

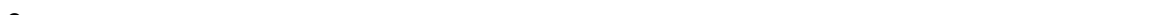
HP-UX 11i to 11i Version 1.6 Driver Migration Guide

Version 1.0



Manufacturing Part Number: N/A

August, 2002



1. 11i to 11i Version 1.6 Driver Migration

Introduction	6
Makefile Changes	6
Changes Required for More Than 4GB of Host Memory	8
32-bit Devices on IPF Systems	8
64-bit Devices on IPF Systems	9
Buffer Allocation Services	14
Interrupt Attribute Services	23
Using KWDB on IPF Systems	29
Starting the KWDB Debugger	29
Booting and Attaching an IPF Target for LAN Debugging	30
Kernel Address Space Layout Difference in IPF and PA	31
On-Line Addition/Replacement Support in HP-UX 11i Version 1.6	32

Contents

Introduction

HP-UX 11i Version 1.6 is targeted to HP Itanium Processor Family (IPF) systems. Some of the IPF platforms running HP-UX do not have I/O TLB support, which maps host memory residing over 4GB range to 32-bit IOVAs. WSIO memory management services assist drivers that manage devices not capable of addressing host memory above 4GB.

This guide is organized into seven sections:

- Makefile changes
- Changes required for more than 4GB of host memory
- Buffer Allocation Services
- Interrupt Attribute Services
- Using KWDB on IPF Systems
- Kernel Address Space Layout Difference in IPF and PA
- On-Line Addition/Replacement support in HP-UX 11i Version 1.6.

Makefile Changes

The following process creates a new Makefile for an IPF driver.

Copy the Makefile `/stand/build/config.mk` to
`/usr/conf/mydriver/Makefile`.

Make the following modifications in the Makefile to build the driver.

1. In the Makefile, change the target “all” to your driver object file. In “mydriver” for example, change the following line in Makefile:

```
from:          all:$(DIR)/$(HPUX)
to:            all:mydriver.o
```

2. In the Makefile replace CONF with your driver name. In “mydriver” for example, change the following line in Makefile:

```
from:          CONF=/stand/build/conf
to:           CONF=mydriver
```

3. Specify ANSI C compiler for “CC=” in the Makefile

4. In the Makefile change the following line:

```
from:          CC=/opt/ansic/bin/cc -Ae
to:           CC=/opt/ansic/bin/cc +legacy_hpc -Ae
```

5. In the Makefile delete the line “KBASE=/usr/conf”. Change the following line in Makefile:

```
from:          INCLUDE_DIRS=-I${KBASE} -I
to:           INCLUDE_DIRS=-I
```

For more information refer to Chapter 6 of the *HP-UX 11i Driver Development Guide*

Changes Required for More Than 4GB of Host Memory

On IPF platforms with host memory present above 4GB physical address, DMA transactions involving buffers above 4GB require I/O devices capable of 64-bit DMA addressing. Drivers that manage I/O devices limited to 32-bit DMA addressing need to copy data from buffers above 4GB to buffers below 4GB to make the data accessible via DMA.

32-bit Devices on IPF Systems

The WSIO services `wsio_setup_dma32()` and `wsio_cleanup_dma32()` allocate and free buffers below 4GB for DMA purposes. These WSIO services allocate the buffers and copy data from the original buffer to the new buffer and vice versa. The services should be used when an attempt to map the buffer using the WSIO mapping service returns `WSIO_MAP_E_HIGH_ADDR` and the driver does not allocate its own DMA buffer below 4GB. For example:

```
retval = wsio_map_dma_buffer(isc, dma_handle, context, hints,
                             range_type, host_range, io_range);

if (retval == WSIO_MAP_E_HIGH_ADDR) {
    status = wsio_setup_dma32(isc, range_type,
                              host_range->iov_base, &tmp_buf,
                              host_range->iov_len, WSIO_DMA_INBOUND);
    if (status == WSIO_ALLOC_OK) {
        /*
         * Use tmp_buf for DMA
         */ tmp_buf needs to be DMA mapped.*/
        ...
        wsio_cleanup_dma32(isc, range_type,
                           host_range->iov_base, tmp_buf,
                           host_range->iov_len, WSIO_DMA_INBOUND);
    }
}
```

If the DMA transaction is outbound, `wsio_setup_dma32()` copies data from the original buffer to the temporary buffer. If it is an inbound transaction, `wsio_cleanup_dma32()` copies data from the temporary buffer to the original buffer.

For cases where the driver allocates its own DMA buffer, the WSIO Buffer Allocation Services should be used.

