Legal Notices

The information in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Warranty. A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

Restricted Rights Legend. Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 for DOD agencies, and subparagraphs (c) (1) and (c) (2) of the Commercial Computer Software Restricted Rights clause at FAR 52.227-19 for other agencies.

HEWLETT-PACKARD COMPANY 3000 Hanover Street Palo Alto, California 94304 U.S.A.

Use of this manual and flexible disk(s) or tape cartridge(s) supplied for this pack is restricted to this product only. Additional copies of the programs may be made for security and back-up purposes only. Resale of the programs in their present form or with alterations, is expressly prohibited.

Copyright Notices. ©copyright 1983-2002 Hewlett-Packard Company, all rights reserved.

Reproduction, adaptation, or translation of this document without prior written permission is prohibited, except as allowed under the copyright laws.

©copyright 1979, 1980, 1983, 1985-93 Regents of the University of California

This software is based in part on the Forth Berkeley Software Distribution under license from the Regents of the University of California.
©copyright 1985-86, 1988 Massachusetts Institute of Technology.
©copyright 1989-93 The Open Software Foundation, Inc.
©copyright 1986 Digital Equipment Corporation.
©copyright 1990 Motorola, Inc.
©copyright 1989-1991 The University of Maryland
©copyright 1988 Carnegie Mellon University

Trademark Notices. UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

X Window System is a trademark of the Massachusetts Institute of Technology.

MS-DOS and Microsoft are U.S. registered trademarks of Microsoft Corporation.

OSF/Motif is a trademark of the Open Software Foundation, Inc. in the U.S. and other countries.

Solaris is a registered trademark of the Sun Microsystems Corporation, in the U.S and other countries.

For additional information, please contact your local HP field representative.
# Contents

## 1. Porting Solaris SCSI Interface Driver to HP-UX

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>10</td>
</tr>
<tr>
<td>HP-UX SCSI Subsystem Architecture</td>
<td>12</td>
</tr>
<tr>
<td>Architectural Comparisons between HP-UX and Solaris</td>
<td>14</td>
</tr>
<tr>
<td>Some Differences</td>
<td>14</td>
</tr>
<tr>
<td>Data Structures</td>
<td>17</td>
</tr>
<tr>
<td>Interface Driver Entry Points</td>
<td>19</td>
</tr>
<tr>
<td>Misc. Service Calls</td>
<td>22</td>
</tr>
<tr>
<td>Comparison of Driver Development Environments</td>
<td>24</td>
</tr>
<tr>
<td>Driver Installation and Initialization</td>
<td>24</td>
</tr>
<tr>
<td>Probing for Devices on the SCSI Bus</td>
<td>25</td>
</tr>
<tr>
<td>Registering Driver With the SCSI Subsystem</td>
<td>25</td>
</tr>
<tr>
<td>Setting Initial SCSI parameters</td>
<td>28</td>
</tr>
<tr>
<td>DMA Attributes</td>
<td>30</td>
</tr>
<tr>
<td>DMA Setup and Transfer</td>
<td>31</td>
</tr>
<tr>
<td>PCI Local Bus Setup and Access</td>
<td>34</td>
</tr>
<tr>
<td>Interrupt Registering</td>
<td>35</td>
</tr>
<tr>
<td>Dynamic Kernel Memory Allocation and Freeing</td>
<td>37</td>
</tr>
<tr>
<td>Synchronization Primitives</td>
<td>38</td>
</tr>
<tr>
<td>Printing/Logging Driver Messages</td>
<td>38</td>
</tr>
<tr>
<td>Support Utilities</td>
<td>38</td>
</tr>
<tr>
<td>Building Drivers</td>
<td>40</td>
</tr>
<tr>
<td>Debugging Drivers</td>
<td>41</td>
</tr>
<tr>
<td>Solaris to HP-UX Service Mapping</td>
<td>42</td>
</tr>
</tbody>
</table>

## 2. Qlisp Sample Driver Comparison for Qlogic ISP12160A Ultra160 Card

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Driver Sources</td>
<td>46</td>
</tr>
<tr>
<td>Description of the Device</td>
<td>47</td>
</tr>
<tr>
<td>Driver Architecture</td>
<td>49</td>
</tr>
<tr>
<td>Driver Data Structures</td>
<td>50</td>
</tr>
<tr>
<td>Differences In Data Structures</td>
<td>54</td>
</tr>
<tr>
<td>Main I/O Path</td>
<td>56</td>
</tr>
</tbody>
</table>

## Glossary
Preface

This document describes the steps involved in porting a Solaris SCSI HBA driver to HP-UX platforms. Its prime objective is to help driver developers port a working Solaris SCSI HBA driver to HP-UX.

HP-UX and Solaris SCSI subsystem are compared, and Operating System specific differences are discussed. Only issues pertaining to porting a Solaris SCSI interface driver to HP-UX platforms are discussed.

Reference Documentation:

   http://www.hp.com/dspp
   http://www.hp.com/dspp
3. Qlogic Firmware Interface Specification, ISPIxxx Intelligent SCSI Processor, 83120-660-00 G.
   http://qlogic.com
4. Qlogic PCI Interface Specification.
   http://qlogic.com
5. HP-UX 11i Sample Qlisp Driver
   http://hp.com/dspp
6. Writing Device Drivers - Sun Solaris
   http://www.soldc.sun.com
7. The SCSI Tutor: An In-Depth Exploration of the SCSI
1 Porting Solaris SCSI Interface Driver to HP-UX
Introduction

Device driver design and development relies heavily on the underlying Operating System and the driver development environment interfaces the Operating System exports to the drivers. When porting a driver from one OS to another, issues arise because driver development environments differ. When porting a driver, one needs to know that the OSs differ in the specific procedures or interfaces for tasks such as installing a driver, setting up DMA resources, PCI device access, communicating with upper layers, synchronization mechanisms, and interrupt handling.

This document addresses one specific case of porting a Sun Solaris PCI SCSI Host Bus Adapter driver to a HP-UX 11i platform. The primary objective of this document is to give the reader, who is already familiar with the Sun Solaris driver development environment, with information on porting HP-UX driver development environment. For detailed information on HP-UX driver development environment (DDE), refer to the HP-UX 11i Driver Development Guide and HP-UX 11i Driver Development Reference.

The Sun Solaris IO stack, specifically the SCSI subsystem, is very similar to HP-UX SCSI subsystem when viewed at a high level. Although, on closer inspection many gaps and distinctions appear. The HP-UX DDE and SCSI subsystem provides most of the functionality available in the Solaris DDI/DDK and SCSI subsystem. Each OS provides certain features not available on the other OS. This document makes every attempt to highlight such differences, especially when they effect the development of a SCSI interface driver.

The SCSI class driver, also known as the SCSI target driver in Solaris, refers to a driver for a SCSI peripheral, such as a disk or tape. This document primarily deals with the issues involved in porting a Solaris SCSI HBA driver to the HP-UX platform. For more information on how to write a SCSI class driver or other types of drivers please consult the HP-UX 11i Driver Development Guide.

As a sample interface driver this document uses the Qlisp driver, which manages the Qlogic ISP12160A Ultra160 card. The Sample Qlogic ISP12160A driver sources for Sun Solaris 8 and HP-UX 11i are provided for reference as part of this porting guide. As a SCSI interface driver, Qlisp is responsible for taking I/O requests and transporting them to the
specified SCSI peripheral. Furthermore, it captures the responses from the SCSI peripherals and returns them to the SCSI subsystem in an OS specific manner. Furthermore, the interface driver is responsible for managing the interconnect between the initiator and targets, I/O time-outs, and other task management functions such as, bus reset, abort task and bus device reset.

Unless mentioned otherwise HP-UX and Solaris refer to HP-UX 11.11 and Solaris 8.0, respectively. Qlisp is the name used to refer to the sample SCSI interface driver. This document uses terms peripherals and devices interchangeably to refer to SCSI peripherals and devices on a SCSI bus.
HP-UX SCSI Subsystem Architecture

The HP-UX SCSI subsystem can be broadly divided into three layers as shown in figure 1-1.

1. **SCSI class drivers**
   
   SCSI class drivers are also known as SCSI device drivers on HP-UX platform. On Solaris platform, SCSI class drivers are referred to as SCSI target drivers. A SCSI class driver implements one or more device models. For more information on HP-UX SCSI class drivers refer to Chapter 9, Writing SCSI Device Drivers in HP-UX 11i Driver Development Guide.

2. **SCSI services**

   SCSI services provide a common interface to the SCSI device drivers and SCSI interface drivers. SCSI services enable a SCSI class driver to work with many different SCSI Interface drivers, i.e. parallel SCSI drivers. In addition the SCSI services layer provides numerous frequently used services, such as probing for devices on the SCSI bus and common SCSI ioctl's. For more information on SCSI services layer see scsi, scsi_ctl manual pages.

