



Serial RapidIO SW and HW ATCA/uTCA

AdvancedTCA/MicroTCA Hardware Development
Santa Clara
Wednesday, October 28, 2009
Daniel Naar - Senior Applications Engineer
dnaar@mc.com

Goal

- Through examples present Mercury's software capability to support configuration and access to SerialRapidIO functionality in ATCA/uTCA

Overview

- RapidIO Overview
- Software Stack
- Parallel processing example to demonstrate
 - Configuring the Fabric
 - Application Access to RapidIO Functionality
 - DMA
 - PIO
 - Doorbells
- Example of hotswap support

RapidIO

- RapidIO
 - Topology agnostic
 - Low overhead
 - Low latency
 - Hardware level guaranteed delivery (link level retry)
 - Multiple priority levels
 - Multicast
 - Read/Write and message semantics
- Excellent match for applications requiring
 - High bandwidth, low overhead data transfer
 - Low latency
 - Low CPU overhead (minimal software stack)
 - Arbitrary network topology

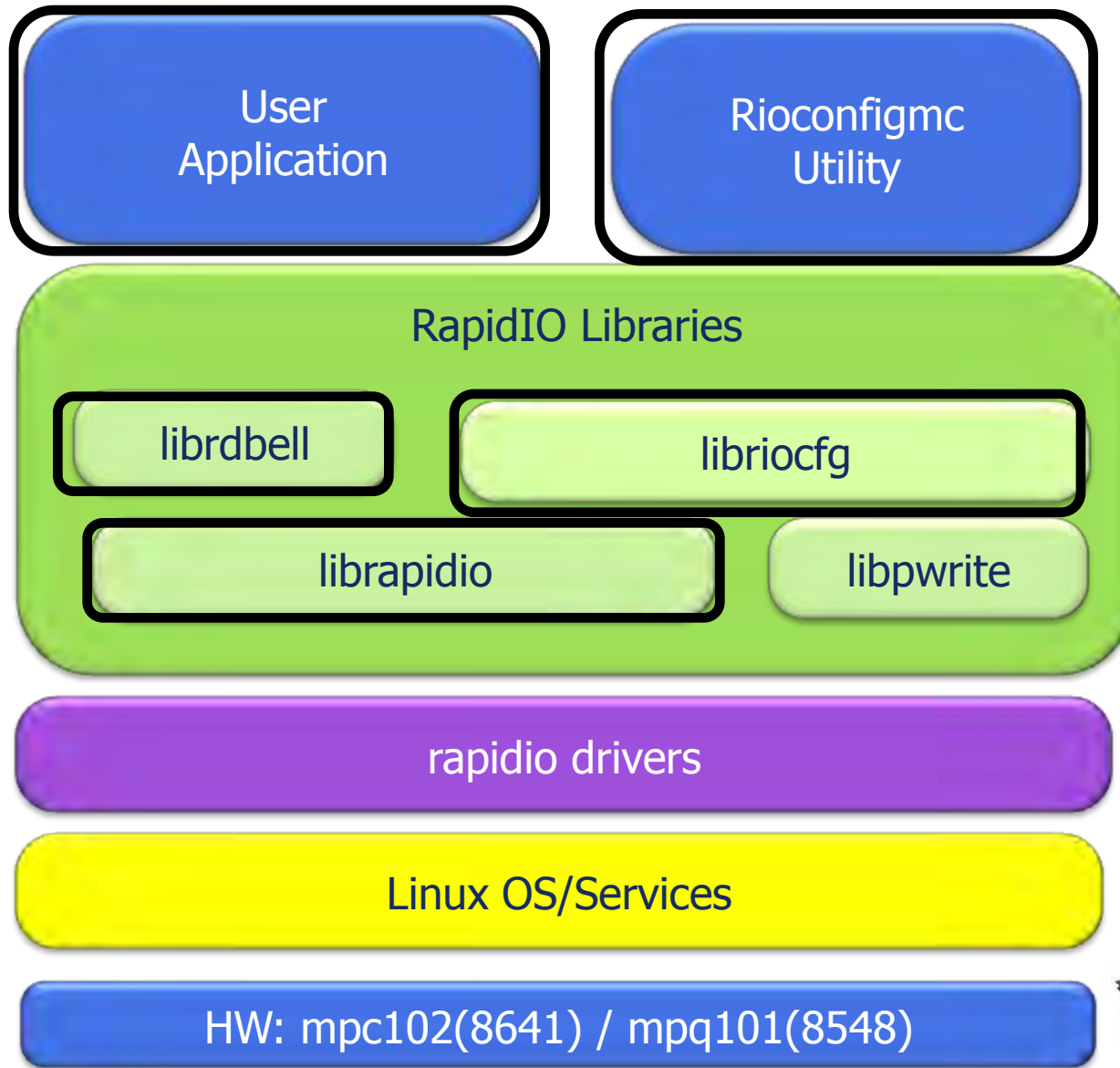
Mercury Software Capability

- Diagnostic utility
- Configuration utility
- Configuration library
- Endpoint library
 - Mapping
 - DMA
 - Maintenance packets
- Doorbell library
- DSP code loader for Mercury DSP-AMCs
- Configuration library
 - Provides configuration capabilities in custom applications

Software Elements Discussed

- System configuration utility and Library
 - rioconfigmc, libriocfg
- Direct endpoint access (PIO, DMA, maintenance packets)
 - librapidio
- Doorbell functionality
 - librdbell

Software Stack



High Level Products – ATCA / uTCA Hardware

Systems



6 & 12 slot
uTCA



2, 5, 14 slot ATCA



Blades



Quad AMC RapidIO
Carrier



RapidIO Switch



BeamFormer Compute
Engine

uTCA Switch

AMCs

(Advanced Mezzanine
Cards)



