A Deprecated WSIO Interfaces
This appendix contains WSIO reference pages that have been deprecated in HP-UX 11i. These interfaces are documented here to support their use in earlier versions. Developers are encouraged to use the newer WSIO interfaces.
NAME

`init_map_context` (CDIO3) – Macro to initialize mapping context structure

SYNOPSIS

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

#define init_map_context (map)
```

PARAMETERS

- `map` Pointer to a map control block struct.

DESCRIPTION

The `init_map_context()` CDIO macro initializes the mapping context structure for use by `wsio_map()`. Note that the use of a context structure in a `wsio_map()` call overrides any alternate allocation scheme that may have been specified by `wsio_set_attributes()`.

The context structure may be used to map a single object for a single I/O, or it may be used to map multiple objects for multiple I/Os. This feature uses fewer system resources. This feature is most useful for non-interleaving devices (see `wsio_set_attributes (WSIO3)`) when the driver will map several objects and retain those mappings for the life of the driver (semipermanent mappings or reused mappings).

In the case where a context is used for multiple I/Os, `wsio_unmap()` must not be called for ANY objects mapped with a particular context until ALL of the I/Os mapped with that context have completed. Failure to ensure that all I/Os have completed may result in data corruption. It is the programmer's responsibility to ensure that all I/Os mapped with a particular context are complete prior to unmapping any of the I/Os.

The context variable `map` may be either a local or global variable. It is the responsibility of the programmer to provide synchronization of this memory object.

EXAMPLE

See example in `wsio_map` (WSIO3).
SEE ALSO

wsio_map(WSIO3), wsio_fastmap(WSIO3),
wsio_set_attributes(WSIO3), wsio_unmap(WSIO3)
Deprecated WSIO Interfaces

Functions

NAME

isrlink(WSIO3) – Register an interrupt service routine

SYNOPSIS

#include<sys/wsio.h>

int isrlink(struct isc_table_type *isc, int (*isr)(),
          int irq_line, long arg1, long arg2);

PARAMETERS

isc Pointer to the ISC structure for the driver.

isr Pointer to the driver's interrupt service routine.

irq_line Interrupt request line asserted by the device. For PCI devices this should be -1 allowing WSIO services to determine the interrupt request line being used.

arg1 Driver defined parameter passed as the first parameter to isr. Typically, isc is passed as arg1.

arg2 Driver defined parameter passed as the second parameter to isr.

DESCRIPTION

The isrlink() WSIO function registers an interrupt service routine (ISR). isrlink() is typically called in the driver_if_init() function, if specified by driver_attach() or in driver_attach().

When isrlink() returns, interrupts for the assigned ratline are enabled. The driver should be prepared to handle an interrupt from its device or another device sharing the irq_line.

RETURN VALUES

0 Successful completion.

WSIO_ERROR Error.
CONSTRAINTS

EXAMPLE

```c
static int
mydrv_if_init(struct isc_table_type *isc)
{
    ...
    mydrv_reset_hw(isc);
    return isrlink(isc, mydrv_isr, -1, (long)isc, 0L);
}
```

SEE ALSO

`driver_attach(WSIO_DRV)`, `driver_if_init(WSIO_DRV)`, `driver_isr(WSIO_DRV)`, `isrunlink(WSIO3)`. 
Deprecation WSIO Interfaces

**NAME**

`isrunlink(WSIO3)` – Remove the ISR registered by `isrlink()`

**SYNOPSIS**

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

int isrunlink (struct isc_table_type *isc, int (*isr)(),
        int irq_line, long arg1, long arg2);
```

**PARAMETERS**

- **isc** Pointer to the ISC structure for the driver.
- **isr** Pointer to the driver's interrupt service routine.
- **irq_line** Interrupt request line asserted by the device. For PCI devices this should be -1, allowing WSIO services to determine the interrupt request line used.
- **arg1** Driver defined parameter passed as the first parameter to ISR.
- **arg2** Driver defined parameter passed as the second parameter to ISR.

**DESCRIPTION**

The `isrunlink()` WSIO function removes the `isr` registered by `isrlink()`. This function should be called before a driver is unloaded but after the device has been quiesced.

**RETURN VALUES**

- **0** Successful completion.
- **WSIO_ERROR** Error.

**CONSTRAINTS**
SEE ALSO

isrlink(WSIO3)
Deprecated WSIO Interfaces

Functions

NAME

\texttt{m\_instance}(WSIO3) – Get the device instance field from the device number

SYNOPSIS

#include <sys/io.h>

int m\_instance(dev_t dev);

PARAMETERS

\begin{itemize}
  \item \texttt{dev} \quad \text{Device number of the device.}
\end{itemize}

DESCRIPTION

The \texttt{m\_instance()} WSIO macro returns the driver instance field from the device number.

RETURN VALUES

CONSTRAINTS

SEE ALSO
NAME

wsio_allocate_shared_memory (WSIO3) – Allocate and map contiguous memory used for continuous DMA.