3. **SCSI Interface drivers**

   SCSI interface drivers are also referred to as SCSI Host Bust Adapted (HBA) drivers. They are responsible for managing the underlying hardware. SCSI Interface driver works with the SCSI class drivers using the interface provided by the SCSI services layer. SCSI interface drivers are responsible for the execution of the I/O and returning of status on the completion of an I/O. They are also responsible for managing the interconnect between the initiator and targets on the SCSI bus, and other task management functions such as bus reset, abort task and bus device reset.
WSIO/CDIO is the Device Driver Environment (DDE) in which both SCSI class drivers and interface drivers are developed and executed. For more information on how to write a HP-UX driver please refer to Chapters 2, 3 and 5 of the HP-UX 11i Driver Development Guide.
Architectural Comparisons between HP-UX and Solaris

At a high level, the SCSI subsystems for HP-UX and Solaris are very similar. Both SCSI subsystems are broadly divided into three layers as explained in the previous section. Developer's differences lie in the specific interfaces and data structures.

Some Differences

Here are some of the differences in the two SCSI subsystems:

• **Here are some of the differences and how they are set by the user:**

  In Solaris, a user sets SCSI parameters in a `driver.conf` file which is accessed by the SCSI HBA driver using DDI calls during driver initialization. The Solaris SCSI HBA driver then sets these parameters in a device-specific manner. In HP-UX on the PA architecture, a user sets these parameters using the BCH (Boot Console Handle) menu. The SCSI interface driver can call `SCSI_GET_INITIATOR_PARAMS` macro to read these values. This macro is shipped in the `scsi_params_macro.h` header file which is shipped with the Qlisp sample driver. As in Solaris, the interface driver in HP-UX sets the parameters in a device-specific manner.

• **Modifying SCSI parameters post-initialization:**

  Solaris SCSI subsystem provides entry points to the HBA driver to communicate SCSI parameters. A SCSI HBA driver communicates through its `tran_getcap()` entry point, and provides information to the upper layers regarding the current value of a SCSI parameter. The same driver receives new values for supported SCSI parameters through its `tran_setcap()` entry point. The HP-UX SCSI subsystem does not provide entry points directly into an interface driver for modifying SCSI parameters. In contrast to Solaris, HP-UX SCSI parameters are communicated from the class driver to the interface driver via special I/O through the `scb` structure. A user sends `ioctl`s to the class driver to cause the class driver to send special I/Os to the interface driver. Please refer to Chapter 8 of the HP-UX 11i Driver Development Guide for details on the `scb->flags` field. Also, see the
man on scsi_ctl to get further detail on the ioctlS to set and get SCSI parameters.

• **How a single I/O passes from SCSI services layer to a SCSI Interface driver.**

In both Solaris and HP-UX, SCSI interface driver’s `if_start()` entry point is called when the SCSI subsystem wants the interface driver to process new I/Os. In Solaris, new I/O is passed directly to the interface driver’s `start()` entry point. In HP-UX, the SCSI services layers employ many queues as the new I/O moves down from the upper layers to the interface driver. It is eventually put on a per SCSI bus select queue. When the interface driver’s `start()` entry point is called by the driver it must dequeue the requests from this queue and begin processing them. If the hardware is at its maximum queue depth the driver may choose to leave the remaining I/O requests on the select queue or even enqueue an I/O request that had been previously dequeued. HP-UX also provides a kernel tunable, `MAX_Q_DEPTH`, to control the flow of I/O to each SCSI bus.HP-UX SCSI services provides a unique I/O identifier.

• **HP-UX SCSI services provide a unique I/O identifier.**

In HP-UX SCSI related information for each I/O is stored in the SCSI control block structure, `scb`. The `scb->tag` field is set to a unique value by the SCSI subsystem before the I/O is passed to the SCSI interface driver. The uniqueness is guaranteed across a SCSI bus and not just a SCSI target or SCSI LUN. An interface driver is not required to use this field, it may choose to do so in order to uniquely identify each I/O. On Solaris, the driver must implement such a scheme and ensure its uniqueness. The range of this value is also adjustable by the driver. This range is controlled by setting `scsi_ifsw->if_max_tag` field to the maximum value. Furthermore, the value assigned to `scsi_ifsw->if_max_tag+1` is the maximum number of active I/Os on a SCSI bus. This is ensured by the SCSI subsystem.

• **How and Where DMA resources are allocated, synchronized and released.**

In Solaris, all the DMA resources for an I/O are allocated in HBA driver entry point `tran_init_pkt()`. This entry point is called by the Solaris SCSI subsystem before it calls the HBA driver `start()` entry point. In a similar fashion, Solaris requires the HBA driver to provide separate entry points for syncing and releasing DMA
resources. In contrast, HP-UX SCSI subsystem places less restriction on an interface driver. The HP-UX SCSI subsystem enqueues new I/OSs on a per bus select queue and calls the interface driver `start()` entry point. It is up to the interface driver on how it manages its allocation, synchronization, and releasing of its DMA resources.
Porting Solaris SCSI Interface Driver to HP-UX
Architectural Comparisons between HP-UX and Solaris

Data Structures

The SCSI subsystem in Solaris and HP-UX export data structures as an interface. A brief description of these data structures is listed below. A more detailed explanation is provided in a later section.

<table>
<thead>
<tr>
<th>Table 1-1</th>
<th>Solaris SCSI Subsystem Data Structures</th>
</tr>
</thead>
<tbody>
<tr>
<td>scsi_device</td>
<td>One for each logical unit attached to the system.</td>
</tr>
<tr>
<td>scsi_pkt</td>
<td>Encapsulates a SCSI I/O. Allocated and released by the driver, when asked to do so by the SCSI subsystem.</td>
</tr>
<tr>
<td>scsi_hba_tran</td>
<td>One for each instance of a SCSI HBA. Driver usually hangs a driver specific, per instance data structure off of tran_hba_private field.</td>
</tr>
<tr>
<td>scsi_address</td>
<td>Represents the address of a SCSI device.</td>
</tr>
</tbody>
</table>

In Solaris, the HBA driver registers itself with the SCSI services in its attach() routine by calling scsi_hba_tran_alloc() and scsi_hba_attach_setup() functions. The HBA driver calls scsi_hba_tran_alloc() to allocate the scsi_hba_tran structure, this is a per instance structure. The HBA driver must fill in its entry point information in the scsi_hba_tran structure and sets scsi_hba_tran->tran_hba_private to point to a driver specific structure before calling scsi_hba_attach_setup(). For HP-UX, the per instance structure is struct isc_table_type; in this structure the driver uses the isc_table_type->if_isc field to point to a driver specific structure. It is not allocated by the driver, rather passed to the driver as a parameter in its PCI attach routine. To register the interface driver entry points with the SCSI subsystem in HP-UX the drive must also allocate and fill the SCSI interface switch structure, struct scsi_ifsw.

In Solaris, each SCSI LUN is represented by scsi_device structure and the address related information for each SCSI peripheral is stored in the scsi_address structure. The HP-UX SCSI subsystem exports the scsi_lun, scsi_tgt, and scsi_bus structure to completely represent a SCSI LUN, target, and bus connection.
Porting Solaris SCSI Interface Driver to HP-UX

Architectural Comparisons between HP-UX and Solaris

In Solaris, the SCSI HBA driver is sent an I/O in a buf structure from where the driver transfers all the required information to the scsi_pkt structure. The SCSI HBA driver never accesses the buf structure thereafter. In HP-UX, similar to Solaris, information regarding an I/O is passed to the SCSI interface driver in a buf structure; SCSI specific information is kept in a separate control block structure. However, unlike Solaris, HP-UX SCSI subsystems enqueue I/Os on a per SCSI bus select queue, which is a queue of buf structures. When the SCSI interface driver is called it dequeues I/Os from its select queue before processing them.

**Table 1-2** HP-UX SCSI Subsystem Data Structures

<table>
<thead>
<tr>
<th>Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct buf</td>
<td>Main structure that contains information regarding an I/O. This structure is shared between the Filesystem layer to the Interface driver layer.</td>
</tr>
<tr>
<td>struct scb</td>
<td>Contains addition information regarding an I/O. Mostly SCSI specific information.</td>
</tr>
<tr>
<td>struct scsi_lun</td>
<td>Allocated for each SCSI LUN. This structure is allocated and initialized when the LUN is first opened, and is deallocated on the last close by the SCSI services. This is owned by the SCSI services.</td>
</tr>
<tr>
<td>struct scsi_tgt</td>
<td>Allocated for each SCSI target. This structure is allocated and initialized when a LUN connected to the target is first opened, and is deallocated on the last close of a LUN connected to the target by the SCSI services. This is owned by the SCSI services.</td>
</tr>
<tr>
<td>struct scsi_bus</td>
<td>Allocated for each SCSI bus. Allocated and initialized when a LUN connected to the target on the SCSI bus is first opened, and is deallocated on the last close of a LUN connected to the target on the SCSI bus by the SCSI services. This is owned by the SCSI services.</td>
</tr>
</tbody>
</table>
Porting Solaris SCSI Interface Driver to HP-UX
Architectural Comparisons between HP-UX and Solaris

Table 1-2  HP-UX SCSI Subsystem Data Structures

| struct scsi_ifsw       | This structure defines SCSI interface driver entry points and parameters as required by the SCSI services. |

Following table shows a mapping of SCSI subsystem data structures between Solaris and HP-UX.

