursively discovers a RIO network. Transactions are sent via the master port passed in *port*.

Name

`rio_mport_is_active` — Tests if master port link is active

Synopsis

```
int rio_mport_is_active (struct rio_mport * port);
```

Arguments

port Master port to test

Description

Reads the port error status CSR for the master port to determine if the port has an active link. Returns `RIO_PORT_N_ERR_STS_PORT_OK` if the master port is active or 0 if it is inactive.

Name

`rio_update_route_tables` — Updates route tables in switches

Synopsis

```
void rio_update_route_tables (struct rio_net * net);
```

Arguments

net RIO network to run update on

Description

For each enumerated device, ensure that each switch in a system has correct routing entries. Add routes for devices that were unknown during the first enumeration pass through the switch.

Name

`rio_init_em` — Initializes RIO Error Management (for switches)

Synopsis

```
void rio_init_em (struct rio_dev * rdev);
```

Arguments

rdev RIO device

Description

For each enumerated switch, call device-specific error management initialization routine (if supplied by the switch driver).

Name

`rio_enum_mport` — Start enumeration through a master port

Synopsis

```
int rio_enum_mport (struct rio_mport * mport, u32 flags);
```

Arguments

mport Master port to send transactions

flags Enumeration control flags

Description

Starts the enumeration process. If somebody has enumerated our master port device, then give up. If not and we have an active link, then start recursive peer enumeration. Returns 0 if enumeration succeeds or `-EBUSY` if enumeration fails.

Name

`rio_build_route_tables` — Generate route tables from switch route entries

Synopsis

```
void rio_build_route_tables (struct rio_net * net);
```

Arguments

net RIO network to run route tables scan on

Description

For each switch device, generate a route table by copying existing route entries from the switch.

Name

`rio_disc_mport` — Start discovery through a master port

Synopsis

```
int rio_disc_mport (struct rio_mport * mport, u32 flags);
```

Arguments

mport Master port to send transactions

flags discovery control flags

Description

Starts the discovery process. If we have an active link, then wait for the signal that enumeration is complete (if wait is allowed). When enumeration completion is signaled, start recursive peer discovery. Returns 0 if discovery succeeds or `-EBUSY` on failure.

Name

`rio_basic_attach` —

Synopsis

```
int rio_basic_attach ( void );
```

Arguments

void no arguments

Description

When this enumeration/discovery method is loaded as a module this function registers its specific enumeration and discover routines for all available RapidIO mport devices. The “scan” command line parameter controls ability of the module to start RapidIO enumeration/discovery automatically.

Returns 0 for success or -EIO if unable to register itself.

This enumeration/discovery method cannot be unloaded and therefore does not provide a matching `cleanup_module` routine.

Driver functionality

Name

`rio_setup_inb_dbell` — bind inbound doorbell callback

Synopsis

```
int rio_setup_inb_dbell (struct rio_mport * mport, void * dev_id, struct
resource * res, void (*dinb) (struct rio_mport * mport, void *dev_id,
ul6 src, ul6 dst, ul6 info));
```

Arguments

mport RIO master port to bind the doorbell callback

dev_id Device specific pointer to pass on event

res Doorbell message resource

dinb Callback to execute when doorbell is received

Description

Adds a doorbell resource/callback pair into a port's doorbell event list. Returns 0 if the request has been satisfied.

Name

`rio_chk_dev_route` — Validate route to the specified device.

Synopsis

```
int rio_chk_dev_route (struct rio_dev * rdev, struct rio_dev ** nrdev,  
int * npnum);
```

Arguments

rdev RIO device failed to respond

nrdev Last active device on the route to *rdev*

npnum *nrdev*'s port number on the route to *rdev*

Description

Follows a route to the specified RIO device to determine the last available device (and corresponding RIO port) on the route.

Name

`rio_chk_dev_access` — Validate access to the specified device.

Synopsis

```
int rio_chk_dev_access (struct rio_dev * rdev);
```

Arguments

rdev Pointer to RIO device control structure

Name

`rio_get_input_status` — Sends a Link-Request/Input-Status control symbol and returns link-response (if requested).

Synopsis

```
int rio_get_input_status (struct rio_dev * rdev, int pnum, u32 *  
lnkresp);
```

Arguments

rdev RIO devive to issue Input-status command

pnum Device port number to issue the command

lnkresp Response from a link partner

Name

`rio_clr_err_stopped` — Clears port Error-stopped states.