SYNOPSIS

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

shmep_status_t wsio_allocate_shared_memory (  
    struct isc_table_type * isc,  
    size_t size,  
    caddr_t * iova,  
    caddr_t * vaddr,  
    wsio_shmem_attr_t type);
```

PARAMETERS

- **isc**  
  Pointer to an ISC table entry.
- **size**  
  Size in bytes or memory to be allocated.
- **iova**  
  Pointer to the returned I/O virtual address.
- **vaddr**  
  Pointer to the returned virtual address.
- **type**  
  Bit mask of the requested memory attributes.

DESCRIPTION

The `wsio_allocate_shared_memory()` WSIO function is a deprecated interface and may be obsoleted in a future release of HP-UX. Use the `wsio_allocate_shared_mem()` WSIO function in its place.

The `wsio_allocate_shared_memory()` function allocates and maps contiguous memory used for continuous DMA. For packet DMA (short lived DMA typical of I/O transactions), temporary mappings should be done using `wsio_map()` . Continuous DMA is intended for accesses by a device on a continuous basis, typically for device control and status.

Platforms that implement I/O virtual addressing will allocate memory contiguous in I/O virtual address space; platforms that do not will allocate memory contiguous in physical address space. Some platforms may allocate memory that is local to an I/O adapter (aka a “bus bridge”), and such memory may be severely limited in size.
The `isc` parameter is a pointer to the ISC table entry assigned to the driver's interface card. It is the handle for the driver instance.

The `size` parameter is the size in bytes of memory to be allocated.

The `iova` parameter is a pointer to the returned I/O virtual address. It is the base address of the allocated memory from the view of the interface card.

The `vaddr` parameter is a pointer to the returned virtual address corresponding to the I/O virtual address. It is the base address of the allocated memory from the view of the processor.

The `type` parameter is a bit mask of the requested memory attributes. Valid memory attributes are the following:

- `WSIO_SHMEM_OPTIMIZE_DEVICE_LATENCY` - allocation should optimize for device access latency. If the platform allows, allocated memory should be local to the I/O adapter connecting the interface card. This is the default attribute if none are specified.
- `WSIO_SHMEM_OPTIMIZE_HOST_LATENCY` - allocation should optimize for host access latency. If the platform allows, allocated memory should be in host memory.
- `WSIO_SHMEM_INBOUND` - hint indicating the allocated memory will be used exclusively for inbound (device to memory) DMA only
- `WSIO_SHMEM_OUTBOUND` - hint indicating the allocated memory will be used exclusively for outbound (memory to device) DMA only
- `WSIO_SHMEM_DEV_WEAK_OK` - hint indicating accesses to the allocated memory can be weakly ordered.
- `WSIO_SHMEM_ALIGN_ON_SIZE` - allocation must align the memory on the size specified. If, for example, size is a power of 2, the base address of the memory allocated must be aligned to the same power of 2 or a multiple of that value.

**RETURN VALUES**

- `SHMEM_OK` Successful completion
- `SHMEM_NO_RESOURCES` Memory not allocated
CONSTRAINTS

EXAMPLES

caddr_t my_iova;
caddr_t my_vaddr;

/*
 * Allocate contiguous memory that is page bytes in size
 * and aligned to a page size boundary. If the platform
 * allows, allocate memory that is local to the I/O
 * adapter (bus bridge) connecting the interface card.
 */
if (wsio_allocate_shared_memory(
    isc, NBPG, &my_iova, &my_vaddr,
    WSIO_SHMEM_OPTIMIZE_DEVICE_LATENCY |
    WSIO_SHMEM_ALIGN_ON_SIZE) != SHMEM_OK) {

    /*
     * Failed to allocate memory for continuous DMA.
     */
}

SEE ALSO

wsio_allocate_shared_mem(WSIO3),
wsio_flush_shared_memory(WSIO3),
wsio_free_shared_memory(WSIO3)
Deprecated WSIO Interfaces
Functions

NAME

wsio_fastmap (WSIO3) – Map all or part of a host address range into an I/O virtual address range

SYNOPSIS

#include <sys/dma.h>

int wsio_fastmap (struct isc_table_type *isc, int range_type, struct iovec *host_range, struct iovec *io_range);

PARAMETERS

isc Pointer to an ISC structure.
range_type The type of host address for host_range. If range_type > 0 then this is the space ID of the address range. Other values are:
            KERNELSPACE Kernel virtual buffer (same as space ID = 0)
host_range A pointer/length pair indicating the host address range of type range_type. The length will be modified by the service to indicate the bytes remaining to be mapped. This length is also returned by wsio_fastmap().
io_range A pointer/length pair filled by wsio_fastmap() with the I/O virtual address range or EISA address range.

DESCRIPTION

The wsio_fastmap() WSIO function provides the same functionality as wsio_map() when the entire host address range resides on a single physical page. This condition is not checked by wsio_fastmap(). Cache-line fragments are ignored (as in wsio_map() with the IO_IGN_ALIGNMENT hint).
Since the behavior and side effects are different for coherent and noncoherent systems, these are discussed separately below. Although the behaviors are slightly different, drivers should not have dependencies on these differences. In all cases, the programming model is the same.

All mappings remain valid until the `io_range` is unmapped by calls to `wsio_unmap()`. If the `io_range` is reused, `dma_sync()` must be used to resynchronize it.

**Behavior on Noncoherent Systems**

On noncoherent systems, the I/O virtual address is equivalent to the host physical address. `wsio_map()` will return an `io_range` at each page break in the `host_range`. If the mapping is for an EISA module, then the `io_range` is filled with EISA addresses and the EISA map will contain the host physical address.

**Behavior on Coherent Systems**

On coherent systems, the I/O virtual address is obtained by creating a mapping in the I/O PDIR. For cache-line fragments, the SAFE bit will be set in the I/O PDIR, unless inhibited by `IO_IGN_ALIGNMENT`. However, it is expected that buflets will be used by the caller to handle them. If the mapping is for an EISA module, then the `io_range` is filled with EISA addresses and the EISA map will contain the corresponding IOVAs.

**RETURN VALUES**

- `wsio_fastmap()` always returns 0 on a Noncoherent system.
- `wsio_fastmap()` returns the following values on a Coherent system:
  - 0: The range was fully mapped.
  - -1: The necessary resources could not be obtained.

**CONSTRAINTS**
WARNING

The IO_NO_SEQ flag is NOT set with wsio_fastmap(), and if the mapping is cache line aligned the IO_SAFE bit is NOT set. Under certain conditions this call MUST NOT be used for PCI. See pci-errata (PC15) for details.

EXAMPLE

The following function maps a single page of memory (virt_page is aligned on a page boundary):