The service `wsio_map_dma_buffer()` recognizes whether or not the platform provides I/O TLB hardware. If not, it returns the status `WSIO_MAP_E_HIGH_ADDR` when the DMA buffer is above 4GB and the driver has not registered the I/O device as capable of 64-bit DMA addressing.

64-bit Devices on IPF Systems

The driver registers 64-bit addressing capability by calling `wsio_dma_set_device_attributes()` specifying `WSIO_DMA_ATTR_ADDR_WIDTH` as 64. For example:

```
status = wsio_dma_set_device_attributes(isc, dma_handle,  
                                       WSIO_DMA_ATTR_ADDR_WIDTH, 64);
```

NAME

`wsio_setup_dma32`(WSIO3) – Allocate a DMA buffer below 4 GB.

SYNOPSIS

```
#include <sys/wsio.h>

int wsio_setup_dma32 (struct isc_table_type *isc,
                     space_t drv_space,
                     void *drv_buf,
                     void **tmp_buf,
                     size_t size,
                     uint32_t trans);
```

PARAMETERS

<i>isc</i>	Pointer to the device <i>isc</i> structure.
<i>drv_space</i>	Space ID of the driver's buffer.
<i>drv_buf</i>	Pointer to the driver's buffer.
<i>tmp_buf</i>	Address of the pointer to the temporary buffer allocated by the service.
<i>size</i>	Size of the buffer in bytes
<i>trans</i>	Indicates the direction of the DMA action: WSIO_DMA_INBOUND Writing to memory WSIO_DMA_OUTBOUND Reading from memory If <i>trans</i> is WSIO_DMA_OUTBOUND and a temporary buffer was allocated, the service will copy the data from the driver's buffer to the temporary buffer.

DESCRIPTION

This WSIO service allocates a temporary buffer below the 4GB memory address. If *trans* is WSIO_DMA_OUTBOUND, data from the driver's buffer is copied to the temporary buffer.

This service does not block. If resources can not be obtained, it returns to the caller with `WSIO_ALLOC_OUT_OF_RESOURCES`.

For best performance, a driver should only call this routine after a call to `wsio_map_dma_buffer()` returns `WSIO_MAP_E_HIGH_ADDR`.

RETURN VALUES

`WSIO_ALLOC_OK`

The temporary buffer was allocated.

`WSIO_ALLOC_OUT_OF_RESOURCES`

The service was unable to allocate a buffer.

CONSTRAINTS

May be called in user or interrupt context

May not be called while holding a spinlock.

EXAMPLES

```
retval = wsio_map_dma_buffer(isc, dma_handle, context, hints,
                             range_type, host_range, io_range);

if (retval == WSIO_MAP_E_HIGH_ADDR) {
    status = wsio_setup_dma32(isc, range_type,
                              host_range->iov_base, &tmp_buf,
                              host_range->iov_len, WSIO_DMA_INBOUND);
    if (status == WSIO_ALLOC_OK) {
        /*
         * Use tmp_buf for DMA
         */
        ...
        wsio_cleanup_dma32(isc, range_type,
                           host_range->iov_base, tmp_buf,
                           host_range->iov_len, WSIO_DMA_INBOUND);
    }
}
```

SEE ALSO

`wsio_cleanup_dma32(WSIO3)`, `wsio_map_dma_buffer(WSIO3)`

NAME

`wsio_cleanup_dma32(WSIO3)` – Clean up buffer allocated by `wsio_setup_dma32()`.

SYNOPSIS

```
#include <sys/wsio.h>

void wsio_cleanup_dma32 (struct isc_table_type *isc,
                        space_t drv_space,
                        void *drv_buf,
                        void *tmp_buf,
                        size_t size,
                        uint32_t trans);
```

PARAMETERS

<i>isc</i>	Pointer to the device <i>isc</i> structure.
<i>drv_space</i>	Space ID of the driver's buffer.
<i>drv_buf</i>	Pointer to the driver's buffer.
<i>tmp_buf</i>	Pointer to the temporary buffer allocated by the service <code>wsio_setup_dma32()</code> .
<i>size</i>	Size of the buffer in bytes.
<i>trans</i>	Indicates the direction of the DMA action: WSIO_DMA_INBOUND Writing to memory WSIO_DMA_OUTBOUND Reading from memory If <i>trans</i> is WSIO_DMA_INBOUND copy the data from the temporary buffer to the driver buffer.

