Software considerations around RapidIO designs

Jim Parisien - January 24, 2008

In general, the topic of software is rather vast. It ranges from low-level drivers, RTOSs, through to applications. However, there are some unique software considerations that are specific to RapidIO. This article will provide a level of understanding of some of the basic information needed to understand how to succeed with board and system level bring up. Also discussed are the built-in high availability features in RapidIO that can help with debugging both software and hardware in a RapidIO system.

What is Discovery and Enumeration and why is it required? Discovery in RapidIO is required to explore a network without knowing its configuration, and assign each "element" — a unique device identifier. In PCI, a device is uniquely identified by its "device number, bus number and function number." In RapidIO we simply use Destination ID.

RapidIO, unlike PCI, can support any system topology as well - Tree, Star, ring and full mesh; the discovery process involves identifying all of the Processing Elements (PEs) in a network, understanding the system topology, and enumerating each endpoint with a unique Destination ID. The Destination ID register is one of the fundamental maintenance registers within any endpoint, along with a number of others discussed below.

Does one really need to perform discovery in a RapidIO system? The short answer is no. System discovery is only used in systems where the configuration is not known ahead of time by software - such as a system where a user can put any set of RapidIO-based blades into a Chassis. However in a static system where PEs are always connected in the same manner through the same switches - such as on a board, with no mezzanine cards - software can initialize switches and endpoints with a known configuration. Many PEs support power up configuration options where routing tables, Destination ID, and other parameters are pre-set from non-volatile memory such as I2C EPROMs, so software is not required.

System discovery is the process of determining what types of PEs are in the system, how they are interconnected, and configuring them for operation. What does that really mean? Let's walk through it one step at a time and look at Discovery, Enumeration, and Initialization separately.

**Discovery**

In its simplest form Discovery is the process of interrogating each switch and endpoint in a system. From this, determine the specific capabilities of each PE, how they are interconnected, and ultimately get an understanding of the network topology itself. If the first PE has another endpoint, then the discovery process is completed. However if the first PE is a switch, then more exploration is
Assuming you wish to discover a RapidIO network from a processor somewhere within a network, how do you interrogate the first PE? To answer, the concept of Maintenance Transactions and understanding how they are used needs to be introduced.

RapidIO uses a Destination-based routing methodology as opposed to a path-based methodology. Therefore, virtually all packets contain a Destination ID in its header. For standard I/O transactions this represents the endpoint to reach. For a Maintenance Transaction it represents more of a direction along the path to a specific endpoint. For Maintenance Transactions, a second parameter called a Hop Count is decremented by each switch along the way and used to ultimately specify the final destination of the packet. Further, routing tables in switches are nothing more than a switch port number lookup table that use Destination ID as an index.

Before discovery, an unknown endpoint is an endpoint that is required to respond to any maintenance transaction using a Destination ID of Hex FF – in a small transport system - following power up. This is sometimes referred to as Promiscuous Mode. Small transport systems support 8-bit Destination IDs, while large transport systems support 16-bit Destination IDs. For the purposes of this article, we will keep to small transport systems, but it is exactly the same with a large transport system with the exception that you would use Hex FFFF instead.

In a simple example where you have a four-port switch and 4 endpoints in a small transport system, the first transaction could be a maintenance read using a Destination ID 0xFF and a hop count of 0 to the Processing Element Features Capability Register (CAR).

CAR will help you understand what type of PE it is—Processor, Switch, Memory or Bridge. In the example, there would be a switch. One of the aspects of a discovery algorithm needed is to consider whether or not it is intended to operate within a multi-host environment. If required, then there are a few additional things to do. The first is to determine if another host has already discovered the device. If so, understand if that host is a higher priority host, or if that host has already enumerated the system.

There are a number of mechanisms that the RapidIO specification has put into place to facilitate this aspect of Discovery. Those are the Host Based Device ID Lock register and the "Discovered" bit within the Port General Control and Status Registers (CSR). The Discovered bit is a simple status bit, that if set, will obviously tell you if another host has encountered this PE. The Host Base Device ID Lock register is a semaphore mechanism that allows a host to determine ownership of a PE or system.

For the purposes of the one switch / 4 Endpoint example, assume that no other host has claim over the switch that was found, then lock the device and set the Discovered bit. The next steps require an understanding of how many ports the switch has, which port of the switch has been entered, and which port to begin to explore out of. To do this, issue a maintenance read to the Switch Port Information CAR. The bit-fields within this register will provide the information. The port chosen is really arbitrary. Assume you came in through Port1 and chose to explore each of the other ports on the switch in sequential order from lowest to highest port. Start by discovering what lies beyond Port 0. At this point in the discovery algorithm, there may be a number of other capability registers in preparation for the third step of initialization.