```c
void my_page_map(uint port_num, void *virt_page) {
    struct iovec host, io;
    host.iov_base = virt_page; /* virt_page is page-aligned */
    host.iov_len = NBPG;
    wsio_fastmap(port_num, KERNELSPACE, &host, &io);
    return io.iov_base;
}
```

WARNINGS

It is up to the caller to ensure that the space to be mapped does not cross a page boundary. On a Coherent system this will be indicated by a return of -1, on a Noncoherent system the call will succeed.

SEE ALSO

dma_sync(CDIO3), init_map_context(CDIO3),
isc_table_type(KER4), pci_errata(PC15), wsio_fastmap(WSIO3),
wsio_remap(WSIO3), wsio_set_attributes(WSIO3),
wsio_unmap(WSIO3)
NAME

wsio_flush_shared_memory(WSIO3) – Flush the memory previously allocated and mapped by wsio_allocate_shared_memory().

SYNOPSIS

#include <wsio/wsio.h>

shmem_status_t wsio_flush_shared_memory(
    struct isc_table_type *isc, size_t size, 
    iova_t iova, vaddr_t vaddr, 
    wsio_shmem_attr_t type);

PARAMETERS

isc Pointer to an ISC table entry.
size Size in bytes or memory to be flushed.
iova I/O virtual address.
vaddr Virtual address corresponding to iova.
type Bit mask of the memory attributes.

DESCRIPTION

The wsio_flush_shared_memory() WSIO function is a deprecated interface and may be obsoleted in a future release of HP-UX. Use the wsio_flush_shared_mem() WSIO function in its place.

The wsio_flush_shared_memory() WSIO function flushes the memory previously allocated and mapped by wsio_allocate_shared_memory(). This ensures that data in the allocated memory is viewed consistently by the device and processors. All parameters passed to wsio_flush_shared_memory() must match the parameters passed to the corresponding call to wsio_allocate_shared_memory().

RETURN VALUES

SHMEM_OK Successful completion
SHMEM_NO_RESOURCES Memory not flushed
Deprecated WSIO Interfaces

Functions

CONSTRAINTS

SEE ALSO

wsio_allocate_shared_memory(WSIO3),
wsio_flush_shared_mem(WSIO3), wsio_free_shared_memory(WSIO3)
Deprecation WSIO Interfaces
Functions

NAME

wsio_free_shared_memory(WSIO3) – Release and unmap contiguous memory previously allocated and mapped by wsioAllocate_shared_memory().

SYNOPSIS

#include <wsio/wsio.h>

void wsio_free_shared_memory (struct isc_table_type *isc,
size_t size, caddr_t iova, caddr_t vaddr,
wsio_shmem_attr_t type);

PARAMETERS

isc Pointer to an isc_table entry.
size Size in bytes or memory to be released.
iova I/O virtual address.
vaddr Virtual address corresponding to iova.
type Bit mask of the memory attributes.

DESCRIPTION

The wsio_free_shared_memory() WSIO function is a deprecated interface and may be obsoleted in a future release of HP-UX. Use the wsio_free_shared_mem() WSIO function in its place.

The wsio_free_shared_memory() function releases and unmaps contiguous memory previously allocated and mapped by wsioAllocate_shared_memory(). All parameters passed to wsio_free_shared_memory() must match the parameters passed to the corresponding call to wsioAllocate_shared_memory().

RETURN VALUES

None
Deprecated WSIO Interfaces

Functions

CONSTRAINTS

SEE ALSO

wsio_allocate_shared_mem(WSIO3), wsio_free_shared_mem(WSIO3), wsio_flush_shared_mem(WSIO3)
NAME

wsio_get_interrupts (WSIO3) – Determine which interrupt has been assigned to a card

SYNOPSIS

#include <sys/wsio.h>

input_t wsio_get_interrupts (struct isc_table_type * isc);

PARAMETERS

isc A pointer to the ISC structure associated with the interface card.

DESCRIPTION

The wsio_get_interrupts() WSIO function determines the IRQ of an interface card hardware module associated with the given ISC structure. It expects the "interrupt" property of the iotree node to have been appropriately assigned during I/O configuration. It can be used by drivers for getting the information needed to set up their isc->eim values and for setting up their isrlink() service calls. PCI bus drivers must use -1 for their isrlink() calls, allowing the PCI services to determine the actual interrupt line assigned.

RETURN VALUES

>0 The IRQ for the card.

-1 isc is NULL or there was a problem retrieving the node's interrupt property.

CONSTRAINTS
Deprecated WSIO Interfaces

Functions

EXAMPLES

mydriver_init(isc)
struct isc_table_type *isc;
{
  int wsio_eim;
  ...

  if (NOT_PCI) {
    mydriver_reset(isc);
    isc->eim = wsio_eim = wsio_get_interrupts(isc);
    isc->eim_control = 0;
  } else {
    mydriver_reset(isc);
    wsio_eim = -1;
  }
  isrlink(isc,mydriver_isr,wsio_eim,isc,isc->if_drv_data);
  ...
}

SEE ALSO

isc_table_type(KER4)
Deprecated WSIO Interfaces
Functions

NAME

wsio_get_pva(WSIO3) – Translate an IO virtual address to its processor virtual address

SYNOPSIS

#include <sys/wsio.h>

void * wsio_get_pva (struct isc_table_type *isc, caddr_t iova);

PARAMETERS

isc Pointer to an ISC structure.
iova An I/O Virtual Address.

DESCRIPTION

The wsio_get_pva() WSIO macro translates I/O virtual addresses into processor virtual addresses. The translation is more efficient if the memory object is equivalently mapped (ProcVA == Physical address). This function is restricted to virtual buffers in kernel space.

RETURN VALUES

<>NULL The processor virtual address, if the I/O virtual address exists and there is a corresponding processor virtual address in KERNELSPACE.
NULL Otherwise.

CONSTRAINTS
EXAMPLES