Synopsis

```
int  rio_clr_err_stopped (struct  rio_dev  *  rdev,  u32  pnum,  u32
err_status);
```

Arguments

| | |
|-------------------|---|
| <i>rdev</i> | Pointer to RIO device control structure |
| <i>pnum</i> | Switch port number to clear errors |
| <i>err_status</i> | port error status (if 0 reads register from device) |

Name

`rio_std_route_add_entry` — Add switch route table entry using standard registers defined in RIO specification rev.1.3

Synopsis

```
int rio_std_route_add_entry (struct rio_mport * mport, u16 destid, u8  
hopcount, u16 table, u16 route_destid, u8 route_port);
```

Arguments

| | |
|---------------------|--|
| <i>mport</i> | Master port to issue transaction |
| <i>destid</i> | Destination ID of the device |
| <i>hopcount</i> | Number of switch hops to the device |
| <i>table</i> | routing table ID (global or port-specific) |
| <i>route_destid</i> | destID entry in the RT |
| <i>route_port</i> | destination port for specified destID |

Name

`rio_std_route_get_entry` — Read switch route table entry (port number) associated with specified destID using standard registers defined in RIO specification rev.1.3

Synopsis

```
int rio_std_route_get_entry (struct rio_mport * mport, u16 destid, u8  
hopcount, u16 table, u16 route_destid, u8 * route_port);
```

Arguments

| | |
|---------------------|--|
| <i>mport</i> | Master port to issue transaction |
| <i>destid</i> | Destination ID of the device |
| <i>hopcount</i> | Number of switch hops to the device |
| <i>table</i> | routing table ID (global or port-specific) |
| <i>route_destid</i> | destID entry in the RT |
| <i>route_port</i> | returned destination port for specified destID |

Name

`rio_std_route_clr_table` — Clear switch route table using standard registers defined in RIO specification rev.1.3.

Synopsis

```
int rio_std_route_clr_table (struct rio_mport * mport, u16 destid, u8  
hopcount, u16 table);
```

Arguments

| | |
|-----------------|--|
| <i>mport</i> | Master port to issue transaction |
| <i>destid</i> | Destination ID of the device |
| <i>hopcount</i> | Number of switch hops to the device |
| <i>table</i> | routing table ID (global or port-specific) |

Name

`rio_find_mport` — find RIO mport by its ID

Synopsis

```
struct rio_mport * rio_find_mport (int mport_id);
```

Arguments

mport_id number (ID) of mport device

Description

Given a RIO mport number, the desired mport is located in the global list of mports. If the mport is found, a pointer to its data structure is returned. If no mport is found, `NULL` is returned.

Name

`rio_mport_scan` — execute enumeration/discovery on the specified mport

Synopsis

```
int rio_mport_scan (int mport_id);
```

Arguments

mport_id number (ID) of mport device

Name

RIO_LOP_READ — Generate `rio_local_read_config_*` functions

Synopsis

```
RIO_LOP_READ ( size, type, len );
```

Arguments

size Size of configuration space read (8, 16, 32 bits)

type C type of value argument

len Length of configuration space read (1, 2, 4 bytes)

Description

Generates `rio_local_read_config_*` functions used to access configuration space registers on the local device.

Name

RIO_LOP_WRITE — Generate rio_local_write_config_* functions

Synopsis

```
RIO_LOP_WRITE ( size, type, len);
```

Arguments

size Size of configuration space write (8, 16, 32 bits)

type C type of value argument

len Length of configuration space write (1, 2, 4 bytes)

Description

Generates rio_local_write_config_* functions used to access configuration space registers on the local device.

Name

RIO_OP_READ — Generate rio_mport_read_config_* functions

Synopsis

```
RIO_OP_READ ( size, type, len );
```

Arguments

size Size of configuration space read (8, 16, 32 bits)

type C type of value argument

len Length of configuration space read (1, 2, 4 bytes)

Description

Generates rio_mport_read_config_* functions used to access configuration space registers on the local device.

Name

RIO_OP_WRITE — Generate rio_mport_write_config_* functions

Synopsis

```
RIO_OP_WRITE ( size, type, len );
```

Arguments

size Size of configuration space write (8, 16, 32 bits)

type C type of value argument

len Length of configuration space write (1, 2, 4 bytes)

Description

Generates rio_mport_write_config_* functions used to access configuration space registers on the local device.

Device model support

Name

`rio_match_device` — Tell if a RIO device has a matching RIO device id structure

Synopsis

```
const struct rio_device_id * rio_match_device (const struct
rio_device_id * id, const struct rio_dev * rdev);
```

Arguments

id the RIO device id structure to match against

rdev the RIO device structure to match against

Description

Used from driver probe and bus matching to check whether a RIO device matches a device id structure provided by a RIO driver. Returns the matching struct `rio_device_id` or `NULL` if there is no match.

Name

`rio_device_probe` — Tell if a RIO device structure has a matching RIO device id structure

Synopsis

```
int rio_device_probe (struct device * dev);
```

Arguments

dev the RIO device structure to match against

Description

return 0 and set `rio_dev->driver` when `drv` claims `rio_dev`, else error

Name

`rio_device_remove` — Remove a RIO device from the system

Synopsis

```
int rio_device_remove (struct device * dev);
```

Arguments

dev the RIO device structure to match against

Description

Remove a RIO device from the system. If it has an associated driver, then run the driver `remove` method. Then update the reference count.

