I_2O RAID Module

Dawn Tse
Mylex Corporation
Senior Staff Engineer, Project Manager

I_2O RAID Module

1. Introduction

In an I_2O environment, RAID (Redundant Array of Independent Disks) controller functionality may be implemented by means of either a device driver module (DDM), or by an intermediate service module (ISM). Through whichever means, the resultant functional unit is referred to as the RAID module.

This section discusses RAID module functions, architecture, and issues requiring consideration when implementing the RAID module in an I_2O environment.

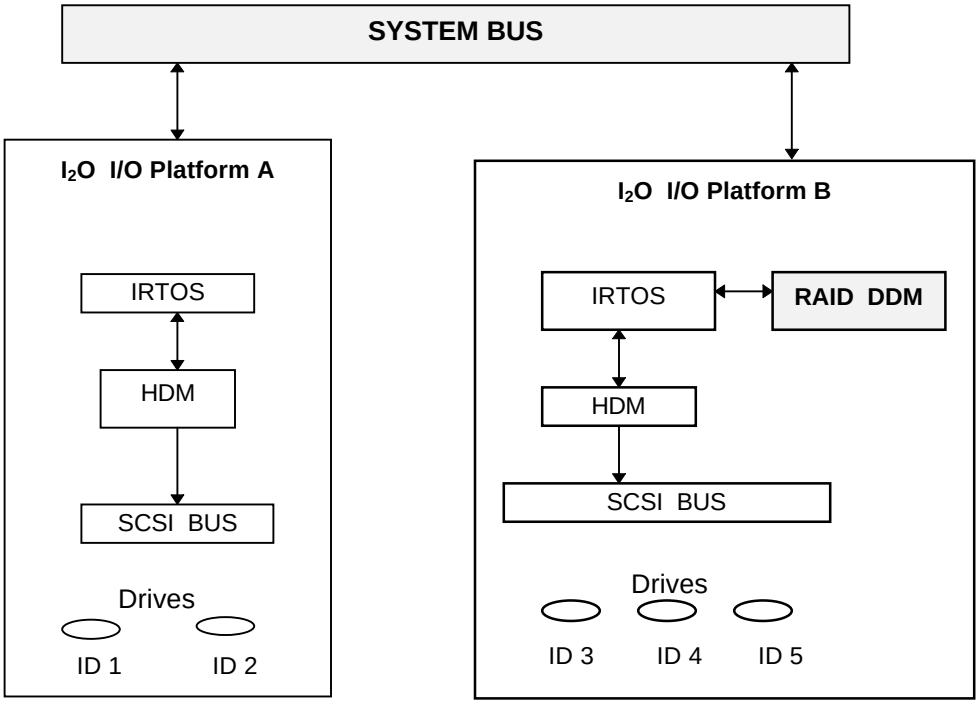
2. Architecture

RAID module functions in the I_2O environment are performed independent of the operating system, independent of the type of microprocessor, and independent of the type of bus (either the system bus or the local bus).

The Mylex RAID module is loaded into the I/O platform and is initialized by the standard I_2O DDM loading procedure. Physical devices under the control of the RAID module are attached to the RAID module through the *attached device* mechanism. Moreover, physical locations of the devices can be within the same I/O platform (IOP), or on different IOPs.

Contrary to the traditional RAID controller, the Mylex I_2O RAID module has no dependencies on hardware platforms, such as types of system bus (PCI, ISA and etc.), or on types of host adapters, types of device controllers, etc. Since the RAID module can be added to any I/O platform that is compliant with the I_2O specification, it widely expands the application of RAID technology without major redesign effort on hardware, firmware, and system drivers. Furthermore, because of the flexibility of the I_2O architecture to allow peer-to-peer messaging and data transfer, expansion of RAID capacities is beyond the limitation of physical configuration issues (e.g., number of buses on a particular IOP, maximum number of devices supported on a specific bus, etc.).

In addition to expanded RAID capacities, the new added flexibility of I_2O also may enable the RAID module to *load balance* more efficiently, therefore, increasing overall performance. In this environment, a RAID module communicates to another HDM or DDM through the I_2O messaging system (the other DDM can also be another RAID module). Therefore a hierarchical RAID configuration can be implemented without major redesign of the RAID module. Moreover, through vendor-unique messages, two RAID modules can backup each other to recover from a single IOP failure (*fail-over* implementation). The following illustration shows such a configuration.



Through the I_2O interface, the RAID module can control any and all of the five physical devices shown in the illustration and can reside on either Platform A or Platform B. The characteristics of either platform's SCSI bus and the system bus are hidden from the RAID module.