DESCRIPTION

This WSIO service frees the temporary buffer allocated by `wsio_setup_dma32()`. If DMA is inbound, it copies the data from the temporary buffer to the driver's buffer. If DMA is outbound, no copying is done and the driver may pass a NULL value for *drv_buf*.

RETURN VALUES

None

CONSTRAINTS

May be called in user or interrupt context.

Must not be called while holding a spinlock.

EXAMPLES

```
retval = wsio_map_dma_buffer(isc, dma_handle, context, hints,
                             range_type, host_range, io_range);

if (retval == WSIO_MAP_E_HIGH_ADDR) {
    status = wsio_setup_dma32(isc, range_type,
                              host_range->iov_base, &tmp_buf,
                              host_range->iov_len, WSIO_DMA_INBOUND);
    if (status == WSIO_ALLOC_OK) {
        /*
         * Use tmp_buf for DMA
         */
        ...
        wsio_cleanup_dma32(isc, range_type,
                           host_range->iov_base, tmp_buf,
                           host_range->iov_len, WSIO_DMA_INBOUND);
    }
}
```

SEE ALSO

`wsio_setup_dma32()`

Buffer Allocation Services

The following WSIO DMA services are called by drivers to allocate and free memory used for DMA buffers or control structures.

The first two services are used to create and destroy memory allocator handles. Drivers create memory allocator handles to specify the type of memory they need to allocate. Drivers can create multiple handles for different types of memory allocation.

The second two services allocate and free memory. These services are sensitive to the I/O device and platform capabilities and limitations including:

- Whether the I/O device supports 32 or 64-bit addressing
- Whether the platform supports an IOPDIR
- Whether the platform supports >4GB memory
- The locality of the device on ccNuma platforms

NAME

`wsio_alloc_mem_handle(WSIO3)` – Specify the type of memory to allocate.

SYNOPSIS

```
#include <sys/wsio.h>

wsio_alloc_status_t wsio_alloc_mem_handle(
    struct isc_table_type * isc,
    wsio_mem_handle_t * mem_handle,
    wsio_mem_alloc_attrib_t attrs);
```

PARAMETERS

<i>isc</i>	Pointer to the device <code>isc</code> structure
<i>mem_handle</i>	Pointer to the returned handle.
<i>attrs</i>	Attributes describing the criteria for the type of memory to allocate.
<code>WSIO_OPTIMIZE_FOR_DEVICE</code>	Allocate memory close to the device. On Multi-cell systems, memory will be allocated on the same cell as the device.
<code>WSIO_OPTIMIZE_FOR_CPU</code>	Allocate memory close to the current CPU. On Multi-cell systems, memory will be allocated on the same cell as the CPU. This is the default behavior.
<code>WSIO_32BIT_MEMORY</code>	Allocate memory below 4GB.
<code>WSIO_IO_CONTIGUOUS</code>	On platforms without an IOPDIR physically contiguous memory will be allocated.

WSIO_ALIGN_ON_SIZE

Allocate memory aligned to a power-of-two value greater than or equal to the requested size.

DESCRIPTION

Drivers call this service to specify the type of memory to allocate. The service returns a *mem_handle* which is passed to the WSIO memory alloc and free routines. Drivers can allocate more than one *mem_handle* to specify different criteria for memory allocation.

While it is not a requirement, it is recommended that drivers call this routine early in their initialization sequence. This is due to the high overhead of the routine. Buffer alignment is as follows:

Table 1-1 Buffer Alignment

Allocation Size	Buffer Aligned On
Less Than Cacheline Size	32 Byte Boundary
Greater Than or Equal to Cacheline Size	Cacheline Boundary
Greater Than or Equal to I/O Page Size (4K)	4K Boundary

RETURN VALUES

WSIO_ALLOC_OK

The handle was allocated

WSIO_ALLOC_OUT_OF_RESOURCES

Unable to allocate the specified resources

WSIO_INVALID_PARAM

A parameter was not valid

CONSTRAINTS

Must not be called in interrupt context.

Must not be called while holding a spinlock.

EXAMPLES

```
wsio_alloc_status_t status;
wsio_vaddr_t vaddr;
wsio_mem_handle_t mem_handle;
size_t size;

if(wsio_alloc_mem_handle(my_isc, &mem_handle,
                        WSIO_32BIT_MEMORY |
                        WSIO_IO_CONTIGUOUS) == WSIO_ALLOC_OK)
{
    /* Allocate memory with a non-blocking call
       to wsio_alloc_mem() */
    status = wsio_alloc_mem(mem_handle, size, &vaddr, 0);
}
}
```

SEE ALSO

```
wsio_alloc_mem(WSIO3), wsio_free_mem(WSIO3),
wsio_free_mem_handle(WSIO3)
```

NAME

`wsio_free_mem_handle(WSIO3)` – Destroy handle allocated by `wsio_alloc_mem_handle()`.