Fiber AMC



8640D AMC



FPGA / TI DSP AMC



Quad TI DSP AMC



Virtex 5 FPGA
AMC

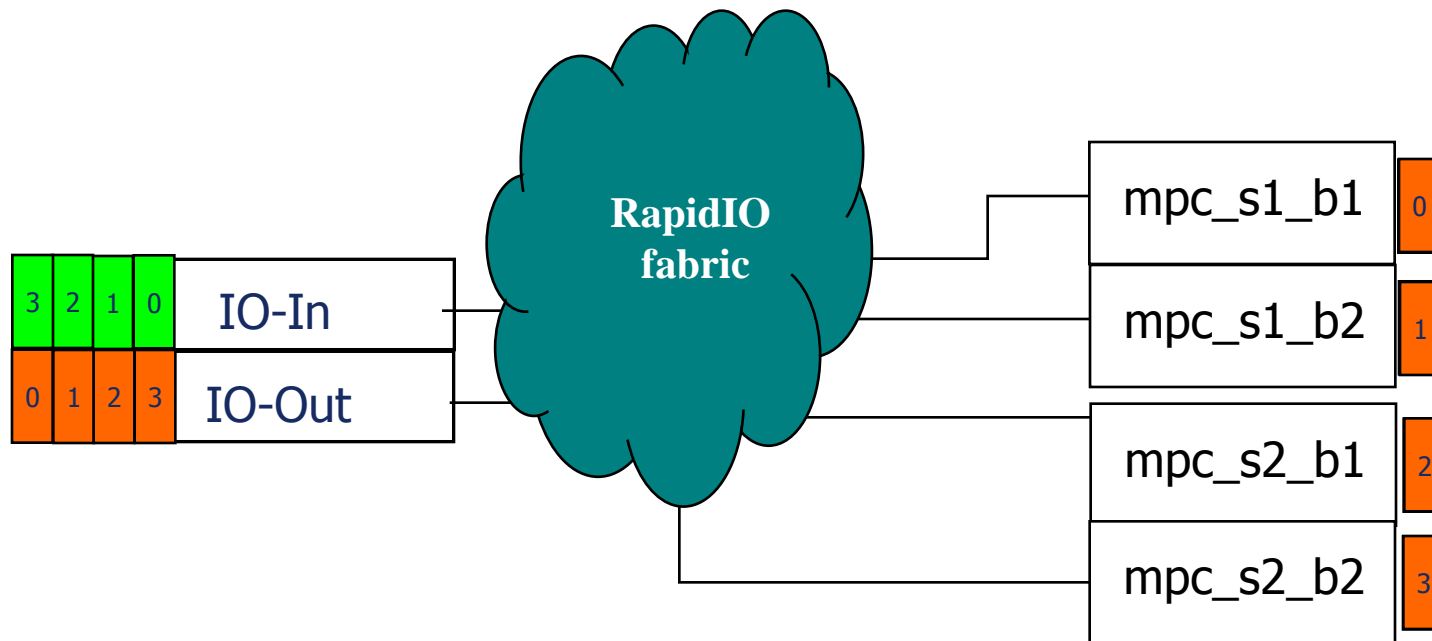


8548 AMC

A broad choice of components supporting both ATCA & uTCA

Parallel Data Processing Example

- Process bound application, overlap processing with IO
- Single buffered
- Doorbell used as "data arrived" signal



What Does Software Need To Do

- Configure fabric
- Setup RapidIO access on MPC-102
- Run processing hot-loop

What Needs To Be Done To Configure SRIO Fabric?

- User
 - Input configuration file that describes the system
- Utility
 - Reduce system level description to a network of switches and endpoints
 - Verify the presence of switches and endpoints
 - Calculate and write the routing tables for each switch
 - Program and verify RapidIO IDs of endpoints