```c
#include <sys/dma.h>
#define HOST_RAM_SIZE 0x8000

extern int coherent_io_enabled;

my_attach(id,isc)
PCI_ID id;
struct isc_table_type *isc;
{
  caddr_t host_ram;
  int i, pages_mapped;
  caddr_t io_tmp, proc_tmp;
  ...

  MALLOC(host_ram,cadr_t, HOST_RAM_SIZE,M_DYNAMIC,M_NOWAIT);
  bzero(host_ram,HOST_RAM_SIZE);
  pages_mapped = HOST_RAM_SIZE/NBPG;
  if (!coherent_io_enabled) {
    /*
     ** need to ensure a contiguous
     ** buffer on processors that do
     ** not have an IO TLB, because
     ** our card expects contiguous
     ** space for task lists
     */
    proc_tmp = host_ram;
    io_tmp = wsio_get_pva(isc,proc_tmp);
    for (i=0;i < pages_mapped;
         i++,io_tmp += NBPG,proc_tmp += NBPG) {
      if (io_tmp != wsio_get_pva(isc,proc_tmp)) {
        msg_printf("my_attach IO buffer not contiguous\n");
        FREE(host_ram,M_DYNAMIC);
        return (*my_pci_saved_attach)(id,isc)
      }
    }
  }
  ...

  return (*my_pci_saved_attach)(id,isc)
}
SEE ALSO

wsio_fastmap(WSIO3), wsio_map(WSIO3), wsio_remap(WSIO3),
wsio_unmap(WSIO3)
NAME

\texttt{wsio\_get\_registers}(WSIO3) – Get the register addresses of an interface card

SYNOPSIS

\begin{verbatim}
#include <sys/wsio.h>

caddr_t * wsio_get_registers (struct isc_table_type * \textit{isc});
\end{verbatim}

PARAMETERS

\textit{isc} A pointer to the ISC structure associated with an interface card.

DESCRIPTION

The \texttt{wsio\_get\_registers()} WSIO function retrieves the register addresses of the interface card associated with the given ISC entry.

Any module that has additional SPA space (for example, graphics) will need access to both of the register-property addresses. Normally, only the first register is retrieved and held in the ISC structure. This routine will allow drivers to get both register pointers if they need them.

\texttt{wsio\_get\_registers()} is not supported for PCI interface cards.

RETURN VALUES

\begin{itemize}
  \item \texttt{-1} Successful completion. The value is a pointer to an array of register sets for this module (for most modules, this will be an array of 1 or 2 elements, corresponding to HPA and SPA).
  \item \texttt{-1} Failure. \textit{isc} is NULL or there was a problem retrieving the node's registers property.
\end{itemize}

CONSTRAINTS
SEE ALSO
NAME

wsio_install_drv_func(WSIO_DRV) – Register a driver function with the WSIO driver environment.