SYNOPSIS

```
#include <sys/wsio.h>

void wsio_free_mem_handle (wsio_mem_handle_t mem_handle);
```

PARAMETERS

mem_handle Handle allocated by a call to `wsio_alloc_mem_handle()`.

DESCRIPTION

This WSIO service destroys a *mem_handle* that was allocated by a previous call to `wsio_alloc_mem_handle()`.

RETURN VALUES

None.

CONSTRAINTS

May be called in user or interrupt context.

EXAMPLES

```
/* mem_handle is a handle allocated using
   wsio_alloc_mem_handle() */

wsio_free_mem_handle(mem_handle);
```

SEE ALSO

`wsio_alloc_mem(WSIO3)`, `wsio_alloc_mem_handle(WSIO3)`,
`wsio_free_mem(WSIO3)`

NAME

`wsio_alloc_mem`(WSIO3) – Allocate memory for DMA buffers or control structures.

SYNOPSIS

```
#include <sys/wsio.h>

wsio_alloc_status_t wsio_alloc_mem(
    wsio_mem_handle_t mem_handle,
    size_t size,
    wsio_vaddr_t * vaddr,
    wsio_alloc_flags_t flags);
```

PARAMETERS

<i>mem_handle</i>	Handle allocated by a call to <code>wsio_alloc_mem_handle()</code>
<i>size</i>	Size of the buffer in bytes.
<i>vaddr</i>	Address of the pointer to the allocated buffer. Pointer is set to NULL if unable to allocate a buffer.
<i>flags</i>	Flags which describe restrictions
	WSIO_SLEEP_OK
	Flag to indicate service can sleep if waiting for resources

DESCRIPTION

This WSIO service allocates memory used for DMA buffers or control structures. The first parameter to the service must be a `mem_handle` allocated by a call to `wsio_alloc_mem_handle()`.

RETURN VALUES

WSIO_ALLOC_OK	The buffer was allocated.
WSIO_ALLOC_OUT_OF_RESOURCES	Unable to allocate the specified resources

CONSTRAINTS

If `WSIO_SLEEP_OK` is set in *flags*:

- Must not be called while holding a spinlock
- Must not be called in interrupt context.

EXAMPLES

```
wsio_alloc_status_t status;
wsio_vaddr_t vaddr;
wsio_mem_handle_t mem_handle;
size_t size;

if(wsio_alloc_mem_handle(my_isc, &mem_handle,
                        WSIO_32BIT_MEMORY |
                        WSIO_IO_CONTIGUOUS) == WSIO_ALLOC_OK)
{
    /* Allocate memory with a non-blocking call
       to wsio_alloc_mem() */
    status = wsio_alloc_mem(mem_handle, size, &vaddr, 0);
}
```

SEE ALSO

```
wsio_alloc_mem_handle(WSIO3), wsio_free_mem(WSIO3),
wsio_free_mem_handle(WSIO3)
```

NAME

`wsio_free_mem`(WSIO3) – Free memory allocated by `wsio_alloc_mem`().

SYNOPSIS

```
#include <sys/wsio.h>

void wsio_free_mem (wsio_mem_handle_t mem_handle,
                   wsio_vaddr_t vaddr);
```

PARAMETERS

<i>mem_handle</i>	Handle allocated by a call to <code>wsio_alloc_mem_handle</code> ().
<i>vaddr</i>	Pointer to the allocated buffer.

DESCRIPTION

This WSIO service frees memory allocated by the service `wsio_alloc_mem`().

RETURN VALUES

None.

CONSTRAINTS

May be called in user or interrupt context.
Must not be called while holding a spinlock.

EXAMPLES

```
/* mem_handle is a handle allocated using
   wsio_alloc_mem_handle */
/* vaddr is a pointer to a block of memory allocated using
   wsio_alloc_mem */

wsio_free_mem(mem_handle, vaddr);
```

SEE ALSO

```
wsio_alloc_mem(WSI03), wsio_alloc_mem_handle(WSI03),  
wsio_free_mem_handle(WSI03)
```

Interrupt Attribute Services

The following two interrupt services are provided to drivers to:

- Set the attribute associated with an interrupt object
- Query the value of the attribute associated with an interrupt object

NAME

`wsio_intr_set_attribute`(WSIO3) – Set the attribute value for an interrupt object.