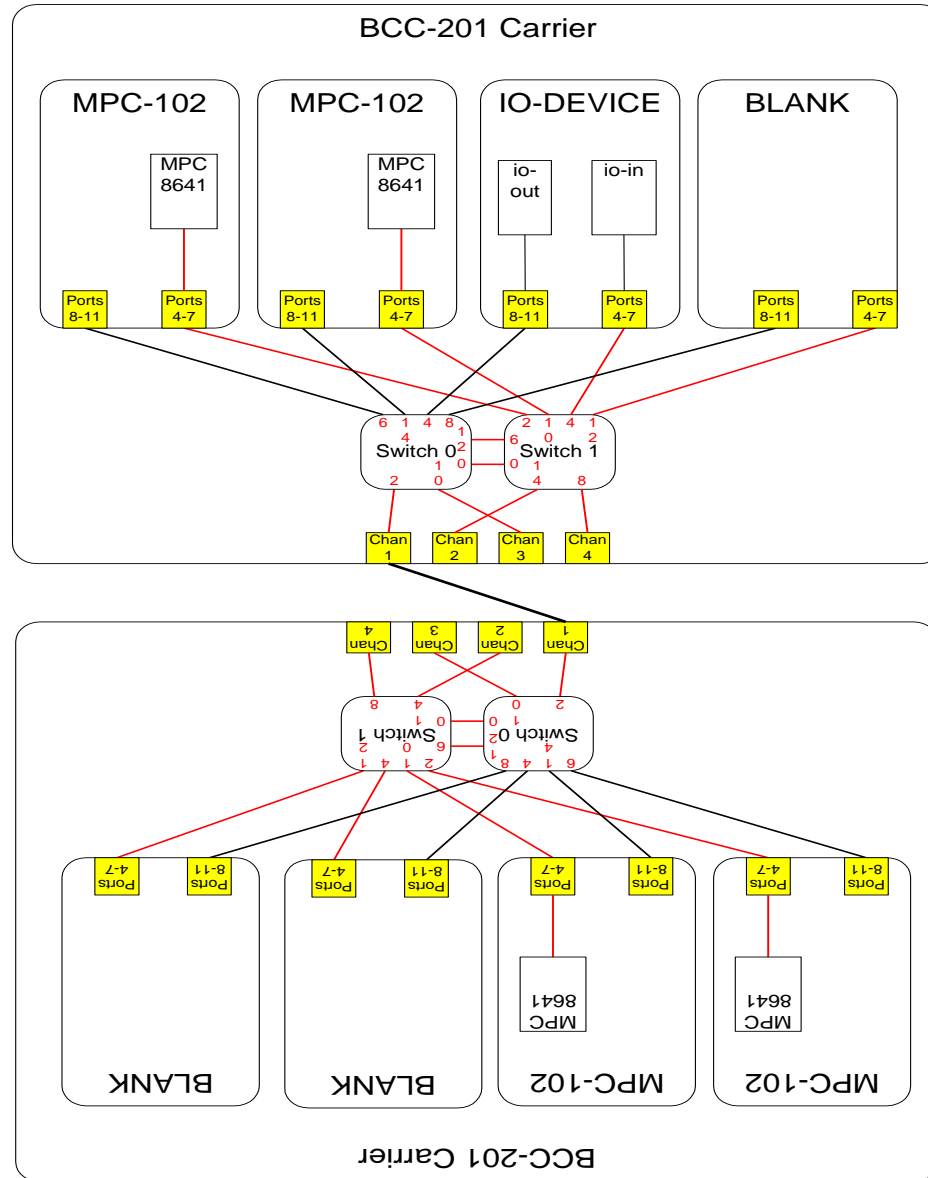
Configuration Utility: RioConfigmc

- Configures the fabric based on user defined configuration file
- Executes on Mercury's MPC-102 and MPQ-101
- Supports all Mercury TCA products
- Extendable to custom products with constraints

System Description

- 5 Slot Full Mesh Chassis
- 2 BCC-201 ATCA carriers inserted in first two slots
- 4 Mercury MPC-102 AMCs (8641D)
 - Two on each carrier
- Custom IO AMC
 - Single AMC
 - Two endpoints

System Topology



System Configuration File

```
# Declare Chassis : chassis <type> <name> <nslots=#> <bplane=type>
chassis ATCA atca_chassis nslots=5 bplane=mesh
```

```
# Declare the carriers : board <type> <name>
board BCC-201 bcc_s1      # AMC carrier in slot 1
board BCC-201 bcc_s2      # AMC carrier in slot 2
```

```
# Declare processing AMCs: board <type> <name>
board MPC-102 mpc_s1_b1   # AMC to be connected to slot 1 bay 1
... repeat for each board
```

```
# Declare Customer IO Board - Declare the board and all options on one line
board GenericAmc ioboard fpipe.4=serial_4 vendorid.4=0x000F deviceid.4=0x0001 fpipe.8=serial_4
  vendorid.8=0x000F deviceid.8=0x0002
```

```
# Connect Carriers – Syntax: <chassis_name.atca.#> <carrier>
connect atca_chassis.atca.1 bcc_s1
connect atca_chassis.atca.2 bcc_s2
```

```
# Connect AMCs - connect brd.mca.# amc_brd
connect bcc_s1.amc.1 mpc_s1_b1
... repeat for each amc
```

```
# Assign Rapidio Ids : assign rioid <board.node.#> <#>
assign rioid mpc_1_1.node.1 11
... repeat for each endpoint
```

```
# assign host <board.node.#>
assign host mpq_4_1.node.1
```

