Introduction

The RapidIO standard is a packet-based fabric interconnect standard designed for use in embedded systems. Development of the RapidIO standard is directed by the RapidIO Trade Association (RTA). The current version of the RapidIO specification is publicly available for download from the RTA web-site [1].

This document describes the basics of the Linux RapidIO subsystem and provides information on its major components.

1 Overview

Because the RapidIO subsystem follows the Linux device model it is integrated into the kernel similarly to other buses by defining RapidIO-specific device and bus types and registering them within the device model.

The Linux RapidIO subsystem is architecture independent and therefore defines architecture-specific interfaces that provide support for common RapidIO subsystem operations.

2. Core Components

A typical RapidIO network is a combination of endpoints and switches. Each of these components is represented in the subsystem by an associated data structure. The core logical components of the RapidIO subsystem are defined in include/linux/rio.h file.

2.1 Master Port

A master port (or mport) is a RapidIO interface controller that is local to the processor executing the Linux code. A master port generates and receives RapidIO packets (transactions). In the RapidIO subsystem each master port is represented by a rio_mport data structure. This structure contains master port specific resources such as mailboxes and doorbells. The rio_mport also includes a unique host device ID that is valid when a master port is configured as an enumerating host.

RapidIO master ports are serviced by subsystem specific mport device drivers that provide functionality defined for this subsystem. To provide a hardware independent interface for RapidIO subsystem operations, rio_mport structure includes rio_ops data structure which contains pointers to hardware specific implementations of RapidIO functions.

2.2 Device

A RapidIO device is any endpoint (other than mport) or switch in the network. All devices are presented in the RapidIO subsystem by corresponding rio_dev data structure. Devices form one global device list and per-network device lists (depending on number of available mports and networks).

2.3 Switch

A RapidIO switch is a special class of device that routes packets between its ports towards their final destination. The packet destination port within a switch is defined by an internal routing table. A switch is presented in the RapidIO subsystem by rio_dev data structure expanded by additional rio_switch data structure, which contains switch specific information such as copy of the routing table and pointers to switch specific functions.

The RapidIO subsystem defines the format and initialization method for subsystem specific switch drivers that are designed to provide hardware-specific implementation of common switch management routines.

2.4 Network

A RapidIO network is a combination of interconnected endpoint and switch devices. Each RapidIO network known to the system is represented by corresponding rio_net data structure. This structure includes lists of all devices and local master ports that form the same network. It also contains a pointer to the default master port that is used to communicate with devices within the network.

2.5 Device Drivers

RapidIO device-specific drivers follow Linux Kernel Driver Model and are intended to support specific RapidIO devices attached to the RapidIO network.

2.6 Subsystem Interfaces

RapidIO interconnect specification defines features that may be used to provide one or more common service layers for all participating RapidIO devices. These common services may act separately from device-specific drivers or be used by device-specific drivers. Example of such service provider is the RIONET driver which implements Ethernet-over-RapidIO interface. Because only one driver can be registered for a device, all common RapidIO services have to be registered as subsystem interfaces. This allows to have multiple common services attached to the same device without blocking attachment of a device-specific driver.

3. Subsystem Initialization

In order to initialize the RapidIO subsystem, a platform must initialize and register at least one master port within the RapidIO network. To register mport within the subsystem controller driver's initialization code calls function rio_register_mport() for each available master port.

After all active master ports are registered with a RapidIO subsystem, an enumeration and/or discovery routine may be called automatically or by user-space command.

RapidIO subsystem can be configured to be built as a statically linked or modular component of the kernel (see details below).

4. Enumeration and Discovery

4.1 Overview

RapidIO subsystem configuration options allow users to build enumeration and discovery methods as statically linked components or loadable modules. An enumeration/discovery method implementation and available input parameters define how any given method can be attached to available RapidIO mports: simply to all available mports OR individually to the specified mport device.

Depending on selected enumeration/discovery build configuration, there are several methods to initiate an enumeration and/or discovery process:

(a) Statically linked enumeration and discovery process can be started automatically during kernel initialization time using corresponding module parameters. This was the original method used since introduction of RapidIO subsystem. Now this method relies on enumerator module parameter which is 'rio-scan.scan' for existing basic enumeration/discovery method. When automatic start of enumeration/discovery is used a user has to ensure that all discovering endpoints are started before the enumerating endpoint and are waiting for enumeration to be completed. Configuration option CONFIG_RAPIDIO_DISC_TIMEOUT defines time that discovering endpoint waits for enumeration to be completed. If the specified timeout expires the discovery process is terminated without obtaining RapidIO network information. NOTE: a timed out discovery process may be restarted later using a user-space command as it is described below (if the given endpoint was enumerated successfully).