Table 1-3  Solaris to HP-UX Mapping of Data Structures

| scsi_device and scsi_address | struct scsi_lun, struct scsi_tgt, struct scsi_bus |
| scsi_pkt, buf                | struct buf, struct scb                                |
| scsi_hba_tran                | struct isc_table_type, struct scsi_ifsw              |

Interface Driver Entry Points

Following section matches Solaris HBA driver entry points to HP-UX Interface driver. In the absence of a proper fit, comments describe the closest fit.

Table 1-4  Solaris to HP-UX Mapping of SCSI Interface Driver Entry Points

| tran_tgt_init() | Called when system attaching a target device instance. In HP-UX, if_open() entry point is called on all logical unit opens. It is an optional entry-point. |
| tran_tgt_probe() | Called during probing if the Solaris HBA is using a custom probe routine. In HP-UX, an interface driver can register its custom probe routine in its driver_install() by calling wsio_register_addr_probe(). See section Device Probing Routine. |
### Architectural Comparisons between HP-UX and Solaris

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tran_tgt_free()</code></td>
<td>Called when system detaching a target device instance.</td>
</tr>
<tr>
<td></td>
<td>In HP-UX, <code>if_close()</code> entry point is called on all logical unit closes. It is an optional entry point.</td>
</tr>
<tr>
<td><code>tran_start()</code></td>
<td>Called to initiate an I/O, which is passed as a parameter.</td>
</tr>
<tr>
<td></td>
<td>In HP-UX, <code>if_start()</code> is called when there are I/Os to be processed. Instead of passing I/O as a parameter to <code>if_start()</code>, it is enqueued on a per SCSI bus select queue. The interface driver dequeue I/Os from the select queue using <code>scsi_dequeue()</code>.</td>
</tr>
<tr>
<td><code>tran_reset()</code></td>
<td>Called to request bus reset or bus device reset.</td>
</tr>
<tr>
<td></td>
<td>In HP-UX, <code>if_reset_bus()</code> is called to initiate a Bus reset. For bus device resets, <code>if_bdr()</code> entry point is called.</td>
</tr>
<tr>
<td><code>tran_abort()</code></td>
<td>Called to send a SCSI ABORT message.</td>
</tr>
<tr>
<td></td>
<td>In HP-UX, <code>if_abort()</code> is called to send a SCSI ABORT message to the indicated logical unit. Unlike Solaris in HP-UX, a specific I/O cannot be aborted using this entry point.</td>
</tr>
<tr>
<td><code>tran_getcap()</code></td>
<td>No proper match.</td>
</tr>
<tr>
<td><code>tran_setcap()</code></td>
<td>No proper match.</td>
</tr>
<tr>
<td><code>tran_init_pkt()</code></td>
<td>Called to initialize an I/O.</td>
</tr>
<tr>
<td></td>
<td>Unlike Solaris, HP-UX does not treat resource allocation as a separate part of I/O processing. HP-UX calls <code>if_start()</code>, where the interface driver can move all of its code from <code>tran_init_pkt()</code>.</td>
</tr>
</tbody>
</table>
### Table 1-4 Solaris to HP-UX Mapping of SCSI Interface Driver Entry Points

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tran_destroy_pkt()</td>
<td>Called to release resources allocated for an I/O.</td>
</tr>
<tr>
<td></td>
<td>Unlike Solaris, HP-UX does not treat resource deallocation as a separate part of I/O processing. An interface driver must clean up before it calls the callback function, scsi_cbfn().</td>
</tr>
<tr>
<td>tran_dmafree()</td>
<td>Called to release only the DMA resources.</td>
</tr>
<tr>
<td></td>
<td>Unlike Solaris, HP-UX does not treat resource deallocation as a separate part of I/O processing. An interface driver must clean up before it calls the callback function, scsi_cbfn().</td>
</tr>
<tr>
<td>tran_sync_pkt()</td>
<td>Called to synchronize an I/O packet.</td>
</tr>
<tr>
<td></td>
<td>HP-UX driver must call <code>dma_sync_io()</code> as per guidelines described in detail in HP-UX 11i DDG. No separate entry point are provided to engage the interface driver in synchronizing data.</td>
</tr>
<tr>
<td>tran_reset_notify()</td>
<td>Called to register/unregistered callback functions with HBA driver, which the HBA driver must call when a bus reset occurs.</td>
</tr>
<tr>
<td></td>
<td>No proper match.</td>
</tr>
<tr>
<td>tran_quiesce()</td>
<td>Called to quiesce the SCSI bus.</td>
</tr>
<tr>
<td></td>
<td>No proper match.</td>
</tr>
<tr>
<td>tran_unquiesce()</td>
<td>Called to unquiesce the SCSI bus.</td>
</tr>
<tr>
<td></td>
<td>No proper match.</td>
</tr>
</tbody>
</table>
Porting Solaris SCSI Interface Driver to HP-UX
Architectural Comparisons between HP-UX and Solaris

Misc. Service Calls

The following is a list of routines provided by HP-UX SCSI services. They are specific to HP-UX SCSI subsystem and do not have matching routine in Solaris.

1. **SCSI Subsystem Callback**

   In Solaris, a HBA driver calls `scsi_pkt->pkt_comp` when it is done with an I/O. In HP-UX, when the interface driver finishes with an I/O, it returns the I/O to the SCSI subsystem by calling `scsi_cbfn()` with a pointer to the `buf` structure as its sole argument. In HP-UX, the driver must not access the `buf`, `scb`, or `scb->if_scb` structures once it calls `scsi_cbfn()`.

   ```c
   void scsi_cbfn(struct buf *bp)
   ```

2. **SCSI Subsystem Queueing Functions**

   The SCSI subsystem sends I/O to an interface driver by queueing it on a per bus select queue. It provides the following interfaces for accessing the new I/Os on the per bus select queue. It is important to note that the driver must acquire and release a SCSI bus lock before and after calling one of the queue routines.

   ```c
   #define TAIL 0
   #define HEAD 1
   void scsi_enqueue(struct buf **qp, struct *bp, int where);
   struct buf* scsi_dequeue(struct buf **qp, int where);
   struct buf* scsi_dequeue_bp(struct buf **qp, struct buf *bp);
   ```

3. **Open Device Tree Access Functions**

   Functions for acquiring pointers to the open tree data structures from a device.

   ```c
   struct scsi_bus* m_scsi_bus(dev_t dev)
   struct scsi_tgt* m_scsi_tgt(dev_t dev)
   struct scsi_lun* m_scsi_lun(dev_t dev)
   struct isc_table_type* m_scsi_isc(dev_t dev)
   ```
4. Macro Functions

Macros are provided by the SCSI subsystem for the convenience of the interface driver. The return bus instance, target id and lun id given a dev_t.

```c
m_bus_id(dev_t dev);
m_tgt_id(dev_t dev);
m_lun_id(dev_t dev);
```

For details refer to Chapter 8 of the HP-UX 11i Driver Development Guide.
Comparison of Driver Development Environments

This section compares functionality provided by HP-UX DDE and Solaris DDI/DDK.

Driver Installation and Initialization

In Solaris, the HBA driver execution starts in the \texttt{\_init()} routine; this is where the driver registers itself with the system. For HP-UX, the interface driver execution begins in the \texttt{driver\_install()} routine, which must be named \texttt{qlisp\_install()} for the Qlisp driver. It is in the \texttt{driver\_install()} routine that the driver links its attach routine to the attach chain and registers itself with WSIO CDIO.

In Solaris, the HBA driver performs all of its initialization in its attach routine. In HP-UX, in the driver attach routine, the driver claims a device if the passed PCI vendor id and device id match the driver’s own list of supported devices. After claiming the device by calling \texttt{isc\_claim()}, the driver calls the \texttt{CONNECT\_INIT\_ROUTINE} macro to arrange for the system to call the driver’s \texttt{init} routine. In HP-UX, the driver performs all of its device initialization in its \texttt{init} routine. The driver \texttt{init} routine is called by the system after all the attach chains have been run.

Following is an excerpt from the Qlisp interface driver attach routine. The interface driver claims the hardware and connects its \texttt{init} routine if it supports the newly discovered hardware.

```c
int qlisp\_pci\_attach(uint32\_t id, struct isc\_table\_type *isc) {
    ...
    /* Check if the passed PCI vendor id and device id match */
    /* the list of supported devices. */
    if (QLISP\_ID\_SUPPORTED(id)) {
        /* If supported, claim the isc entry and connect the init routine */

        isc\_claim(isc, &qlisp\_wsio\_info);
        CONNECT\_INIT\_ROUTINE(isc, qlisp\_init\_master);
    }
```
/* Call the next routine in the attach chain */
return (*((saved_qlisp_pci_attach)(id, isc));

Probing for Devices on the SCSI Bus

Identifying all the devices on a SCSI bus is a critical part of SCSI initialization. This is accomplished by probing for devices on the SCSI bus. Probing for devices on a SCSI bus can be performed either by a driver probe routine or the SCSI services probe routine. The probing routine provided by HP-UX SCSI services is `parallel_scsi_probe()`. It is similar to the Solaris probing routine `scsi_hba_probe()`. It is the driver's responsibility to register a probing routine.