In order to discover what exists out Port 0 simply decide to issue a maintenance packet with a Destination ID of 0xFF and hop count of 1. First, configure the routing table in the switch to "steer" the next maintenance packet out Port 0. There can be two ways to configure a switch's routing table...
- Standard and Optional Proprietary methods. The standard interface allows one to specify the output port number for any given Destination ID in one global look up table. There are a number of other details associated with routing tables such as default port for non-programmed entries.

In this example, the entire routing table is not ready to configure as the entire system has not yet been discovered. None the less, the next maintenance packets need to be forced out Port 0. Using a Destination ID of 0xFF, simply write an 0xFF into the Standard Route Configuration Destination ID Select CSR and an 0x0 into the Standard Route Configuration Port Select CSR. Again, look at the simplest form of routing table configuration so consult the User Manual for the switch that may be used for more advanced configuration requirements as they all offer different sets of advanced capabilities. Make sure to enter a route for the return path, back to the host from which you are discovering the system. Let’s presume that the Destination ID of our Host is zero, write a 0 into the Standard Route Configuration Destination ID Select CSR and a 1 (because the switch on Port 1 was entered) into the Standard Route Configuration Port Select CSR.

Before leaving the switch, discover what the width of the link is by using the Serial Port x Control CSR. Optionally, many devices also allow you to discover the link speed. Both of these parameters are useful when trying to determine Throughput and Congestion touched on later in this article.

Now we have configured the routing table to allow the maintenance packet to traverse the switch if the Destination ID of 0xFF and a Hop Count of 0x1 are used. At this point, consider issuing the maintenance packet and attempt to read the Processing Element Features CAR in whatever device that may be connected to Port 0.

But what if there isn’t anything connected to it? A bus timeout will happen after a period of time because no response to the read request will arrive. While this approach is perfectly fine, it will slow down the discovery process due to cumulative timeout delays. This can be appreciable for large systems with many 16-port switches for example. To make it a little more time efficient, before issuing any maintenance transactions out port 0, read the Serial Port 0 Error and Status (CSR) and determine if the port has trained or not. This could save costly bus timeouts and shorten system configuration time considerably.

If two link partners are physically connected at the same bus frequency they will automatically go through a training sequence and establish a PHY level connection and hardware will set a flag indicating that it has trained. For advanced users, some PEs support software configuration of bus frequency as well so consider implementing a frequency-scanning algorithm that sweeps all frequency’s to discover devices across different bus clock domains.

Presuming the port is trained, issue the maintenance packet read to the Processing Element Features CAR in whatever device may be connected out Port 0. You discover that it is an Endpoint. As with any PE discovered, repeat the test described earlier to see if another host has locked the PE and already discovered it. Assuming one has not, lock the PE and set the Discovered bit. Now enumerate it and initialize or configure certain capabilities within it. Alternatively, do this after discovery.

The next step is to explore out of the next port of the switch. Simply overwrite the routing table entry of 0xFF to Port 0 with 0xFF to Port 2. Why not Port 1? Remember, the switch came in on Port 1. The next Port to explore in the sequence is Port 2 and then Port 3. The discovery algorithm will want to keep track of this. The return path route is the same as what used for Port 0. In this process, keep track of the network topology as it is discovered. In some systems you will encounter multiple paths between PEs and many circular loops in full mesh systems. The discovery algorithm will need to detect when the revisited parts of a network have already discovered and where in the existing
known network map it happened to reconnecting to.

Read more In-Depth Technical Features

Enumeration

Although there are certainly exceptions to this, Enumeration is largely about assigning unique Destination IDs to Endpoints. Processors, Memory devices and Bridges all require unique Destinations IDs before a full routing table can be configured within each switch.

In a fixed system, you could sequentially assign non-reserved Destination IDs to each Endpoint you find. Typically Destination IDs of 0x0, 0x1, and 0xFE are reserved for host specific functions, and 0xFF is the default power up Destination ID value. So starting at 0x5, for example and sequentially assigning Destination IDs 0x6, 0x7, etc is a perfectly acceptable practice. However, in a more dynamic environment where new blades or mezzanine cards may be inserted or removed, you may want to take a different approach. Why? Well let's assume that you assign a Destination ID of 0x5 to a device on Port 1 of a switch and 0x6 to a device on Port 3 of a switch because nothing is connected to Port 2 of the switch. What happens if the card on Port 1 that contains a single Endpoint is replaced with a card that has a switch and several endpoints? Or what happens if a new card is plugged into Port 2 where there was previously no card? You are left with either having a discontinuity in Destination IDs across a system - and even across blades - or you need to reenumerate the entire system. The former may be perfectly acceptable in some systems but the latter may incur undesirable system wide disruptions in communications due to something as simple as a hot plug event because it would require likely all routing tables to be reconfigured. You could simply decide to count by 16’s and use 0x5, 0x15, 0x25, etc. This approach may provide you with adequate room but quickly consume all of the available Destination IDs if limited to a small transport system.