(b) Statically linked enumeration and discovery process can be started by a command from user space. This initiation method provides more flexibility for a system startup compared to the option (a) above. After all participating endpoints have been successfully booted, an enumeration process shall be started first by issuing a user-space command, after an enumeration is completed a discovery process can be started on all remaining endpoints.

(c) Modular enumeration and discovery process can be started by a command from user space. After an enumeration/discovery module is loaded, a network scan process can be started by issuing a user-space command. Similar to the option (b) above, an enumerator has to be started first.

(d) Modular enumeration and discovery process can be started by a module initialization routine. In this case an enumerating module shall be loaded first.

When a network scan process is started it calls an enumeration or discovery routine depending on the configured role of a master port: host or agent.

Enumeration is performed by a master port if it is configured as a host port by assigning a host destination ID greater than or equal to zero. The host destination ID can be assigned to a master port using various methods depending on RapidIO subsystem build configuration:

(a) For a statically linked RapidIO subsystem core use command line parameter "rapidio.hdid=" with a list of destination ID assignments in order of mport device registration. For example, in a system with two RapidIO controllers the command line parameter "rapidio.hdid=-1,7" will result in assignment of the host destination ID=7 to the second RapidIO controller, while the first one will be assigned destination ID=-1.

(b) If the RapidIO subsystem core is built as a loadable module, in addition to the method shown above, the host destination ID(s) can be specified using traditional methods of passing module parameter "hdid=" during its loading:

- from command line: "modprobe rapidio hdid=-1,7", or
- from modprobe configuration file using configuration command "options", like in this example: "options rapidio hdid=-1,7". An example of modprobe configuration file is provided in the section below.

NOTES:

(i) if "hdid=" parameter is omitted all available mport will be assigned destination ID = -1;

(ii) the "hdid=" parameter in systems with multiple mports can have destination ID assignments omitted from the end of list (default = -1).

If the host device ID for a specific master port is set to -1, the discovery process will be performed for it.

The enumeration and discovery routines use RapidIO maintenance transactions to access the configuration space of devices.

NOTE: If RapidIO switch-specific device drivers are built as loadable modules they must be loaded before enumeration/discovery

process starts. This requirement is cased by the fact that enumeration/discovery methods invoke vendor-specific callbacks on early stages.

4.2 Automatic Start of Enumeration and Discovery

Automatic enumeration/discovery start method is applicable only to built-in enumeration/discovery RapidIO configuration selection. To enable automatic enumeration/discovery start by existing basic enumerator method set use boot command line parameter "rio-scan.scan=1".

This configuration requires synchronized start of all RapidIO endpoints that form a network which will be enumerated/discovered. Discovering endpoints have to be started before an enumeration starts to ensure that all RapidIO controllers have been initialized and are ready to be discovered. Configuration parameter CONFIG_RAPIDIO_DISC_TIMEOUT defines time (in seconds) which a discovering endpoint will wait for enumeration to be completed.

When automatic enumeration/discovery start is selected, basic method's initialization routine calls rio_init_mports() to perform enumeration or discovery for all known mport devices.

Depending on RapidIO network size and configuration this automatic enumeration/discovery start method may be difficult to use due to the requirement for synchronized start of all endpoints.

4.3 User-space Start of Enumeration and Discovery

User-space start of enumeration and discovery can be used with built-in and modular build configurations. For user-space controlled start RapidIO subsystem creates the sysfs write-only attribute file '/sys/bus/rapidio/scan'. To initiate an enumeration or discovery process on specific mport device, a user needs to write mport_ID (not RapidIO destination ID) into that file. The mport_ID is a sequential number (0 ... RIO_MAX_MPORTS) assigned during mport device registration. For example for machine with single RapidIO controller, mport_ID for that controller always will be 0.

To initiate RapidIO enumeration/discovery on all available mports a user may write '-1' (or RIO_MPORT_ANY) into the scan attribute file.

4.4 Basic Enumeration Method

This is an original enumeration/discovery method which is available since first release of RapidIO subsystem code. The enumeration process is implemented according to the enumeration algorithm outlined in the RapidIO Interconnect Specification: Annex I [1].