The following code snippet shows a driver registering the HP-UX SCSI services probe routine `parallel_scsi_probe()`.

```c
int qlisp_install (void)
{
  ...
  /* Register probe routine. Use the one provided by the system */
  wsio_register_addr_probe(parallel_scsi_probe, QLISP_NAME) ;
  ...
}
```

For in depth information on the probe routine and how to write a custom probe routine please refer to Chapter 5 of the HP-UX 11i Driver Development Guide.

Registering Driver With the SCSI Subsystem

Both Solaris and HP-UX require SCSI drivers to register with the SCSI subsystem. This is to make the SCSI subsystem aware of the interface driver.

On the Solaris platform this is done in the driver attach routine by calling the pair of routines `scsi_hba_tran_alloc()` and `scsi_hba_attach_setup()`. Main data structure involved here is `scsi_hba_tran_t`, which is allocated and owned by the SCSI subsystem. One of its fields, `tran_hba_private`, is typically set to point to a driver...
specific structure. The SCSI entry point, pointers are also contained in the `scsi_hba_tran_t` structure. After filling the `scsi_hba_tran_t` structure, the HBA driver passes it to the Solaris SCSI services by `scsi_hba_attach_setup()`.

In HP-UX an interface driver registers itself with the SCSI subsystem in its `init` routine by setting an element in the global `scsi_isc[]` array to point to interface driver's `isc` instance. The index in the global array `scsi_isc[]` is the WSIO instance number of the driver, which can be retrieved by calling `wsio_isc_to_instance()`. For HP-UX the SCSI entry points routines are contained in the SCSI interface driver switch structure, `struct scsi_ifsw`. As a part of registering itself with the SCSI subsystem the driver must allocate and initiate the `scsi_ifsw` structure. Thereafter, the driver must set `isc->ifsw` to the address of the `scsi_ifsw` structure. This is done in the drivers `init` function as shown in the code snippet below. For a detailed explanation of `struct scsi_ifsw` please see Chapter 8 of the HP-UX 11i/11.0 Driver Development Guide.

In HP-UX sample driver the `init` routine is `qlisp_init_master()`, which also calls `qlisp_init_common()`.

```c
/*
 * Driver's init routine.
 * Does all the driver initialization.
 */
static int
qlisp_init_master(struct isc_table_type *isc)
{
  qlisp_isc_t *lisc = isc->if_isc;
  qlisp_shared_isc_t *share = lisc->share;

  /* Save the driver's instance for later use.
   */
  lisc->wsio_inst = wsio_isc_to_instance(isc, NULL);

  ...

  if (qlisp_init_common(isc) != QLISP_OK)
    goto recover_2;

  ...
}

/*
 * Does common part of the SCSI initialization.
 */
```
Porting Solaris SCSI Interface Driver to HP-UX
Comparison of Driver Development Environments

static int qlisp_init_common(struct isc_table_type *isc)
{
    qlisp_isc_t*lisc=isc->if_isc;
    qlisp_shared_isc_t*share=lisc->share;

    ...}

/*Register driver with HP-UX SCSI subsystem. */
scsi_isc[isc->wsio_inst] = isc;

... /*Registration driver entry points with the SCSI subsystems by *allocating and initializing *the SCSI switch structure, struct scsi_ifsw. */
if(qlisp_int_ifsw) !=QLISP_OK)
    return (QLISP_ERR);

... return (QLISP_OK);
}

/* Fill in the interface data for SCSI services in scsi_ifsw. */
static int qlisp_init_ifsw(struct isc_table_type *isc)
{
    qlisp_isc_t *lisc = isc->if_isc;
    qlisp_shared_isc_t *share = lisc->share;
    struct scsi_ifsw *qlisp_ifsw;

    /* * Allocate memory for scsi_ifsw used by Services. */
    qlisp_ifsw = (struct scsi_ifsw *)kmalloc(sizeof(struct scsi_ifsw),
        M_IHV, M_WAITOK);
    (void) bzero((caddr_t)qlisp_ifsw, sizeof(struct scsi_ifsw));

    /* Fill in the entry points and other information. */
Porting Solaris SCSI Interface Driver to HP-UX
Comparison of Driver Development Environments

qlisp_ifsw->if_flags = IF_BUS_TAGS | IF_B2_LIST;
qlisp_ifsw->if_max_tag = share->req_max_iocb - 1;
qlisp_ifsw->if_scb_size = sizeof(qlisp_scb_t);
qlisp_ifsw->if_lun_size = 0;
qlisp_ifsw->if_tgt_size = 0;
qlisp_ifsw->if_bus_size = sizeof(qlisp_bus_t);
qlisp_ifsw->if_open = qlisp_if_lun_open;
qlisp_ifsw->if_start = qlisp_if_start;
qlisp_ifsw->if_abort = qlisp_if_abort;
qlisp_ifsw->if_bdr = qlisp_if_bdr;
qlisp_ifsw->if_reset_bus = qlisp_if_reset_bus;
qlisp_ifsw->if_max_io_size = 0;
qlisp_ifsw->if_beg_align = 0;
qlisp_ifsw->if_end_align = 0;

/* Complete SCSI subsystem registration. */
isc->ifsw = (caddr_t)qlisp_ifsw;
return (QLISP_OK);

Setting Initial SCSI parameters

In Solaris a user sets SCSI parameters for a device in the driver.conf file. As the driver is going through its initialization it reads this information using standard DDI calls. The SCSI driver then sets these parameters in a device specific manner. In HP-UX, a user sets these parameters using the BCH menu. The SCSI driver reads the values using the SCSI_GET_INITIATOR_PARAMS macro. This macro returns two SCSI parameters, Initiator Id, and the SCSI transfer rater. For details please refer to chapter 5 of the HP-UX 11i Driver Development Guide.

int qlisp_init_common(struct isc_table_type *isc)
{
  /* Get pointer to driver specific, per instance structure */
  qlisp_isc_t *lisc = isc->if_isc;
  qlisp_shared_isc_t *share = lisc->share;

  int initiator_id; /* HBA ID, can be set in BCH or card BIOS menu */
  int sdtr_period; /* HBA speed, can be set in BCH or card BIOS menu */
.../* Get the SCSI initiator parameters.  
* The required information can be obtained by calling the macro  
* SCSI_GET_INITIATOR_PARMS  
*/
SCSI_GET_INITIATOR_PARMS(isc, initiator_id, sdtr_period);

/* No valid SCSI rate found. Set according to device capabilities */
if (sdtr_period == -1)  {
  sdtr_period = 9; /* For Ultra3 */
}

/* Set the corresponding isc fields to the SCSI parameters obtained */
isc->my_address = initiator_id;
isc->bus_min_sdtr_period = sdtr_period;

/* Set req offset according to the device capability */
isc->bus_max_reqack_offset = 12;

/* Set the max bus width */
isc->bus_min_sdtr_period = sdtr_period;

/* Now set these parameters on the HBA in a device specific manner */
...

As in Solaris, the interface driver in HP-UX sets the parameters in a  
device specific manner.

Here is the mapping for setting these options on HP-UX systems.

**Table 1-5**  
**Mapping Solaris SCSI parameters to HP-UX**

| **SCSI_OPTION_DR** | Enables disconnect/reconnect.  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| SIOC_IO ioctl, set flags field to  
| SCTL_NO_DISC. This causes the SCSI device driver to set SCB_NO_DISC flag in the  
| scb->flags field. |

| **SCSI_OPTIONS_SYNC** | Enables synchronous transfer capability.  
|----------------------|-----------------------------------------------|
|                      | SIOC_IO ioctl, set flags field to  
|                      | SCTL_INIT_SDTR. This causes the SCSI device driver to set SCB_SDTR flag in the  
|                      | scb->flags field. |
Porting Solaris SCSI Interface Driver to HP-UX  
Comparison of Driver Development Environments

<table>
<thead>
<tr>
<th>Table 1-5</th>
<th>Mapping Solaris SCSI parameters to HP-UX</th>
</tr>
</thead>
</table>
| **SCSI_OPTIONS_LINK** | Enables link command support.  
- Not supported in HP-UX. |
| **SCSI_OPTIONS_PARITY** | Enables parity support.  
- Can’t be modified by a user. |
| **SCSI_OPTIONS_TAG** | Enables tagged queuing support.  
- Can’t be modified by a user. |
| **SCSI_OPTIONS_FASTxx** | Enables FASTxx SCSI support.  
- This is similar to transfer rate in HP-UX.  
This can be set by the user in BCH menu. It is read by calling  
SCSI_GET_INITIATOR_PARM macro. |
| **SCSI_OPTIONS_WIDE** | Enables WIDE SCSI.  
SIOC_IO ioctl set flags field to  
SCTL_INIT_WDTR. This cause the SCSI device driver to set  
SCB_WDTR flag in the scb->flags field. |
| **SCSI-INITIATOR-ID** | Initiator ID.  
- In HP-UX, this is set by the user in BCH menu. It is accessed by calling  
SCSI_GET_INITIATOR_PARM macro. |
| **SCSI-RESET DELAY** | SCSI bus or device reset recovery time in milliseconds.  
Not user configurable. |

**DMA Attributes**

HP-UX provides various WSIO services for DMA and PCI local bus access. The WSIO DMA services accept certain flags to optimize the resource allocation policies for inbound and outbound data paths. When porting a Solaris SCSI Interface driver to HP-UX, these additional features of HP-UX should be considered to improve the driver performance and functionality.
Porting Solaris SCSI Interface Driver to HP-UX
Comparison of Driver Development Environments

In Solaris, a HBA driver uses the `ddi_dma_attr_t` structure to fill in DMA engine specific attributes necessary to allocate DMA resources for a device. It is typically passed as a parameter in ensuing calls to allocate DMA resources.