3. Operating Environment

The RAID module can operate on any I_2O I/O platform (IOP) consisting of a processor, memory, and I/O devices. This IOP can either be on the motherboard (designed to control system *and* private I/O devices), or on a controller card (designed to control private functions specific to certain bus implementations). Inside the IOP is an IRTOS (special-purpose real time operating system). The IRTOS must provide the following services for the RAID modules:

- OS services for the execution environment, including memory allocation, task management, ISR, timer, etc.
- DMA engine for accessing the host system memory, and to transport data between modules
- A message layer for the communication between modules
- Configuration service for loading and initializing DDMs, hardware configuration, and registration of logical device(s).

There is a system resource manager which resides on the host system that initializes the IOP into the system, and provides information about other IOPs. The system resource manager also provides for the configuration dialog for configuring the RAID virtual drives.

4. Initialization

A functional embedded I_2O environment must be established before the RAID module can be loaded. If the RAID module has not been stored, install the DDM and *Load DDM request* from the host to the IOP *load RAID* module.

If the RAID module has already been stored on the IOP (as part of the IOP initialization sequence), they load and initialize the RAID module with its parameter block when they scan its physical devices. Since the IOP maintains a list of physical devices, their locations, and their associated DDMs, when the IOP instructs the RAID module to initialize with an *objectAttach* message, it also passes the device's logical IDs (TID), which were assigned to the RAID module previously.

The RAID module then sends *ReadBlock* request messages for each device to fetch the configuration table from the device to memory. Once the configuration is verified, the RAID module claims those I_2O objects, and then creates one *block storage class* I_2O object for each virtual drive. For each such I_2O object, the RAID module registers an entry point, known as an *even service routine* (ESR) to place its information in the IOP configuration table. A TID is issued to the I_2O object by the IOP and the ESR is dispatched to process any message that is addressed to this TID.

When the physical configuration changes, which evidenced by that either IOP cannot locate an I_2O object which was assigned to RAID module, or that the RAID module detects the configuration table to be invalid, the IOP or RAID module then requests a configuration dialog by setting a flag in the IOP's configuration database. The host initiates the configuration dialog at a later time. During the configuration dialog, the RAID module interacts with the system, and ultimately the user, to establish new configuration parameters.

5. Configuration

Configuration dialog is initiated by the host. The RAID module requests a configuration dialog by setting a flag in the IOP's logical configuration table. The host sends the RAID module a *ConfigDialog request* message, whose response provides a configuration template to be completed by the host. The host sends a *ConfigResponse request* that returns the completed template to the RAID module and requests the next dialog. Each dialog result in another *Config-Response request* until the RAID manager sets the final flag in the reply.

For each device configured as part of the RAID, the host sends an *AttachObject* to the RAID module. If the device is on another IOP, the host must send *ConnectIOP* to establish aliases for sending messages between IOP executives. Before an IOP can assign a device (an I_2O object) to the RAID module, it must determine if the device is local or remote. If the device resides on another IOP, the local IOP sends a *connection setup request* to the remote IOP to establish the connection and assign the alias TIDs.

The RAID manager generates a unique device tag for each device and stores the relationship of the device tag to a specific physical device in its module parameter block. Furthermore, the RAID module also stores the physical information of a device on the device itself, so that the next time the IOP is booted, the RAID module can determine whether a device is the same, is a replace-

ment, or should be treated as a new device. When the RAID module detects that hardware has been changed, it will not create the virtual system drive as an I_2O object. Instead, it will set the Configuration dialog flag in the IOP's logical configuration table and wait for the host to initiate a configuration dialog.

6. Memory Management

There are three types of memory affecting the RAID module in an I_2O environment:

- System memory
- IOP private memory
- Shared memory

System memory is accessible only from the system bus. This type of memory only has system memory addresses.

The second type, IOP private memory, is accessed by the local IOP. This type of memory has only local memory addresses.

Shared memory, the third memory type, can be accessed by the local IOP, by the system bus, and by other IOPs. Shared memory on one IOP is considered to be system memory by other IOPs that can access it. Shared memory has two addresses – the system memory address and the local memory address.

The RAID module uses IOP private memory for its internal operation (e.g., variables, storage of internal tables, etc.). The RAID module uses shared memory for the cache. During initialization, it calls IRTOS to allocate shared memory as the cache buffer. Once allocated, the cache buffer is managed by the RAID module, and not released back to the IRTOS until the removing of the RAID module.

7. Data Path

Several types of data block movement are possible in the I_2O RAID module implementation.