This method can be configured as statically linked or loadable module. The method's single parameter "scan" allows to trigger the enumeration/discovery process from module initialization routine.

This enumeration/discovery method can be started only once and does not support unloading if it is built as a module.

The enumeration process traverses the network using a recursive depth-first algorithm. When a new device is found, the enumerator takes ownership of that device by writing into the Host Device ID Lock CSR. It does this to ensure that the enumerator has exclusive right to enumerate the device. If device ownership is successfully acquired, the enumerator allocates a new rio_dev structure and initializes it according to device capabilities.

If the device is an endpoint, a unique device ID is assigned to it and its value is written into the device's Base Device ID CSR.

If the device is a switch, the enumerator allocates an additional rio_switch structure to store switch specific information. Then the switch's vendor ID and device ID are queried against a table of known RapidIO switches. Each switch table entry contains a pointer to a switch-specific initialization routine that initializes pointers to the rest of switch specific operations, and performs hardware initialization if necessary. A RapidIO switch does not have a unique device ID; it relies on hopcount and routing for device ID of an attached endpoint if access to its configuration registers is required. If a switch (or chain of switches) does not have any endpoint (except enumerator) attached to it, a fake device ID will be assigned to configure a route to that switch. In the case of a chain of switches without endpoint, one fake device ID is used to configure a route through the entire chain and switches are differentiated by their hopcount value.

For both endpoints and switches the enumerator writes a unique component tag into device's Component Tag CSR. That unique value is used by the error management notification mechanism to identify a device that is reporting an error management event.

Enumeration beyond a switch is completed by iterating over each active egress port of that switch. For each active link, a route to a default device ID (0xFF for 8-bit systems and 0xFFFF for 16-bit systems) is temporarily written into the routing table. The algorithm recurs by calling itself with hopcount + 1 and the default device ID in order to access the device on the active port.

After the host has completed enumeration of the entire network it releases devices by clearing device ID locks (calls rio_clear_locks()). For each endpoint in the system, it sets the Discovered bit in the Port General Control CSR to indicate that enumeration is completed and agents are allowed to execute passive discovery of the network.

The discovery process is performed by agents and is similar to the enumeration process that is described above. However, the discovery process is performed without changes to the existing routing because agents only gather information about RapidIO network structure and are building an internal map of discovered devices. This way each Linux-based component of the RapidIO subsystem has a complete view of the network. The discovery process can be performed simultaneously by several agents. After initializing its RapidIO master port each agent waits for enumeration completion by the host for the configured wait time period. If this wait time period expires before enumeration is completed, an agent skips RapidIO discovery and continues with remaining kernel

initialization.

4.5 Adding New Enumeration/Discovery Method

RapidIO subsystem code organization allows addition of new enumeration/discovery methods as new configuration options without significant impact to the core RapidIO code.

A new enumeration/discovery method has to be attached to one or more mport devices before an enumeration/discovery process can be started. Normally, method's module initialization routine calls rio_register_scan() to attach an enumerator to a specified mport device (or devices). The basic enumerator implementation demonstrates this process.

4.6 Using Loadable RapidIO Switch Drivers

In the case when RapidIO switch drivers are built as loadable modules a user must ensure that they are loaded before the enumeration/discovery starts. This process can be automated by specifying pre- or post- dependencies in the RapidIO-specific modprobe configuration file as shown in the example below.

File /etc/modprobe.d/rapidio.conf:

```
# Configure RapidIO subsystem modules

# Set enumerator host destination ID (overrides kernel command line option)
options rapidio hdid=-1,2

# Load RapidIO switch drivers immediately after rapidio core module was loaded
softdep rapidio post: idt_gen2 idtcps tsi57x

# OR :

# Load RapidIO switch drivers just before rio-scan enumerator module is loaded
softdep rio-scan pre: idt_gen2 idtcps tsi57x

--------------------------
```

NOTE:

In the example above, one of "softdep" commands must be removed or commented out to keep required module loading sequence.

5. References

[1] RapidIO Trade Association. RapidIO Interconnect Specifications.
http://www.rapidio.org.

[2] Rapidio TA. Technology Comparisons.
http://www.rapidio.org/education/technology_comparisons/

[3] RapidIO support for Linux.
https://lwn.net/Articles/139118/

[4] Matt Porter. RapidIO for Linux. Ottawa Linux Symposium, 2005
https://www.kernel.org/doc/ols/2005/ols2005v2-pages-43-56.pdf