In HP-UX, an interface driver uses the `wsio_dma_attribute_t` structure to fill in DMA related information. This structure is then passed as an argument in calls to `wsio_set_dma_attributes()`, `wsio_dma_set_device_attributes()`, and other DMA mapping routines.

**DMA Setup and Transfer**

HP-UX provides extensive DMA setup and transfer services to keep the driver as isolated from the underlying platform as possible. Typically an interface driver allocates one or more DMA handles in its driver `init` routine by calling `wsio_allocate_dma_handle()`. It would then use DMA services routines, `wsio_dma_set_device_attributes()` or `wsio_set_dma_attributes()` to specify attributes of the type of DMA it will use the object for. A driver may configure one handle for large packet DMA and another for continuous DMA involving small buffers. Later, when setting up a DMA transfer the driver passes the handle to one of the DMA mapping routines. HP-HX DMA services, like Solaris, allow the drivers to specify callback functions for allocating resources.

For all the above mentioned services that allocate DMA resources, there are corresponding WSIO services to free the resources. For example, a handle is released by calling `wsio_free_dma_handle()`.

HP-UX DDE provides `dma_sync_IO()` to synchronize the process caches with the device view of memory. This is a requirement on non-cache coherent or semi-coherent platform. However, `dma_sync_IO()` does not do anything on a fully cache-coherent platform. The driver developers must use this function to write platform independent drivers. Also, refer to the HP-UX 11i Driver Development Guide for a detailed explanation.

The request and response queues are allocated, setup and synchronized in a conceptually similar fashion in both the sample drivers. However, there are differences in the specific DMA calls.

In Solaris, for each DMA resources a DMA handle is first allocated by calling `ddi_dma_alloc_handle()`. Memory is allocated for this handle by calling `ddi_dma_mem_alloc()`. Finally, `ddi_dma_addr_bind_handle()` is called to allocated the rest of DMA resources so that the device can finally perform a DMA transfer.
Shown below is DMA allocation code from the Solaris sample driver.

```c
/*
 * qlisp_init_alloc_queues -
 *    Allocate and map request / response queues for card
 */
int
qlisp_init_alloc_queues(qlisp_isc_t *lisc)
{
    qlisp_shared_isc_t *share = lisc->share;
    ddi_dma_cookie_t cookie;
    uint_t count;
    size_t qlen;

    ... 

    qlen = (share->qlisp_queue_sz) * sizeof(qlisp_q_ent_t);
    share->req_queue_p = share->rsp_queue_p = NULL;

    /* Allocate the DMA handles for request and response queues */
    if(ddi_dma_alloc_handle(lisc->devp, &qlisp_dma_attr, DDI_DMA_SLEEP,
                         NULL, &share->req_queue_DMA_handle) != DDI_SUCCESS) {
        return (QLISP_ERR);
    }
    if (ddi_dma_mem_alloc(share->req_queue_DMA_handle, qlen,
                         &be_dev_attr, DDI_DMA_CONSISTENT, DDI_DMA_SLEEP,
                         NULL, (caddr_t *)&share->req_queue_p, &reqlen,
                         &share->req_queue_ACC_handle) != DDI_SUCCESS) {
        return (QLISP_ERR);
    }
    if (ddi_dma_addr_bind_handle(share->req_queue_DMA_handle,
                              NULL, (caddr_t)share->req_queue_p, qlen,
                              DDI_DMA_RDWR|DDI_DMA_CONSISTENT,
                              DDI_DMA_SLEEP, NULL, &cookie, &count) != DDI_DMA_MAPPED) {
        return (QLISP_ERR);
    }

    /* Set the device register to this value */
    share->dma_req_queue_p = cookie.dmac_address;

    ... 
```
Shown below is a code snippet from the HP-UX Qlisp sample driver.

```c
/*
 * qlisp_init_alloc_queues - 
 *   Allocate and map request / response queues for card
 */
qlisp_init_alloc_queues(struct isc_table_type *isc)
{
    qlisp_isc_t *lisc = isc->if_isc;
    qlisp_shared_isc_t *share = lisc->share;

    /* Calculate the size of the req & resp queues */
    share->qlisp_queue_sz = MIN(qlisp_queue_sz, QLISP_MAX_TAGS);

    /* Allocate the DMA handles for request and response queues */
    share->req_queue_DMA_handle = wsio_allocate_dma_handle(isc);
    share->rsp_queue_DMA_handle = wsio_allocate_dma_handle(isc);

    /* Check if DMA handles allocated properly */
    if ((share->req_queue_DMA_handle == NULL) ||
        (share->rsp_queue_DMA_handle == NULL)) {
        msg_printf("qlisp - DMA handles are not allocated\n");
        return (QLISP_ERR);
    }

    /* Allocate Continuous DMA buffer */
    if (wsio_allocate_shared_mem(isc, share->req_queue_DMA_handle,
        (share->qlisp_queue_sz) * sizeof(qlisp_q_ent_t), &tmp_addr,
        (caddr_t *)&share->req_queue_p, /*lint !e740*/
        WSIO_IO_SHMEM_ALIGN_ON_SIZE) != WSIO_MAP_OK)
    {
        QLISP_MSG("qlisp: allocate_shared_memory req_queue_p failed!\n");
        return (QLISP_ERR);
    }

    /* Synchronize the DMA buffer before using it. */
    DMA_SYNC_IO(KERNELSPACE, share->req_queue_p,
        ((share->qlisp_queue_sz) * sizeof(qlisp_q_ent_t)),
```

Porting Solaris SCSI Interface Driver to HP-UX
Comparison of Driver Development Environments

```c
IO_WRITE | IO_NO_SYNC); /*lint !e921*/

return (QLISP_OK);
}
```

**PCI Local Bus Setup and Access**

In Solaris, the driver access the PCI configuration space of the device by calling:

- `pci_read_cfg_uintXX_isc(isc_table_type, offset, &value)`
- `pci_config_putXX(ddi_acc_handle_t, offset, value)`

Where XX can be 8, 16, 32 or 64. For a Solaris driver must call `pci_config_setup()` to setup PCI configuration address space, before it can access it.

In a HP-UX WSIO driver, access to the PCI configuration space of the device is made by calling:

- `pci_read_cfg_uintXX_isc(isc_table_type, offset, &value)`
- `pci_write_cfg_uintXX_isc(isc_table_type, offset, value)`

Where XX can be 8, 16, or 32. Here are samples of these services:

In an HP-UX WSIO interface driver, PCI first base address register is mapped and the address is stored in the `isc->if_reg_ptr` field. If there are more than one PCI BAR in the device that needs to be mapped, then the driver can call kernel service `map_mem_to_host()` to map the physical bus address to the host virtual address. The device's memory mapped I/O space can be accessed through:

- `READ_REG_UINTxx_ISC(isc_table_type, address from, address to)`
- `WRITE_REG_UINTxx_ISC(isc_table_type, address to, value)`

**For example:**

```c
{
    struct isc_table_type *isc = lisc->isc;
    uint32_t *addr;
    ...
    READ_REG_UINT32_ISC(isc, addr, (uint32_t*)data);
    ...
    WRITE_REG_UINT32_ISC(isc, addr, data)
    ...
}
```
Interrupt Registering

Interrupt registering is a three step process:

1. Allocate an interrupt object.
   
   WSIO provides `wsio_intr_alloc()` to allocate an interrupt object.

2. Setup the interrupt object.
   
   Once an interrupt object has been allocated, a driver calls `wsio_set_irq_line()` to setup a line based interrupt object.

3. Activate the interrupt object.
   
   Finally, drive calls `wsio_intr_activate()` to activate the interrupt object. The interrupt object must be activated before the system will call the driver’s ISR.