Run Configuration

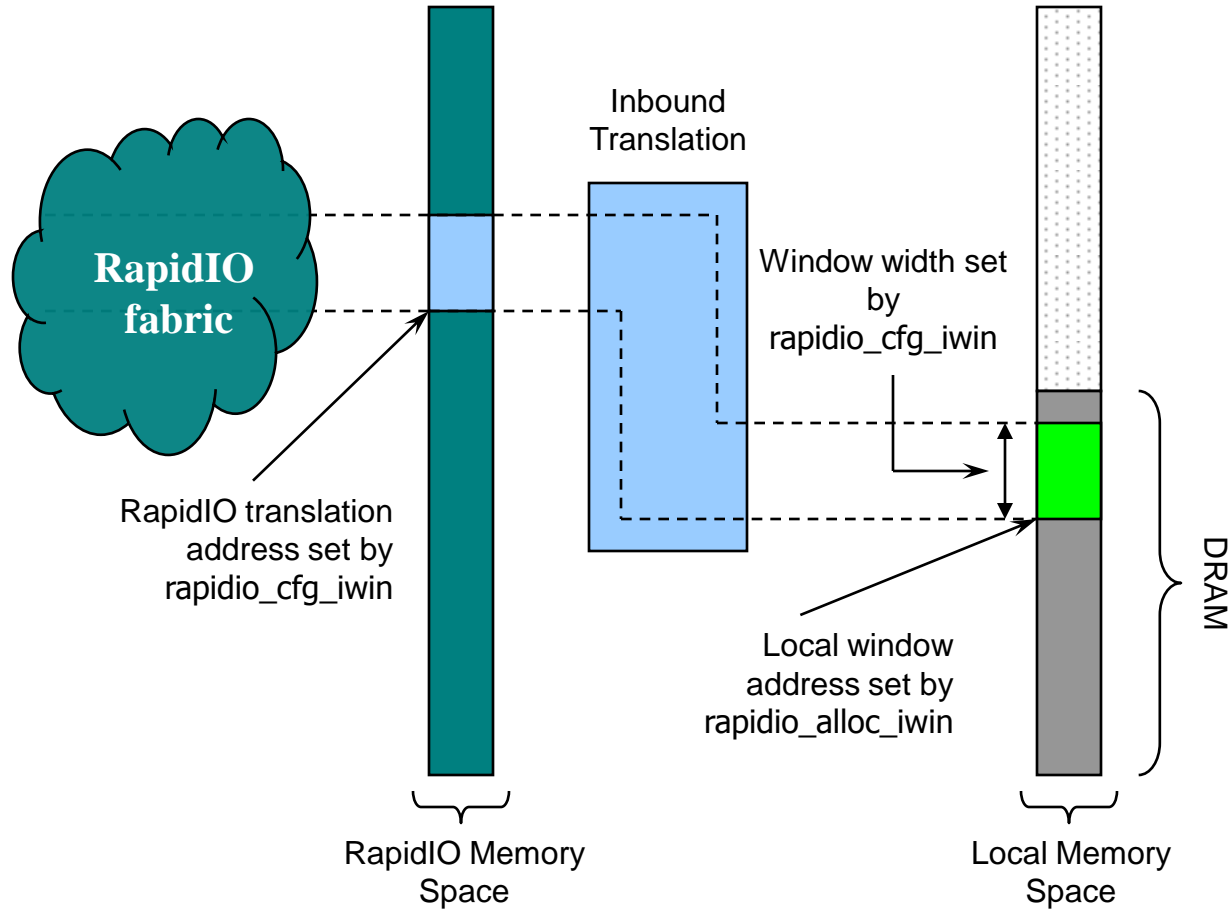
- % rioconfigmc -i simple_example.cfg
- Running rioconfigmc
 - parses and verifies configuration file
 - verifies system configuration matches hardware
 - calculates shortest path routing
 - programs switches

What does the application need to do?

- Configure inbound mapping as target DMA from IO
- Configure outbound mapping to IO device
 - Use mapping to program IO devices and to setup outbound DMA
- Setup DMA to transmit data to outbound IO device.
- Configure doorbell interface to receive data arrival notification
- Manage and process data
 - Receive doorbell
 - Process data
 - Start outbound DMA