SYNOPSIS

#include <wsio/wsio.h>

int wsio_install_drv_func (wsio_drv_info_t *drv_hdr,
       wsio_drv_func_type_t func_id,
       wsio_drv_func_t drv_func,
       wsio_uintptr_t arg1,
       wsio_uintptr_t arg2);

PARAMETERS

drv_hdr A pointer to the driver's wsio_drv_info_t structure.
func_id Identify what function a driver is registering.
drv_func A driver's function to register with wsio.
arg1 func_id dependent.
arg2 func_id dependent.

func_id drv_func

WSIO_DRV_CLAIMFUNC

A driver's claim function to claim underlying devices.

arg1 & arg2 - not applicable; set to 0.

DESCRIPTION

This function is used to register a driver function with WSIO. The type of function is identified by func_id. When a service is needed, WSIO will execute this function. Depending on the function type, the args may be passed back to a driver. A driver should call this function only after it has called wsio_install_driver().
**RETURN VALUES**

- **WSIO_OK**
  Successful completion.

- **WSIO_DRV_FUNC_NULL**
  drv_func is NULL.

- **WSIO_DRV_NOT_FOUND**
  Invalid driver. Driver should call 
  wsio_install_driver() first.

- **WSIO_INFO_NULL**
  wsio_drv_info_t pointer is NULL.

- **WSIO_UNKNOWN_FUNC_TYPE**
  Invalid func_id.

**CONSTRAINTS**

**EXAMPLE**

```c
static wsio_drv_info_t my_drv_info {
    ....
} 
int my_claim(wsio_generic_data_t *data_ptr) 
{ 
    .........
} 
int my_install(void) 
{ 
    ....
    if (wsio_install_driver(&my_drv_info)) {
        ....
        /* Register my claim function, arg1 and arg2 are 
         * reserved, set to 0 
        */
        if (wsio_install_drv_func(&my_drv_info,
            WSIO_DRV_CLAIM_FUNC, my_claim, 0, 0))
            ....
    } 
    ....
} 
```
SEE ALSO

wsio_install_drv_event_handler(WSIO3),
wsio_query_supported_function(WSIO_DRV)
NAME

wsio_map(WSIO3) – Map all or part of a host address range into an I/O virtual address range

SYNOPSIS

#include <sys/wsio.h>

int wsio_map (struct isc_table_type *isc, io_map_t *map_cb,
             int hints, int range_type,
             struct iovec *host_range,
             struct iovec *io_range);

PARAMETERS

isc A pointer to the isc struct for this device which contains information that is bus specific used by the underlying mapping services to correctly set up the mapping hardware and other bus specific details.

map_cb A control structure which is private to the mapping service. This structure stores the mapping context across multiple calls to wsio_map(). map_cb can be a local variable. If used, it must be initialized before the first call to wsio_map() via a call to init_map_context(). A non-NULL value causes the default IOVA allocation scheme to be used regardless of previous calls to wsio_set_attributes().

hints Hints which change the behavior of wsio_map():

   IO_CONTIGUOUS Indicates that wsio_map() must allocate a single contiguous I/O virtual range. If wsio_map() is unable to do this, it will return -1. Of course, on noncoherent systems, the object must by physically contiguous. This hint implies IO_IGN_ALIGNMENT.
### Deprecated WSIO Interfaces

#### Functions

**IO_IGN_ALIGNMENT**

Indicates that `wsio_map()` should not set the SAFE bit for cache-line fragments and that `wsio_map()` should not return separate `io_ranges` for cache-line fragments.

**IO_LOCK**

Forces the LOCK bit to be set in the I/O PDIR for this mapping. This hint should be used for devices which can request exclusive access to memory. For instance, an EISA card can assert the EISA LOCK signal to request exclusive access to memory. Memory objects used in this way must be mapped with the `IO_LOCK` hint specified.

**IO_NO_SEQ**

Turns off the SEQUENTIAL bit in the I/O PDIR for this mapping. This inhibits prefetching of data for this object by the I/O Adapter. This hint has no effect for modules which do not prefetch data.

**IO_SAFE**

Forces the SAFE bit to be set in the I/O PDIR for this mapping. This causes the I/O subsystem to perform read-modify-write bus transactions for this mapping. This hint should be specified if sub-cacheline sized DMA will be used for the buffer.

**IO_SEMA**

Provides a hint that this memory object will be used as a semaphore.

**IO_UPDATE**

Forces the UPDATE bit to be set in the I/O PDIR for this mapping.

**range_type**

The type of host address for `host_range`. If `range_type > 0`, this is the space ID of the address range. Other values are:

- **KERNELSPACE**: Kernel virtual buffer (same as space ID = 0)
Deprecated WSIO Interfaces

Functions

host_range  A pointer/length pair indicating the host address range of type range_type. The length will be modified by the service to indicate bytes remaining to be mapped. This length will also be returned by wsio_map().

io_range  A pointer/length pair filled by wsio_map() with the I/O virtual address range or the EISA address range.