In the Solaris sample driver, the driver interrupt service routine `qlisp_intr()` is registered in `qlisp_init_interrupts()` during the call to `qlisp_attach()`, which is the driver attach entry point.

```c
int qlisp_init_interrupts(qlisp_isc_t *lisc)
{
    ...

    /*
     * get cookie so we can initialize the mutexes
     */
    if (ddi_get_iblock_cookie(lisc->devp, (u_int)0, &share->iblock_cookie) != DDI_SUCCESS) {
        QLISP_LOG("qlisp_init_interrupts: Unable to allocate iblock cookie");
        return (QLISP_ERR);
    }

    /*
     * Intialize mutexes before connecting ISR to IRQ line.
     */
    mutex_init(&share->mp_lock, "qlisp interrupt mutex",
               MUTEX_DRIVER, (void *) share->iblock_cookie);

    /* Add interrupt handler */
    if (ddi_add_intr(lisc->devp, (u_int)0, &share->iblock_cookie,
                     (ddi_idevice_cookie_t *)0, qlisp_intr, (caddr_t)lisc)) {
        QLISP_LOG("qlisp_init_interrupts: unable to add ISR");
        return (QLISP_ERR);
    }

    ...
```

Chapter 1
/* Now turn on the interrupts on the device */

...

In the HP-UX sample driver, the driver interrupt service routine \texttt{qlisp_isr()} is also registered in \texttt{qlisp_init_interrupts()} but it is called by the driver \texttt{init} function and not the \texttt{attach} function.
int
qlisp_init_interruptions(struct isc_table_type *isc)
{
    ...

    /* Allocate spinlock */
    if ((share->mp_lock = alloc_spinlock(C700_ID_LOCK_ORDER,
        QLISP_MP_LOCK_NAME)) == NULL) {
        QLISP_MSG("qlisp: Unable to allocate spinlock.\n");
        return (QLISP_ERR);
    }

    /* Allocate interrupt object */
    if (wsio_intr_alloc(isc, (wsio_drv_isr_t)qlisp_isr,
        (uintptr_t)isc, (uint64_t)0, &share->iobj) != WSIO_OK) {
        QLISP_MSG("qlisp(%d): wsio_intr_alloc failed!\n", lisc->wsio_inst);
        return (QLISP_ERR);
    }

    /* Set Interrupt parameters */
    if (wsio_intr_set_irq_line(isc, share->iobj, (intptr_t)WSIO_IRQ_LINE_AUTO,
        (uint64_t)0L) != WSIO_OK) {
        QLISP_MSG("qlisp(%d): wsio_intr_set_irq_line failed!\n", lisc->wsio_inst);
        return (QLISP_ERR);
    }

    /* Activate Interrupt line */
    if (wsio_intr_activate(isc, share->iobj) != WSIO_OK) {
        QLISP_MSG("qlisp(%d): wsio_intr_activate failed!\n", lisc->wsio_inst);
        return (QLISP_ERR);
    }

    /* Enable interrupts on the device */

    ...
}

In HP-UX, an interrupt handler can be unregistered by calling
wsio_intr_deactivate() or wsio_intr_deactivate_nowait(), and
the interrupt object released by calling wsio_intr_free(). For more
detailed information refer to HP-UX 11i Driver Development Guide.

Dynamic Kernel Memory Allocation and Freeing

In Solaris, dynamic kernel memory is allocated in the driver by calling
kmem_alloc() or kmem_zalloc(). Drive must call kmem_free() in
Porting Solaris SCSI Interface Driver to HP-UX  
Comparison of Driver Development Environments

order to release the memory.  
In HP-UX, the way to allocate memory is `kmalloc()`. The driver must call `kfree()` in order to release the previously allocated memory.

**Synchronization Primitives**

In Solaris, mutexes are the preferred synchronization primitives for device drivers. The mutex interface includes `mutex_init`, `mutex_destroy`, `mutex_enter`, `mutex_tryenter`, `mutex_exit`, and `mutex_owned`. For a detailed discussion please consult the Solaris DDG.

HP-UX offers beta semaphores and spinlocks as synchronization mechanisms. Spinlocks are widely used in drivers. The beta semaphore interface includes; `b_init_sema`, `b_cpsema`, `b_owns_sema`, `b_psema`, and `b_vsema`. The spinlock interface includes `alloc_spinlock`, `dealloc_spinlock`, `spinlock`, `spinunlock`, `owns_spinlock`, `cspinlock`, and `initlock`.

For a complete discussion of HP-UX synchronization primitives, see Chapter 4 on Multiprocessing in the HP-UX 11i Driver Development Guide.

The spinlock function `get_sleep_lock()` needs special mention; it synchronizes between sleep and wakeup and protects the critical section between the time it is called and the time the calling thread calls sleep. The spinlock is released as part of the sleep call.

**Printing/Logging Driver Messages**

In Solaris drivers, messages are logged using the kernel service `cmn_err()`. Similarly, in HP-UX, messages can be logged using kernel service `msg_printf()`.

**Support Utilities**

Solaris, provides `prtconf` to check peripherals connected to the system. Another utility `modinfo`, provides the current list of loaded driver modules.

HP-UX offers `ioscan` to achieve the result of both the utilities mentioned above. The iscan utility scans the system I/O tree to determine currently installed hardware. In addition, it also provides various flags in order to filter the scan and thus the output to certain
type of device.

$ioscan -fnC disk

<table>
<thead>
<tr>
<th>Class</th>
<th>I</th>
<th>H/W Path</th>
<th>Driver S/W State</th>
<th>H/W Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>disk</td>
<td>0</td>
<td>10/0/14/0.0.0</td>
<td>sdisk CLAIMED</td>
<td>DEVICE</td>
<td>MITSUMI CD-ROM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/dev/dsk/c0t0d0</td>
<td></td>
<td>/dev/rdsk/c0t0d0</td>
<td></td>
</tr>
<tr>
<td>disk</td>
<td>1</td>
<td>10/0/15/0.6.0</td>
<td>sdisk CLAIMED</td>
<td>DEVICE</td>
<td>QUANTUM ATLAS5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/dev/dsk/c2t6d0</td>
<td></td>
<td>/dev/rdsk/c2t6d0</td>
<td></td>
</tr>
</tbody>
</table>

A useful utility to retrieve information on a SCSI target device is `diskinfo`.

$ diskinfo /dev/rdsk/c2t6d0

SCSI describe of /dev/rdsk/c2t6d0:

- vendor: QUANTUM
- product id: ATLAS5-9LVD
- type: direct access
- size: 8886762 Kbytes
- bytes per sector: 512

Another useful utility is `iostat`, which reports I/O statistics for each active SCSI disk on the system. For details please do a \texttt{man} on the above mentioned commands.
Building Drivers

Drivers include kernel header files. Obviously, these files and their respective locations differ between Solaris and HP-UX. Typically, drivers that belong to a class require a set of header files. Header files required by a SCSI Interface driver can be obtained from the sample Qlisp interface driver. A list of common header files required by a HP-UX SCSI interface driver is provided below.

<table>
<thead>
<tr>
<th>Header File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/usr/include/sys/buf.h</td>
<td>The buf I/O buffer structure.</td>
</tr>
<tr>
<td>/usr/include/sys/conf.h</td>
<td>Device switching tables, drv_ops_t and drv_info_t.</td>
</tr>
<tr>
<td>/usr/include/sys/file.h</td>
<td>Flags for open() system calls.</td>
</tr>
<tr>
<td>/usr/include/sys/sysmacros.h</td>
<td>Some commonly used fields in some drivers minor numbers.</td>
</tr>
<tr>
<td>/usr/include/sys/uio.h</td>
<td>The uio structure and its elements.</td>
</tr>
<tr>
<td>/usr/include/sys/types.h</td>
<td>Required for declaring non-basic types.</td>
</tr>
<tr>
<td>/usr/include/sys/param.h</td>
<td>Required by all drivers.</td>
</tr>
<tr>
<td>/usr/include/sys/errno.h</td>
<td>Required for UNIX system error numbers.</td>
</tr>
<tr>
<td>/usr/include/sys/malloc.h</td>
<td>Services needed for acquiring and releasing memory</td>
</tr>
<tr>
<td>/usr/include/sys/mp.h</td>
<td></td>
</tr>
<tr>
<td>/usr/include/sys/kern_svcs.h</td>
<td>Provides declarations for kernel services.</td>
</tr>
<tr>
<td>/usr/include/sys/io.h</td>
<td>The ISC table.</td>
</tr>
<tr>
<td>/usr/include/sys/debug.h</td>
<td>Provides special interfaces for debug kernels.</td>
</tr>
<tr>
<td>/usr/include/sys/callout.h</td>
<td>Required for timeout services.</td>
</tr>
</tbody>
</table>
Drivers are typically placed under the /usr/conf/driver_name directory.

A sample makefile is provided with the DDK to assist driver writers to develop one for their drivers. The HP-UX ANSI C compiler is required to build drivers on HP-UX.

A detailed description of how to compile and building a HP-UX driver is given in Chapter 6, Installing Your Driver of the HP-UX 11i, Driver Developers Guide.

### Debugging Drivers

If the device driver developer needs to for debug, HP-UX provides two very useful debugging tools for this purpose.