7.1 Data from remote IOP share memory to host system memory.

The RAID module issues a *Read* request to a DDM which resides on a remote IOP with a destination SG list which contains a host system address. If the DDM is a cache controller, and request data is in the shared memory of the target IOP, the DDMs move data from its *share* memory to the host system.

7.2 Data from a physical bus on a remote IOP to host system memory.

The RAID module issues a *Read* request to a HDM which resides on a remote with a destination SG list which contains a host system address. The HDM issues a SCSI command and moves data from the local SCSI bus to the host system memory.

7.3 Data between local IOP's local bus and host system memory.

The RAID module issues a *Read* request to a HDM which resides on the same IOP as itself. The HDM moves data from the local SCSI bus to host memory.

7.4 Data between shared memory and host system memory.

The RAID module calls the IRTOS DMA APIs to move a data block between the host memory and local shared memory.

7.5 Data between remote IOP's shared memory and local IOP's shared memory.

The RAID module issues a *Read* request to a DDM which resides on a remote IOP with a destination SG list containing a local shared memory address. If the DDM is a cache controller and the request data is in the shared memory of the target IOP, the DDMs move data from its shared memory to the local IOP's shared memory.

The Raid module issues a *Write* request to a DDM which resides on a remote IOP with a local shared memory address as a source address. The DDM has a write-back cache flag *on*, it moves data from the source into its local shared memory and marks dirty.

7.6 Data between remote IOP's local bus and local IOP's shared memory

The RAID module issues a *Read* request to a HDM on a remote IOP with its local shared memory address as destination address. Remote HDM moves data from a local SCSI bus to the local IOP's shared memory (and vice versa).

From the possible data paths described above, the RAID module must provide a system memory reference when it sends a request (message), even though the RAID module executes in the IOP's local address domain. However, when the RAID module operates on the data in functions such as parity generation, it must translate the system address to a local address. As a result of this requirement, the RAID module has an ATU (address translation unit). Based on the parameters provided to the RAID module during the registration (such as base address of shared memory, shared memory length, base offset between the system base address and local base address, etc.), the ATU can translate one address into another.

8. RAID Module Normal Operation

RAID module normal functions during the messaging layer include:

- Communication between the RAID module and others
- Read operations

- Write operations

8.1 Communication between the RAID module and others

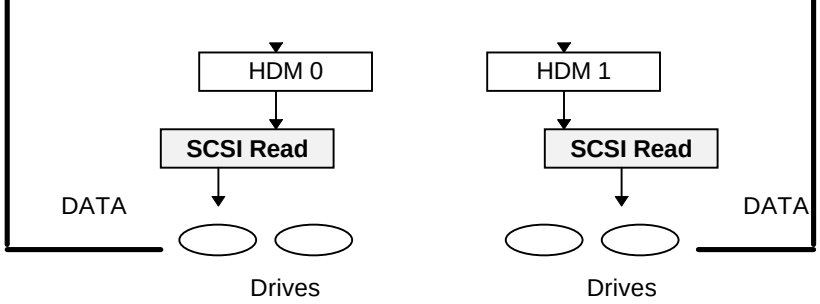
The RAID module creates virtual interfaces with other modules through standard I_2O message-passing protocol. The RAID module creates one event queue per virtual drive. Each event queue is associated with a single thread of execution. When the RAID module registers a virtual drive as an I_2O object, it specifies which event queue receives messages addressed to that object's TID.

When the IOP receives a request message addressed to one of the virtual drives that is under RAID module control, the request is posted to the associated event queue. The RAID module swaps the initiator and target addresses when replying to the request message. The RAID module creates a new message to reply and hold the request message frame until all associated processing is completed. Then it frees the received message.

MessageFail – When the RAID module receives a request message and determines the request cannot be completed, it creates a message with both the REPLY bit and the FAIL bit set. In normal *reply*, the RAID module uses a default reply template. If only one transaction is replied, it uses a template for the single-transaction; otherwise, it uses a template for multiple transactions.

8.2 Read Operations

ReadBlock – The following illustration shows how a RAID module handles *ReadBlock* messages. It breaks the request into one or more ReadBlock messages targeted at one or more HDMs with destination SG list address(es) to host memory. Individual HDM then issues SCSI *read* commands and controls the data flow from the device to the host.


HDM 0
HDM 1
SCSI Read
SCSI Read
DATA
DATA
Drives
Drives

8.3 Write Operations

WriteBlock – When the RAID module receives a *WriteBlock* request, it may issue one or more *ReadBlock* message(s) to individual HDMs to fetch old data from each device into shared local memory. It also calls the local IOP's API function to DMA data from host memory into the shared local memory. The RAID module then issues a *WriteBlock* message to individual HDMs for transporting data from their local shared memory to the device.