Inbound Mapping Overview

MPC-102



Setting Up Inbound Mapping

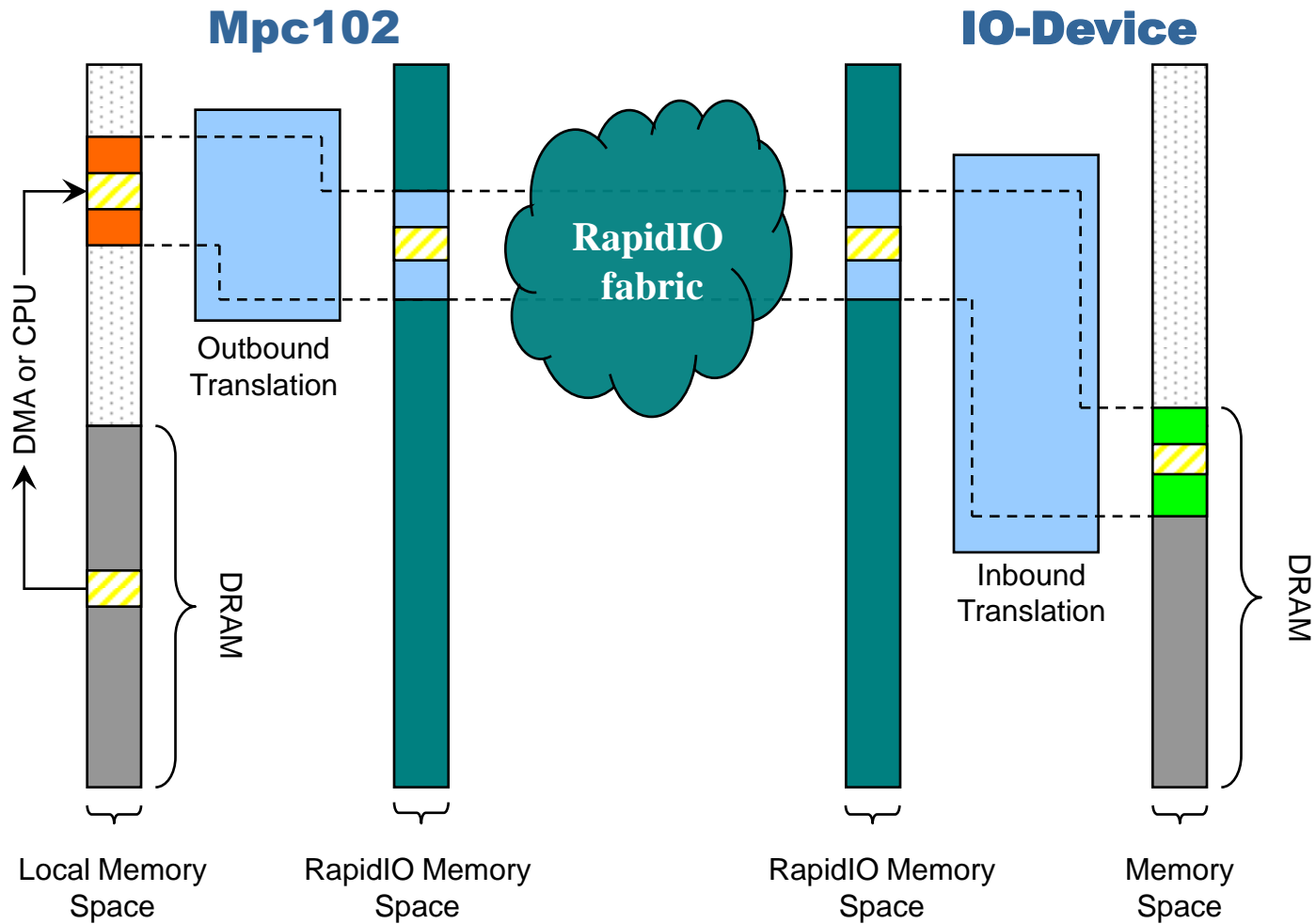
```
// Allocate mapping resources
rc = rapidio_alloc_iwin (nbytes, 0,
                        &iwin_mmem_h, // Handle to window memory
                        &iwin_id);    // Window Id

// Configure the mapping
rc = rapidio_cfg_iwin(iwin_id,      // window handle
                     NODE_BASE_ADDR, // rapidIO address
                     e_rd_snoop,    // read packet type
                     e_wr_snoop,    // write packet type
                     nbytes         // size of inbound window
                     );

// Get a pointer to the local memory.
p_buffer = rapidio_get_mptr(iwin_mmem_h);

// IO device is DMAs data to memory accessible via p_buffer for processing
```

Outbound Mapping To IO Device



Setup Mapping To IO Devices

// Allocate the mapping resource, inbound IO

```
rapidio_alloc_owin (nbytes, 0, &ioin_h, &io_in_win_id);  
ioin_rioid = Value_DefinedIn_Configuration_File;
```

// Outbound IO

```
rapidio_alloc_owin (nbytes, 0, &ioout_h, &io_out_win_id);  
ioout_rioid = Value_DefinedIn_Configuration_File;
```

// Configure the mappings

```
rapidio_cfg_owin(ioin_win_id, ioin_rioid, RAPIDIO_BASE_ADDR_OF_IO_DEV, ...);  
rapidio_cfg_owin(ioout_win_id, ioout_rioid, RAPIDIO_BASE_ADDR_OF_IO_DEV, ...);
```