#### Q4

Q4 is a kernel crash dump analyzer. Even though best suited for examining kernel dumps, it can also be used to take a snapshot of live kernel and driver data structures. It has a highly user-friendly interface as compared to hba, and is just as powerful in terms of features.

Refer to the Q4 Documentation for additional information.

#### KWDB

Is a Kernel source level debugger. KWDB uses two systems during a debugging session. The one, called the host, runs the kwdb debugger; the other system, called the target, runs the kernel to be debugged. The kwdb debugger is run as a process on the host; just like wdb (gdb). The target system executes the kernel being debugged and is booted in a special manner for kwdb debugging. Target and host communicate with

<table>
<thead>
<tr>
<th>Table 1-6 Common Header Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>/usr/include/sys/scsi.h</td>
</tr>
<tr>
<td>/usr/include/sys/dma.h</td>
</tr>
<tr>
<td>/usr/include/sys/pci.h</td>
</tr>
<tr>
<td>/usr/include/sys/wsio.h</td>
</tr>
<tr>
<td>/usr/include/sys/spinlock.h</td>
</tr>
</tbody>
</table>
Porting Solaris SCSI Interface Driver to HP-UX

Comparison of Driver Development Environments

each other over a LAN or RS-232 link, depending on the types of devices installed in the target. KWDB runs on the PA-RISC architecture, but can remotely debug a PA-RISC or Itanium system.

Refer to HP-UX KWDB Kernel/Driver Debugger 0.9 for addition information.

**Solaris to HP-UX Service Mapping**

<table>
<thead>
<tr>
<th>Solaris to HP-UX Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>kmem_alloc</td>
</tr>
<tr>
<td>kmem_zalloc</td>
</tr>
<tr>
<td>kmem_free</td>
</tr>
<tr>
<td>ddi_get_iblock_cookie</td>
</tr>
<tr>
<td>ddi_add_intr</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>ddi_remove_intr</td>
</tr>
<tr>
<td>ddi_get8/16/32</td>
</tr>
<tr>
<td>ddi_put8/16/32</td>
</tr>
<tr>
<td>ddi_config_get8/16/32</td>
</tr>
<tr>
<td>ddi_config_put8/16/32</td>
</tr>
<tr>
<td>pci_config_setup</td>
</tr>
<tr>
<td>pci_config_teardown</td>
</tr>
<tr>
<td>mutex_init</td>
</tr>
</tbody>
</table>
### Table 1-7  Solaris to HP-UX Mapping

<table>
<thead>
<tr>
<th>Solaris Function</th>
<th>HP-UX Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>mutex_destroy</td>
<td>dealloc_spinlock</td>
</tr>
<tr>
<td>mutex_enter</td>
<td>spinlock</td>
</tr>
<tr>
<td>mutex_tryenter</td>
<td>cspinlock</td>
</tr>
<tr>
<td>mutex_exit</td>
<td>spinunlock</td>
</tr>
<tr>
<td>mutex_owned</td>
<td>owns_spinlock</td>
</tr>
<tr>
<td>cmn_err</td>
<td>cmn_err, msg_printf</td>
</tr>
<tr>
<td>drv_usecwait</td>
<td>busywait</td>
</tr>
<tr>
<td>timeout</td>
<td>Ktimeout, timeout</td>
</tr>
<tr>
<td>untimeout</td>
<td>untimeout</td>
</tr>
</tbody>
</table>
Porting Solaris SCSI Interface Driver to HP-UX

Comparison of Driver Development Environments
2 Qlisp Sample Driver Comparison for Qlogic ISP12160A Ultra160 Card
Qlisp Sample Driver Comparison for Qlogic ISP12160A Ultra160 Card

Driver Sources

A closer comparison is performed in this chapter between the HP-UX Qlisp sample driver and the Solaris Qlisp sample driver for the Qlogic ISP12160A card. This section is seeks to bring out and address issues that may arise when porting a Solaris SCSI HBA driver to HP-UX SCSI Interface driver. This section focuses only on PCI-based SCSI HBA drivers.

The Solaris SCSI HBA driver for Qlogic ISP12160A Ultra160 is referred to as the Solaris sample driver. The HP-UX SCSI Interface driver for Qlogic ISP12160A Ultra 160 is referred to as the HP-UX sample driver and it is provided with the DDK.

As discussed before, the HP-UX and Solaris SCSI subsystems are very similar. They both provide almost the same entry points, albeit with different names and calling syntax. Both, HP-UX and Solaris export certain services that neither driver implements for the sake of keeping the driver simpler. For example, hot-swapping is not implemented in either driver.

Driver Sources

Both the sample drivers have similar source structures. They are divided into the following files:

<table>
<thead>
<tr>
<th><strong>Table 2-1</strong> Driver Source Files</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>scsi_qlisp_multi.c</code></td>
</tr>
<tr>
<td><code>scsi_qlisp.c</code></td>
</tr>
<tr>
<td><code>scsi_qlisp.h</code></td>
</tr>
<tr>
<td><code>scsi_qlisp_fw12160.h</code></td>
</tr>
</tbody>
</table>
Description of the Device

A brief description of the device and its functionality is included to make it easy to understand the driver. It also provides a context and perspective for the data structures. This is not a complete description of the device. For complete instructions, please refer to the Qlogic's hardware reference manual.

The ISP12160A supports dual channel, Ultra160 (Fast-80) SCSI functionality. It interfaces the PCI bus to two Ultra160 SCSI buses and contains an on-board RISC processor. The interface between the ISP12160A firmware and the driver consists of two queues; request and response. The queues are located in host memory and are organized as circular fixed-length queues of 64-byte entries (FIFOs). They are memory mapped and their physical addresses are stored in the device registers.

Both the request and response queues are composed of fixed-length 64-byte entries of the following type.

```c
#define QLISP_Q_ENT_SZ 64
typedef union {
    ubit8 data[QLISP_Q_ENT_SZ];
    qlisp_cmd_ent_t   cmd;
    qlisp_cmd64_ent_t cmd64;/* Simple command entry */
    qlisp_cont_ent_t  cont;
    qlisp_cont64_ent_t cont64;
    qlisp_stat_ent_t  stat;
    qlisp_mark_ent_t  mark;
    qlisp_xcmd_ent_t  xcmd;
    qlisp_perf_stat16_ent_t pstat16;
    qlisp_perf_stat32_ent_t pstat32;
} qlisp_q_ent_t;
```
Qlisp Sample Driver Comparison for Qlogic ISP12160A Ultra160 Card

Description of the Device

For example a cmd64 type command entry has the following definition.

typedef struct {
    qlisp_iocb_hdr_t hdr;

    uint32_t    handle;
    uint8_t     lun_id;
    uint8_t     tgt_id;
    uint8_t     cdb_len_lo;
    uint8_t     cdb_len_hi;
    uint16_t    control_flags;
    uint16_t    res0;
    uint16_t    timeout;
    uint16_t    segment_cnt;
    uint8_t     cdb[12];

    uint64_t   res1;

#define QLISP_CMD64_DSEG_CNT     2
    qlisp_data64_seg_t  dseg[QLISP_CMD64_DSEG_CNT];
}

} qlisp_cmd64_ent_t;

For other definitions please see scsi qlisp.h file. For detailed explanation of each field please refer to the Qlogic Hardware reference manual.
Driver Architecture

The HP-UX sample driver is actually two drivers. One for claiming the PCI interface card and probing the SCSI channels underneath the PCI function and another for claiming the SCSI controllers. The Solaris driver doesn't implement this feature, in other words, it only recognizes the one SCSI channel beneath the PCI function. This was done to keep the Solaris SCSI driver simple and focused on SCSI subsystem related issues.

It is very important to understand that this kind of two-driver architecture is required for special cards with multiple SCSI channels behind a single PCI function. In such cases, the operating system hardware scan cannot recognize the SCSI controllers that are underneath the single PCI interface. Therefore, the two driver approach is required. The qlisp_multi driver registers a driver specific probe function to scan and identify the two SCSI channels under the single PCI function during the driver installation. Now, if the interface card was designed in such a way that each SCSI channel is on a separate PCI function, this kind of two driver approach would not be required.

Also, as far as each driver's functionality is concerned, the qlisp_multi driver is involved only in building the I/O tree with the SCSI controllers. The real work is done by the qlisp driver.
Driver Data Structures

The names of some driver specific data structures were kept the same in order to help the reading and understanding the code.

The following are the main data structures used in the HP-UX Qlisp sample driver.

### Table 2-2  
Data Structures for HP-UX Sample Driver

<table>
<thead>
<tr>
<th>Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qlisp_isc_t</td>
<td>Local interface specific control structure. This is allocated per device instance. Has a pointer to the system ISC structure.</td>
</tr>
<tr>
<td>qlisp_shared_isc_t</td>
<td>Common device control structure. This is allocated when the master port is initialized.</td>
</tr>
<tr>
<td>qlisp_scb_t</td>
<td>Interface-specific SCSI control block structure. This is used to keep track of the associated bp and scb.</td>
</tr>
<tr>
<td>qlisp_bus_t</td>
<td>Interface-specific SCSI bus structure. Contains the NexusTable[], which is used to locally associate an I/O (based on the tag) with an I/O request. In Solaris, the NexusTable[] is moved to qlisp_shared_isc_t because that is the per bus structure.</td>
</tr>
<tr>
<td>qlisp_q_ent_t</td>
<td>Device specific structure. This is the union of all the 64-byte IOCB's supported by the device. The Qlogic ISP12160 supports: command, command A64, Extended command, Continuation, Continuation A64, Marker, Status, and Extended Status.</td>
</tr>
</tbody>
</table>
The following are the main data structures used in the Solaris Qlisp -sample driver.