There is no perfect answer here but if you know that your system will never support a blade with more than 5 endpoints, you could optimize your approach by leaving room for 5 devices for every port on a switch that could potentially support a blade. One of the key things to recognize is that once you change the Destination ID of an Endpoint, in order to access it using the new Destination ID, you will need to enter a new routing table entry for that Destination ID.

In the simple one switch four-endpoint example, you could enumerate each endpoint individually and then initialize the routing table at the end for each of the new Destination IDs assigned. In a multi-switch environment however, you need to keep in mind that while some switches have full routing tables configured, others may be in various stages of “forced steering” of specific Destination IDs or partial groups of Destination IDs. So you will need to decide whether you configure all of the tables at the end of discovery and enumeration or if you do it on the fly.

Initialization

Initialization involves the configuration of standard and proprietary capabilities of PE’s and in some cases applications themselves. The nature of initialization is very specific to the devices you have in your system, how much of their non-standard capabilities you wish to utilize and the how adaptive your applications need to be to system resource capabilities. In general, initialization can either be performed in parallel to the discovery and enumeration processes, or after the fact.

One of the primary aspects of initialization is ensuring all of the routing tables in each switch in a system reflect the newly enumerated endpoint Destination IDs. This will allow endpoints to
communicate with each other without requiring knowledge of the system topology itself as switches will handle the routing of packets and ensure delivery using the link level store and forward techniques associated with RapidIO devices.

As with many standard RapidIO capabilities there are both standard and sometimes proprietary methods of configuring routing tables. You will want to consult with the user manuals to decide if the benefits of using proprietary capabilities are desired and initialize appropriately. Adaptive algorithms for leveraging device specific capabilities are possible based on the specific Device and Vendor ID information found in the Device Identity CAR. In the case of some proprietary features such as routing tables however, you will want to ensure that there is a homogeneous set of capabilities across all switches before you choose to implement a specific non-standard routing mechanism. Otherwise the capability may not be something that can be supported across the entire path from endpoint to endpoint.

Many switches support features such as Multicast where, again, both standard and proprietary initialization of registers may be required. For example Multicast mask registers need to be configured associating multicast ports to Destinations IDs.

A number of standard common capability registers exist in both switches and endpoints that facilitate determining what capabilities exist in a device and therefore what may need to be initialized. Source and Destination Operations CARs help you understand the types of RapidIO transactions that a PE can support. These range from simple I/O reads and writes to message level transactions using mailboxes and doorbells. Here is an example where you may want to build flexibility into an application to optimize, say, reliability or communication efficiency based upon the types of transactions supported. For example, if the endpoints that a particular application needs to utilize all support nWrites with response and Streaming writes, for larger payloads, then it could improve reliability and efficiency over having to use just nWrites alone. A smart application could therefore automatically detect these capabilities and optimize performance.

Extended Features and Proprietary CSRs are often also available for software to determine what other capabilities exist and could be leveraged once initialized. To facilitate this, RapidIO supports a mechanism for interrogating Extended Features through a series of linked list structures. A specific area in maintenance register memory space is also assigned to support configuring proprietary functionality.

It is through standard, extended and proprietary features that a whole host of higher-level error, event and performance management capabilities can be exploited.

When properly configured, software can be interrupted when certain parts of a system begin to get congested by using ingress buffer high water marks to generate port writes interrupts to a host processor. As links begin to exhibit too many CRC errors, redundant routes can be enabled to steer traffic around problem areas as host software works to re-train devices on either side of a link. Throughput monitors can be used to resource balance networks and optimize peak performance. These are just a few of the capabilities that exist in today's switches and endpoints that can be and in many cases already are being used.

At a higher level, System architects must decide how endpoints in a system are to be made aware of what system resources are available to them. This could be predetermined and hard-coded into software in a fixed system. However it may be the host processor's responsibility to communicate this to some or all endpoints in a system. A method of sharing of system topology, Destination IDs, routing information and System Resources and their capabilities needs to be considered. This information may also need to be refreshed from time to time as system resources are added or
removed from some systems. In some systems, you may want to consider how redundant hosts will communicate and what information they should share so that soft failover scenarios can exist. While some of these topics go beyond the scope of what we are discussing today, they are aspects that you may want to keep in mind.

Once initialization has completed, endpoints are now able to use the system. The Master Enable bit, which exists in the Port General CSR, is provided as the common mechanism for communicating this "ready" status to all endpoints.

With the proliferation of RapidIO switch and endpoint devices today, there is a significant diversity of rich features to be exploited in software. They can be used for both run time and diagnostic purposes and there are real world examples of products using them today.

In conclusion, RapidIO specific software need not be complicated for simple systems and there are significant capabilities to support the more complex requirements demanded of higher end systems.