Name

`rio_match_bus` — Tell if a RIO device structure has a matching RIO driver device id structure

Synopsis

```
int rio_match_bus (struct device * dev, struct device_driver * drv);
```

Arguments

dev the standard device structure to match against

drv the standard driver structure containing the ids to match against

Description

Used by a driver to check whether a RIO device present in the system is in its list of supported devices. Returns 1 if there is a matching struct `rio_device_id` or 0 if there is no match.

Name

`rio_bus_init` — Register the RapidIO bus with the device model

Synopsis

```
int rio_bus_init ( void );
```

Arguments

void no arguments

Description

Registers the RIO mport device class and RIO bus type with the Linux device model.

Sysfs support

Name

`rio_create_sysfs_dev_files` — create RIO specific sysfs files

Synopsis

```
int rio_create_sysfs_dev_files (struct rio_dev * rdev);
```

Arguments

rdev device whose entries should be created

Description

Create files when *rdev* is added to sysfs.

Name

`rio_remove_sysfs_dev_files` — cleanup RIO specific sysfs files

Synopsis

```
void rio_remove_sysfs_dev_files (struct rio_dev * rdev);
```

Arguments

rdev device whose entries we should free

Description

Cleanup when *rdev* is removed from sysfs.

PPC32 support

Name

`fsl_local_config_read` — Generate a MPC85xx local config space read

Synopsis

```
int fsl_local_config_read (struct rio_mport * mport, int index, u32  
offset, int len, u32 * data);
```

Arguments

| | |
|---------------|--|
| <i>mport</i> | RapidIO master port info |
| <i>index</i> | ID of RapdiIO interface |
| <i>offset</i> | Offset into configuration space |
| <i>len</i> | Length (in bytes) of the maintenance transaction |
| <i>data</i> | Value to be read into |

Description

Generates a MPC85xx local configuration space read. Returns 0 on success or `-EINVAL` on failure.

Name

`fsl_local_config_write` — Generate a MPC85xx local config space write

Synopsis

```
int fsl_local_config_write (struct rio_mport * mport, int index, u32  
offset, int len, u32 data);
```

Arguments

| | |
|---------------|--|
| <i>mport</i> | RapidIO master port info |
| <i>index</i> | ID of RapdiIO interface |
| <i>offset</i> | Offset into configuration space |
| <i>len</i> | Length (in bytes) of the maintenance transaction |
| <i>data</i> | Value to be written |

Description

Generates a MPC85xx local configuration space write. Returns 0 on success or `-EINVAL` on failure.

Name

`fsl_rio_config_read` — Generate a MPC85xx read maintenance transaction

Synopsis

```
int fsl_rio_config_read (struct rio_mport * mport, int index, u16 destid,  
u8 hopcount, u32 offset, int len, u32 * val);
```

Arguments

| | |
|-----------------|--|
| <i>mport</i> | RapidIO master port info |
| <i>index</i> | ID of RapdiIO interface |
| <i>destid</i> | Destination ID of transaction |
| <i>hopcount</i> | Number of hops to target device |
| <i>offset</i> | Offset into configuration space |
| <i>len</i> | Length (in bytes) of the maintenance transaction |
| <i>val</i> | Location to be read into |

Description

Generates a MPC85xx read maintenance transaction. Returns 0 on success or `-EINVAL` on failure.

Name

`fsl_rio_config_write` — Generate a MPC85xx write maintenance transaction

Synopsis

```
int fsl_rio_config_write (struct rio_mport * mport, int index, u16  
destid, u8 hopcount, u32 offset, int len, u32 val);
```

Arguments

| | |
|-----------------|--|
| <i>mport</i> | RapidIO master port info |
| <i>index</i> | ID of RapdiIO interface |
| <i>destid</i> | Destination ID of transaction |
| <i>hopcount</i> | Number of hops to target device |
| <i>offset</i> | Offset into configuration space |
| <i>len</i> | Length (in bytes) of the maintenance transaction |
| <i>val</i> | Value to be written |

Description

Generates an MPC85xx write maintenance transaction. Returns 0 on success or `-EINVAL` on failure.

Name

`fsl_rio_setup` — Setup Freescale PowerPC RapidIO interface

Synopsis

```
int fsl_rio_setup (struct platform_device * dev);
```

Arguments

dev platform_device pointer

Description

Initializes MPC85xx RapidIO hardware interface, configures master port with system-specific info, and registers the master port with the RapidIO subsystem.

Chapter 5. Credits

The following people have contributed to the RapidIO subsystem directly or indirectly:

1. Matt Porter<mporter@kernel.crashing.org>
2. Randy Vinson<rvinson@mvista.com>
3. Dan Malek<dan@embeddedalley.com>

The following people have contributed to this document:

1. Matt Porter<mporter@kernel.crashing.org>