<table>
<thead>
<tr>
<th>qlisp_isc_t</th>
<th>Very similar to the HP-UX - not in composition, but in use. One allocated per instance. A pointer to this structure is stored in the per HBA structure <code>scsi_hba_tran-&gt;tran_hba_private</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>qlisp_shared_isc_t</td>
<td>This structure contains most of the SCSI bus related variables, i.e. locks, bus status, request and response queues, etc.</td>
</tr>
<tr>
<td>qlisp_cmd_t</td>
<td>Encapsulates a complete SCSI command. It is pointed to by <code>scsi_pkt-&gt;pkt_ha_private</code>. It contains the index/tag for the IOC B sent to the SCSI bus.</td>
</tr>
</tbody>
</table>
Driver Data Structures

Table 2-3

<table>
<thead>
<tr>
<th>qlisp_slot_t</th>
<th>Is an array in qlisp_shared_isc_t. Is an index of all I/Os sent to the HBA.</th>
</tr>
</thead>
<tbody>
<tr>
<td>qlisp_q_ent_t</td>
<td>Since, this is a device specific structure, this remains unchanged between HP-UX and Solaris.</td>
</tr>
</tbody>
</table>
Figure 2-2  Driver Data Structure Inter-relationship in Solaris platform.
Qlisp Sample Driver Comparison for Qlogic ISP12160A Ultra160 Card

Driver Data Structures

Differences In Data Structures

The following are some of the differences between the Solaris and HP-UX versions of these data structures.

1. In Solaris, each I/O sent to the SCSI HBA driver is encapsulated in a `scsi_pkt` structure. The address of the destination is provided in the `scsi_address` structure, which is also stored in a field of the `scsi_pkt` structure. For keeping addition information regarding each I/O, Qlisp has a driver specific structure for each I/O, `qlisp_cmd_t`. It is in this structure that Qlisp keep its status for each I/O. Each I/O has an entry in the `qlisp_shared_isc_t->nexus_table[]` array. The `nexus_table[]` array index is used to uniquely identify and timeout I/Os. In HP-UX, the `nexus_table[]` array is in the `qlisp_bus_t` structure.

When porting from Solaris to HP-UX, `qlisp_cmd_t` and `qlisp_slot_t` structures go away. The three structures - `qlisp_cmd_t`, `scsi_pkt`, and `scsi_address` structures are replaced loosely by `qlisp_scb_t`, `qlisp_bus_t`, `scb`, `scsi_bus` and `buf` structures.

2. Since the Solaris sample does not implement the dual port functionality of the device `qlisp_isc_t` is much smaller when compared to HP-UX version. Nonetheless, this structure still represents the state of the PCI device as a whole.

3. The structure, `qlisp_share_isc_t`, contains all the information for each SCSI channel on both HP-UX and Solaris. This includes all the stated information, request and response queues, DMA mappings, and locks.

4. Another difference is in the driver internal locking mechanisms. To keep the driver simple only one lock is used for an instance of the Solaris SCSI HBA. A conditional variable is used on Solaris, to implement a sleep/wakeup mechanism as a safety net to return timed-out I/Os. On HP-UX, since there are no conditional variables, similar functionality is implemented using wait channels and sleep/wakeup calls.

5. On Solaris, a driver accesses the `buf` structure only during initial setup of an I/O in the entry point `qlisp_scsi_init_pkt()`. In HP-UX, information regarding an I/O is not moved from the `buf` structure to reduce copying information. For managing addition
information required by the SCSI subsystem, the SCSI control block structure, **struct scb**, is used. In this case, the scb structure contains the SCSI cdb and sense data. Also, **scb->if_scb** is provided for use by the interface driver.
Main I/O Path

For both Solaris and HP-UX and I/O starts with a buf structure. It ends with the interface driver calling the callback function.

Solaris, unlike HP-UX, requires a separate HBA driver entry point to allocate resource for an I/O. So, on Solaris an I/O starts getting build when the HBA entry point `qlisp_scsi_init_pkt()` is called. In HP-UX, resource allocation is done in the `qlisp_if_start()` entry point. This also happens to be the entry point that is called when HP-UX SCSI subsystem wants the interface driver to start processing a new I/O. Once, `qlisp_if_start()`, is called, its order of business is to acquire the `scsi_bus` lock and dequeue an I/O from the select queue. Once an I/O has been dequeued, it’s recommended that the `scsi_bus` lock is released. Now the interface driver can further process the I/O it just dequeued.

If the device is busy or not enough resources are available, in Solaris the `qlisp_scsi_start()` entry point may return the I/O by returning `TRAN_BUSY`. In HP-UX, the interface driver can enqueue the I/O back on the select queue by calling `scsi_enqueue()`. If the interface driver find an error condition with the I/O it sets the appropriate fields in the `scb` structure before calling `scsi_cbfn()` with the I/O.

How an I/O is sent to the SCSI peripheral is typically a device dependent and OS independent activity. Qlisp calculates the number of entries required in the Qlogic 12160 request queue and fills these entries appropriately. It then signals the hardware that there are new requests that need processing. From here on the I/O is owned by the hardware, and the driver usually does not touch the `scb` or `buf` structures.

When the hardware is done with the I/O it causes an interrupt. Qlogic 12160A also causes an interrupt when it is done with a mailbox command. In either case, the driver Interrupt Service Routine (ISR) is called by the system. This routine is `qlisp_intr()` in the Solaris sample driver and `qlisp_isr()` in the HP-UX sample driver. Since an interrupt can be caused by multiple events on the hardware, the ISR routine first reads the status register and determines exactly what causes the interrupt. If it turns out that the I/O completed, then the ISR routines reads and saves the information from the response queue. Next order of business is to identify the I/O, which means to relate it to a `qlisp_scb_t` structure in HP-UX or in Solaris to a `qlisp_cmd_t` structure. The HP-UX SCSI subsystem provides unique tags for each I/O sent to the
interface driver. This tag is used by the HP-UX sample driver to identify I/Os sent to the device. Once this is done, the driver can transfer any status information and call the callback routine. In Solaris, the HBA driver calls the function pointed to by scsi_pkt->pkt_comp with the scsi_pkt structure. In HP-UX, the interface driver calls scsi_cbfn() with the buf structure.
Qlisp Sample Driver Comparison for Qlogic ISP12160A Ultra160 Card

Main I/O Path
Glossary

A

Adapter Card Physical hardware, under software control, which is typically attached either directly to an I/O bus or to an auxiliary bus (e.g. SCSI) attached to a directly connected adapter. A device typically combines a hardware controller with the raw mechanism (e.g. disk controller with disk, frame buffer with display).

B

BDR Bus Device Reset. This message is used in order individually to reset a device (without other device reset).

C

Continuous DMA A type of DMA that makes a host memory buffer continuously available to an I/O device. This type of DMA is mainly used for control structures and circular queues that are shared between the device driver and the hardware device.

D

DMA Direct Memory Access. The method used to transfer data to or from I/O devices without the involvement of the CPU.

H

HBA Host Bus Adapter

I

Initiator A SCSI device, which requests an I/O to be performed by another SCSI device.

Interface Driver An HP-UX software driver that controls the adapter card.

Interface Select Code (ISC) Each instance of an adapter card has an ISC entry that the system maintains in an internal table. Each ISC entry is used by WSIO to maintain interface device driver information.

Interrupt Service Routine (ISR) A function that handles interrupts that are received for a specific device driver. A pointer to this is linked into a system vector table that corresponds to the interrupt that was received.

P

Packet DMA A type of DMA that maps a host memory buffer temporarily. This type of DMA is mainly used when pre-existing memory objects must be mapped for DMA, or when a mapping only needs to be temporary.

PCI Peripheral Component Interconnect. An industry standard, high performance bus with support for both 32 and 64-bit data transfer. Used as an interconnect mechanism between
high-integrated peripheral controller components, add-in boards and processor/memory system.

**Queue**

**Queue Tag** The value associated with an I/O process that uniquely identifies it from other queued I/O processes on a logical unit for the same initiator.

**SCSI** Small Computer System Interface. An industry standard external I/O bus available on all HP9000 servers and workstations.

**SDTR and WDTR** Synchronous Data Transfer Request/Wide Data Transfer Request. These are the SCSI messages used to negotiate data transfer parameters.

**Select Queue** Queue from where the SCSI interface driver (qlisp) picks up the I/O from for processing.

**Target** A SCSI device, which performs an operation requested by an initiator.