// Get a local pointer to remote memory on inbound IO device, and configure

```
p_io_in = rapidio_get_mptr(io_in_h);  
setup_io_input_device(p_io_in);
```

// Get pointer to remote memory on outbound IO device, and configure

```
p_io_out = rapidio_get_mptr(io_out_h);  
setup_io_output_device(p_io_out);
```

Setup Doorbell Access

// Open the doorbell interface.

```
rc = riobell_open(queue_depth, flags);
```

// To receive a doorbell.

```
doorbell_t doorbell;
```

```
rc = riobell_rcv(&doorbell, RIOBELL_WAIT, 0);
```

// Accessible fields.

```
doorbell.doorbell ; // A 16 bit value.
```

```
doorbell.sourceid; // Source RioId
```

```
doorbell.targetid; // Target RioId.
```

Setup DMA To OutBound IO Device

```
// Allocate handle to a chain
```

```
rc = rapidio_alloc_dma_chain( e_low,    // priority
                             &ioout_chain_id); // dma chain handle (returned)
```

```
// Queue transfer descriptions to the chain. Simple case, no striding.
```

```
rc = rapidio_que_dma_raw(ioout_chain_id, // dma chain handle
                          iwin_mmem_h,  // source memory handle
                          0,            // source buffer offset
                          io_out_h,     // Memory descriptor off outbound io device.
                          0,            // destination buffer offset
                          nbytes,       // Number bytes transferred
                          ...           // no striding.
                          );
```

```
// The transfer description is complete and chain id can be used during hot-loop.
```

Application Data Handling Loop

```
for(;;) {  
    // Wait for doorbell from IO device  
    riodbell_rcv(&ioin_doorbell, RIODBELL_WAIT, 0);  
  
    // Process the data that is now in memory pointed to by p_buffer.  
    app_process_data(p_buffer);  
  
    // Done processing, start dma, and wait until complete.  
    rapidio_start_dma(ioout_chain_id, RIO_DMA_WAIT_INT);  
}
```

Hot-Swap support

- Capability to replace a RapidIO endpoint or board without reconfiguring the entire system.
- Capability to detect that an endpoint has been inserted or removed, and allow the user application to take an action.
- Does not define policy (i.e. action to be taken). This is left to the user application.
- Possible Actions that user policy could enforce upon detecting a lost endpoint
 - Bring system down.
 - Run in a diminished capacity.

How Is Hot-Swap Supported?

- Insertion and extraction port write events generated by switches and are device specific.
- sRIO switch must support port reset
- User application is notified of insertion or extraction via an event queue accessible via libriocfg API.

Changes To Configuration Support Hot Swap

- Enable events on switches by adding following
 - enable events
- Modify command line options to rioconfigmc
 - Add the “-d” option to launch daemon to service event interface

Code Snippet Event Queue

```
// Create the queue
```

```
ret = riocfg_createEventQueue("my_queue", maxDepth, &qHandle);
```

```
// Enable specific events of interest.
```

```
ret = riocfg_enableEvent(qHandle, e_LinkGood);
```

```
ret = riocfg_enableEvent(qHandle, e_LinkDown);
```

```
while(1)
```

```
{
```

```
    int j;
```

```
    rioevent_t event;
```

```
    printf("\nWaiting for event ...\n");
```

```
    riocfg_getEvent(qHandle, &event);
```

```
    app_handle_event(event);
```

```
}
```

Example of Event (Link Down)

- Deactivate mpc_1_1 via shelf manager
 - `cli deactivate 82 2 #` Detailed command line interface
- Rioconfigmc receives port writes and forwards via event queue.
- Output from real run
 - `event=LINK DOWN`
 - `slotInstance=1`
 - `bay=0` (Detected on carrier)
 - `portType=2` (AMC)
 - `port=513` (0x201 -> Bay 2, Fatpipe-3: 1)
- User app can take action based on event.

Overview

- Software Stack Overview
- Configuring the Fabric
- Examples using DMA, PIO, and Doorbells
- Configuration Support for Hotswap

References

- http://www.mc.com/products/software/ensemble_rapidio_middleware.aspx
- www.rapidio.org



Q & A