DESCRIPTION

The wsio_map() macro maps the host address range into an I/O virtual address range. For EISA busses, the range is also mapped into EISA space. It may take multiple calls to wsio_map() to map the entire host range due to cache-line and page alignment restrictions. If the host_range is not aligned on a cache-line, then the first io_range will also not be cache-line aligned and will contain only the bytes in the same cache-line. Likewise, if the host_range does not end on a cache-line, then the last io_range will represent a cache-line fragment (but will be cache-line aligned). Buffers for the cache-line fragments must be managed by the caller.

Since the behavior and side effects are different for coherent and noncoherent systems, these are discussed separately below. Although the behaviors are slightly different, drivers should not have dependencies on these differences. In all cases, the programming model is the same. For each mapping:

- Call init_map_context() for the map_cb if used.
- Repeatedly call wsio_map() until the return value is less than or equal to 0.

All mappings remain valid until the io_range is unmapped via calls to wsio_unmap(). If the io_range is reused, dma_sync() must be used to resynchronize it.

Behavior on Noncoherent Systems

On noncoherent systems, the I/O virtual address is equivalent to the host physical address. wsio_map() will return an io_range at each page break in the host_range. If the mapping is for an EISA module, then the io_range is filled with EISA addresses and the EISA map will contain the host physical address.
Behavior on Coherent Systems

On coherent systems, the I/O virtual address is obtained by creating a mapping in the I/O PDIR. For cache-line fragments, the SAFE bit will be set in the I/O PDIR unless inhibited by IO_IGN_ALIGNMENT. However, it is expected that buffers will be used by the caller to handle them. If the mapping is for an EISA module, then the io_range is filled with EISA addresses and the EISA map will contain the corresponding IOVAs.

RETURN VALUES

>0 The number of bytes remaining in the host_range.
0 The range was fully mapped.
-1 The necessary resources could not be obtained.

CONSTRAINTS

WARNING

Under certain conditions a PCI master MUST have the IO_SAFE and IO_NO_SEQ flag bits set to ensure coherency. See pci-errata (PC15).

EXAMPLES

The following function maps a set of host pointer/length pairs given by host_vec (WSIO driver):

```c
int my_driver_output(struct isc_table_type *isc, int vec_cnt, struct iovec *host_vec)
{
    io_map_t context;
    struct iovec *io_vec;
    struct my_dma_type *dma_desc;
    int resid;

    init_map_context(&context);

    /* Allocate a DMA structure for my DMA model */
    MALLOC( dma_desc, sizeof(struct my_dma_type) );
```
The following example attempts to allocate 32Kbytes of contiguous memory for a PCI SCSI bus master's task lists. Because there is no API for contiguous memory on a Noncoherent system the routine checks for this. This scheme has the highest chance of success if it is done during PCI attach time because memory is generally not fragmented at that time. For 10.20 coherent_io_enabled indicates whether this is a Coherent system or a Noncoherent system.

```c
#define HOST_RAM_SIZE 0x8000
#define TMP_BUF_SIZE 0x200
extern int coherent_io_enabled;

sample_pci_attach(id, isc)
    PCI_ID id;
    struct isc_table_type *isc;
    {   
        caddr_t tmp_buf,host_ram;
        /* Synchronize DMA buffer (outbound DMA) */
        dma_sync_IO(KERNELSPACE, host_vec->iov_base,  
                    host_vec->iov_len, IO_SYNC_FORDEV);
        /* Point to the beginning of the DMA Vector area */
        io_vec = &dma_desc->iov;
        /* Do mapping for each host vector */
        for (; vec_cnt; host_vec++){
            do {
                resid = wsio_map(my_isc, context,0,KERNELSPACE,  
                                host_vec,io_vec);
                if (resid < 0){
                    /* Handle Error condition */
                }
            } while (resid > 0);
        }
        dma_desc->iov_cnt = io_vec - &dma_desc->iov;
        /* Kick off the DMA */
        return my_start_output(isc,dma_desc);
    }
```
caddr_t tmp_buf_phys, host_ram_phys;
struct iovec host_vec, io_vec;
struct iovec *io_vec_ptr *workptr;
io_map_t map_cb;

/* code to check that it's our card */
/* set up memory */
MALLOC(tmp_buf, caddr_t, TMP_BUF_SIZE, M_DYNAMIC, M_NOWAIT);
bzero(tmp_buf, TMP_BUF_SIZE);
MALLOC(host_ram, caddr_t, HOST_RAM_SIZE, M_DYNAMIC, M_NOWAIT);
bzero(host_ram, HOST_RAM_SIZE);