SYNOPSIS

```
#include <sys/wsio.h>

int
wsio_intr_set_attribute ( IN struct isc_table_type *isc,
                        IN wsio_intr_object_t *iobj,
                        IN wsio_intr_attr_t attrib,
                        IN intptr attr_val)
```

PARAMETERS

<i>isc</i>	Pointer to the device ISC structure.
<i>iobj</i>	Interrupt object returned by the <code>wsio_intr_alloc()</code> call.
<i>attrib</i>	The interrupt attribute to change.
<i>attr_val</i>	Value to be assigned to the attribute.

DESCRIPTION

This service is called to set an attribute value associated with the interrupt object. The interrupt corresponding to *iobj* must be disabled before making this call. This can be accomplished by calling the `wsio_intr_deactivate()` or `wsio_intr_deactivate_nowait()` call. Refer to the respective manpages for further information.

This routine can be called from a non-blocking context.

Currently there are no attributes that can be set using this call.

RETURN VALUES

WSIO_OK	The attribute was successfully changed.
WSIO_INVALID_ISC	Not a valid ISC structure.

WSIO_INTR_INV_OBJ	Not a valid interrupt object.
WSIO_INTR_ACTIVATED	The interrupt object is activated, so an attribute can not be sent.
WSIO_NOT_IMPLEMENTED	The specified interrupt attribute is not implemented.
WSIO_PARM_ERROR	The one or more arguments passed to the <code>wsio_intr_set_attribute</code> call were invalid.
WSIO_ERROR	Setting the attribute failed.

CONSTRAINTS

This service must be called on an interrupt object that has been disabled. Refer to the `wsio_intr_deactivate` or `wsio_intr_deactivate_nowait` manpages for further information.

EXAMPLES

```
wsio_intr_object_t iobj;
wsio_intr_attrib_t attrib;
intptr_t          attr_val;

status = wsio_intr_set_attribute ( isc, iobj, attrib,
                                  attr_val);
```

SEE ALSO

```
wsio_intr_alloc(WSIO3), wsio_intr_free(WSIO3),
wsio_intr_activate(WSIO3), wsio_intr_deactivate(WSIO3),
wsio_intr_deactivate_nowait(WSIO3),
wsio_intr_set_cpu_spec(WSIO3), wsio_intr_set_irq_line(WSIO3),
wsio_intr_get_attribute(WSIO3)
```

NAME

`wsio_intr_get_attribute(WSIO3)` – Get the attribute value for an interrupt object.

SYNOPSIS

```
#include <sys/wsio.h>
int
wsio_intr_get_attribute ( IN struct isc_table_type *isc,
                          IN wsio_intr_object_t *iobj,
                          IN wsio_intr_attr_t attrib,
                          OUT intptr_t attr_val);
```

PARAMETERS

<i>isc</i>	Pointer to the ISC table entry for the driver instance.
<i>iobj</i>	WSIO interrupt object.
<i>attrib</i>	Type of interrupt attribute to query.
<i>attr_val</i>	Current value of the attribute.

DESCRIPTION

This service returns the current value of an interrupt attribute for the specified interrupt object. Valid interrupt attributes that can be queried include:

Attribute	Value Returned/Description
WSIO_INTR_ATTR_ENABLED	WSIO_INTR_ATTR_VAL_SET Interrupt is enabled; WSIO_INTR_ATTR_VAL_CLEAR Interrupt is disabled.
WSIO_INTR_ATTR_ORDERED	WSIO_INTR_ATTR_VAL_SET Interrupts are ordered with respect to DMA; WSIO_INTR_ATTR_VAL_CLEAR Interrupts are not ordered with respect to DMA.

WSIO_INTR_ATTR_LEVEL	WSIO_INTR_EDGE_SENSITIVE The interrupt is edge triggered; WSIO_INTR_LEVEL_SENSITIVE The interrupt is level triggered.
WSIO_INTR_ATTR_SHARE_VEC	WSIO_INTR_SHARED The driver's ISR is shared; WSIO_INTR_EXCLUSIVE The driver's ISR is not shared.

This interface can be called from a non-blocking context.

RETURN VALUES

WSIO_OK	Attribute was successfully changed.
WSIO_INVALID_ISC	Not a valid ISC structure.
WSIO_INTR_INV_OBJ	Not a valid interrupt object.
WSIO_NOT_IMPLEMENTED	Specified attribute can not be queried.
WSIO_PARM_ERROR	The <i>attrib</i> parameter is not valid.

