

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	66
Structures	66
Enumeration and Discovery	79
Driver functionality	107
Device model support	121
Sysfs support	126
PPC32 support	128
5. Credits	134

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

```
ul6 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_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_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_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_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_release_inb_pwrite` — release inbound port-write message service

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_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_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_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_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` — process inbound port-write message

Synopsis

```
int rio_inb_pwrite_handler (union rio_pw_msg * pw_msg);
```

Arguments

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_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_release_dma` — release specified DMA channel

Synopsis

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

Arguments

dchan DMA channel to release

Chapter 4. Internals

This chapter contains the autogenerated documentation of the RapidIO subsystem.

Structures

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_DE- V_RESOURCES]	RIO resources this device owns
pwcback	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
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

sys_size	RapidIO common transport system size
phy_type	RapidIO phy type
phys_efptr	RIO port extended features pointer
name[RIO_MAX_M- PORT_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

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_first` — return first destID in use

Synopsis

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

Arguments

net RIO network

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_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_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_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_alloc_net` — Allocate and configure a new RIO network

Synopsis

```
struct rio_net * rio_alloc_net (struct rio_mport * port, int do_enum,  
                                ul6 start);
```

Arguments

port Master port associated with the RIO network

do_enum Enumeration/Discovery mode flag

start logical minimal start id for new net

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

Synopsis

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

Arguments

port Master port associated with port-write handling

enable 1=enable, 0=disable

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

<i>mport</i>	RIO master port to bind the doorbell callback
<i>dev_id</i>	Device specific pointer to pass on event
<i>res</i>	Doorbell message resource
<i>dinb</i>	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 device 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_s-  
tatus);
```

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>