/*
** WSIO mapping services do different things
** on coherent IO systems (C-Class) and
** non-coherent IO systems (B-Class).
**
** In both cases the mapping call manipulates
** the host iovec base and length fields so
** that wsio_map can be called repeatedly.
**
** -------------------------------
** coherent IO systems behave as follows:
** A single call attempts to map all
** pages in host.iov_len. The IO_CONTIGUOUS
** flag forces a call failure if the mapping
** crosses a 'range' boundary (currently 32K)
** AND unmaps all of the pages. If the
** IO_CONTIGUOUS flag is not set, the call
** will return with 0, but with host.iov_len > 0,
** indicating that you need to call wsio_map
** again, n.b., the next mapping may NOT be
** contiguous from the point of view of PCI.
**
** -------------------------------
** non-coherent IO systems behave as follows:
** The IO_CONTIGUOUS flag is ignored, and
** at most, each call will map a single
** page (4K), there is no guarantee that
** malloc will have given you a contiguous
** buffer. Basically, you need to check
** each page and call wsio_map for each page.
**
** The unmap call is a no-op on B-Class
**
** n.b., don't depend upon the 'range' boundary
** for future releases remaining at 32K. I went
** ahead and commented this because we have seen a
** failure mode due to a mapping order of:
** tmpbuf = 200 bytes = a page, and then
** hostram = 0x7a4 bytes = 8 pages
** i.e., a total of 9 pages crossing a
** range boundary -> the map call failed.
**
*/

int pages_mapped = HOST_RAM_SIZE / NBPG;
int i, map_ret;

if (HOST_RAM_SIZE % NBPG)
    pages_mapped ++;
/* see if we have a physically contiguous buffer
 * on B-Class
*/
if (!coherent_io_enabled) {
    caddr_t phys_tmp, virt_tmp;

    virt_tmp = host_ram;
    phys_tmp = wsio_get_pva(isc,virt_tmp);
    for (i=0;i<pages_mapped;i++,phys_tmp += NBPG,virt_tmp += NBP G) {
        if (phys_tmp != wsio_get_pva(isc,virt_tmp)) {
            msg_printf("sample attach B-Class buffer not contiguous\n"");
            FREE(tmp_buf, M_DYNAMIC);
            FREE(host_ram, M_DYNAMIC);
            return (*sample_pci_saved_attach)(id, isc);
        }
    }
}
/*
 ** now do the mappings -
 ** do hostram first because
 ** it fills a C-Class 'range'
 */
MALLOC(io_vec_ptr, struct iovec *,
    sizeof(struct iovec) * pages_mapped, M_DYNAMIC, M_NOWAIT);
if (io_vec_ptr == NULL) {
    FREE(tmp_buf, M_DYNAMIC);
    FREE(host_ram, M_DYNAMIC);
    return (*sample_pci_saved_attach)(idc, isc);
Deprecated WSIO Interfaces

Functions

```c
}
workptr = io_vec_ptr;
host_vec.iov_base = host_ram;
host_vec.iov_len = HOST_RAM_SIZE;
init_map_context(&map_cb);
for (i=0;i<pagesMapped;i++,workptr++) {
    map_ret = wsio_map(isc, &map_cb,
        IO_CONTIGUOUS | IO_SAFE | IO_LOCK,
        KERNELSPACE,&host_vec, workptr);
    if (map_ret < 0) {
        msg_printf("sample attach: could not map hostram pointer\n"
        );
        FREE(tmp_buf, M_DYNAMIC);
        FREE(host_ram, M_DYNAMIC);
        FREE(iov_ec_ptr, M_DYNAMIC);
        return (*sample_pci_saved_attach)(id, isc);
    }
    if (i==0)
        host_ram_phys = io_vec_ptr->iov_base; /* base addr for PCI */
    if (map_ret == 0)
        break;
}
/**
 ** now map the tmp_buf
 **
 ** Note that we map it after hostram
 ** because 10.20 uses a 32 K range and
 ** if we did it before hostram the
 ** contiguous call would fail on a C class
 ** because we would have used up TMP_BUF_SIZE
 ** of the 32 K range.
 **
 */
init_map_context(&map_cb);
host_vec.iov_base = tmp_buf;
host_vec.iov_len = TMP_BUF_SIZE;
    if ( wsio_map ( isc , &map_cb ,
        IO_CONTIGUOUS | IO_SAFE | IO_LOCK,
        KERNELSPACE , &host_vec , &io_vec ) ) {
        printf("sample attach Could not map() tmp_buf pointer.\n",
    );
    io_vec.iov_base = host_ram_phy_addr;
    io_vec.iov_len = HOST_RAM_SIZE;
    wsio_unmap(isc, io_vec);
    FREE(tmp_buf, M_DYNAMIC);
    FREE(host_ram, M_DYNAMIC);
```
FREE(io_vec_ptr, M_DYNAMIC);
return (*sample_pci_saved_attach)(id, isc);
}

/* set up rest of stuff e.g., isrlink
isc_claim( isc, &wsio_sample_drv_info );
return (*sample_pci_saved_attach)(id, isc);
}

SEE ALSO

dma_sync(CDIO3), init_map_context(CDIO3), pci_errata(PCI5),
wsio_fastmap(WSIO3), wsio_remap(WSIO3),
wsio_set_attributes(WSIO3), wsio_unmap(WSIO3)
Deprecated WSIO Interfaces

Functions

NAME

*wsio_register_probe_func*(WSIO3) – Insert a driver-specified probe function into the global probe list