CONSTRAINTS

None

EXAMPLES

```
wsio_intr_object_t iobj;
wsio_intr_attr_t attrib;
intptr_t attr_val;

attrib = WSIO_INTR_ATTR_LEVEL;
/* Find out if the interrupt associated with iobj is edge
 * or level triggered.
 */
status = wsio_intr_get_attribute ( isc, iobj, attrib,
                                &attr_val);
```

SEE ALSO

```
wsio_intr_alloc(WSI03), wsio_intr_free(WSI03),  
wsio_intr_activate(WSI03), wsio_intr_deactivate(WSI03),  
wsio_intr_deactivate_nowait(WSI03),  
wsio_intr_set_cpu_spec(WSI03), wsio_intr_set_irq_line(WSI03),  
wsio_intr_set_attribute(WSI03)
```

Using KWDB on IPF Systems

The following describes the use of KWDB on IPF systems using LAN communication and a comm-server. For a comprehensive list of options and detailed guidelines see *KWDB 2.0 Users Guide* at <http://www.hp.com/dspp>.

Starting the KWDB Debugger

To use KWDB to debug a target kernel, start KWDB on a copy of the target kernel executable file on your host system. the `kwdb run` string syntax is:

```
kwdb kernel_file:
```

where *kernel_file* specifies the pathname of the host's copy of the target's kernel file.

The kernel file must be the same as the kernel file booted on the target system. If you are doing source level debugging, you probably want to have the source files that were compiled with debug flags present on your host system. If the source files are not in the default path searched by KWDB, you can tell KWDB where to find them by using the KWDB command:

```
dir path-name
```

where *path-name* is the directory path to the location of the source files.

For HP IPF systems, a `btlan` card is normally used by KWDB for LAN communication between host and target systems. It is possible to select the Intel100 card for communication by booting the target using the `-z` option.

After booting the target, run `kwdb` on the host system as follows:

```
$ kwdb kernel_file  
(kwdb) target ia64_kern comm_server[:47001]  
(kwdb) attach 0 target_id
```

Booting and Attaching an IPF Target for LAN Debugging

Boot the target machine to ISL, and enter the following at the boot prompt:

```
HPUX> boot kernel_file -dlan
```

(Boot-and-wait) - The kernel stops and waits for a client connection during the bootup sequence, rather than booting up normally.

```
HPUX> boot kernel_file -d0x11
```

(Boot-without-wait) - You can attach the debugger client to the target at a later time. On the target, `kernel_file` is usually `/stand/vmunix`.

Kernel Address Space Layout Difference in IPF and PA

The virtual address space layout in IPF is different from that in PA-RISC. On IPF platforms 64-bit virtual address space is divided into 8 different regions (compared to 4 different regions, called quadrants, in PA-RISC). Therefore, in IPF platforms the most significant 3 bits (61-63) of address are used to index a region.

In IPF platforms kernel text and data always fall in the 8th region, i.e, bits 61 through 63 are one. In PA-RISC, kernel text and data always fall in the first quadrant, i.e., bits 61 through 63 are zero.

So, in IPF platforms if we use the kernel addresses as signed integer values for some purpose, all these addresses result in negative values. Following is an example:

In the sleep/wakeup mechanism used in the network driver, the function that decides that the calling function has to sleep returns a negative address value on which calling function has to sleep. This distinguishes between positive error values which are also possible return values. Since the kernel address already has a negative value, making the address negative again yields a positive value. This breaks the sleep/wakeup mechanism.

To avoid this problem, the function which returns the negative address can now return the address as it is, since the MSB (64th bit) is already set for kernel addresses. However, to make code generic to PA and IPF platforms, a check can be added to see if the address is positive. If it is positive, then the negative of it is returned. The calling function now sleeps on a positive value of the (always negative) returned address.

So, actually, the function sleeps on the absolute value of the original address. There are two addresses possible for one absolute value. One with the MSB set and the other with the MSB not set (all other bits being the same). But these addresses never occur together because they fall in different regions (quadrants) and all the kernel data fall in one region (quadrant).

On-Line Addition/Replacement Support in HP-UX 11i Version 1.6

PCI OLA/R functionality is not supported in this HP-UX 11i Version 1.6. However, when porting a driver from 11i to 11i Version 1.6, and OLA/R support is implemented in an 11i driver, no changes are required to the driver source code. OLA/R operations can not be performed on the card, though.