SYNOPSIS

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

void wsio_register_probe_func (int (*func)(), char *if_class);
```

PARAMETERS

- **func**: The name of your probe function.
- **if_class**: The string that will be matched with the *drv_path* field of driver *wsio_drv_data_t* structs registered during *driver_install()*.

DESCRIPTION

The WSIO service *wsio_register_probe_func()* inserts a driver-specified probe function into the global probe list. The underlying probe routines match your *if_class* string with driver *drv_path* strings to determine which probe to use. These routines match, character by character, up to a space or underline character, at which point the match succeeds. Note that the maximum string length is limited to 16 characters. For example, if *scsi* was passed in as the *if_class* parameter when registering your probe function, it would successfully match *scsi_ctl* in the *drv_path* field of the *scsi_ctl* driver's *wsio_drv_data_t* struct.

RETURN VALUES

None.

CONSTRAINTS
SEE ALSO

\texttt{driver\_class\_probe(WSIO\_DRV)}, \texttt{driver\_addr\_probe(WSIO\_DRV)},
\texttt{wsio\_drv\_data\_t(WSIO4)}
Deprecated WSIO Interfaces

Functions

NAME

wsio_remap (WSIO3) – Map a host range into a pre-mapped I/O Virtual Address range

SYNOPSIS

#include <sys/dma.h>

int wsio_remap (struct isc_table_type *isc, int range_type,
                struct iovec *host_range, struct iovec *io_range);

PARAMETERS

isc An isc_table_type structure used to get interface and CDIO information.

range_type The type of host address for host_range. If range_type > 0 then this is the space ID of the address range. Other values are:

KERNELSPACE Kernel virtual buffer (same as space ID = 0)

host_range A pointer/length pair, indicating the host address range of type range_type. The length will be modified by the service to indicate bytes remaining to be mapped.

io_range A pointer/length pair, which was filled by a previous call to wsio_map(), wsio_fastmap(), or wsio_remap(). wsio_remap() will map the new host_range into this range.

DESCRIPTION

The wsio_remap() WSIO function is normally used by CDIOs such as EISA which have to manage CDIO specific map registers. It is generally not used by driver writers.
The `wsio_remap()` WSIO function maps a pre-allocated I/O virtual address to new `host_ranges`. The `io_range` must use exactly the same number of mapping resources as the previous mapping. This can be ensured by making sure the buffers are page-aligned and equal sizes.

Cache-line fragments are ignored (IO_IGN_ALIGNMENT is assumed).

Since the behavior and side effects are different for coherent and noncoherent systems, these are discussed separately below. Although the behaviors are slightly different, drivers should not have dependencies on these differences.

All mappings remain valid until the `io_range` is unmapped via calls to `wsio_unmap()`. If the `io_range` is reused, `dma_sync_IO()` must be used to resynchronize it.

**Behavior on Noncoherent Systems**

On noncoherent systems, the I/O virtual address is equivalent to the host physical address. This will likely not be the same as the I/O virtual address provided by the caller. Therefore, `wsio_remap()` will fill `io_range` with the new address range. If the mapping is for an EISA module, then the `io_range` is filled with EISA addresses and the EISA map will contain the host physical address.

**Behavior on Coherent Systems**

On coherent systems, the I/O virtual address is remapped to point to the new `host_range`. The page type bits are not modified.

**RETURN VALUES**

- **0** The range was fully mapped.
- **-1** The necessary resources could not be obtained.

**CONSTRAINTS**

**SEE ALSO**

`dma_sync(CDIO3), wsio_fastmap(WSIO3), wsio_map(WSIO3), wsio_remap(WSIO3), wsio_unmap(WSIO3)`
Deprecated WSIO Interfaces

**NAME**

`wsio_set_attributes` (WSIO3) – Set map function attributes

**SYNOPSIS**

```c
#include <sys/dma.h>

void wsio_set_attributes (struct isc_table_type *isc, int attributes);
```

**PARAMETERS**

- `isc` Pointer to an ISC structure.
- `attributes` Attributes which change the default behavior of `wsio_map()`, `wsio_fastmap()`, `wsio_remap()`, and `wsio_unmap()`. More than one attribute can be set by ORing them together. The following attributes are defined:
  - `IO_DEBUG_DMA` Turns on additional checks in mapping services. This should be used for debugging only. The services will call `panic()` if any problems are detected.
  - `IO_INTERLEAVED_DMA` The device is likely to interleave many I/O requests. Mass storage devices are an example of this type of device. This is the default behavior.
  - `IO_NONINTERLEAVED_DMA` The device is likely to satisfy a single I/O request at a time. This is typical of networking devices. This attribute cannot be specified with `IO_INTERLEAVED_DMA`.

Even if this attribute is set, the services will behave as if they are `IO_INTERLEAVED_DMA` if the `map_cb` argument is non-NULL in calls to `wsio_map()`. Networking cards are typical of noninterleaved devices.
For small buffers that won’t cross a page boundary, and page size in length.

DESCRIPTION

The `wsio_set_attributes()` WSIO function alters the default behavior of `wsio_fastmap()`, `wsio_map()`, `wsio_remap()`, and `wsio_unmap()`.

RETURN VALUES

`wsio_set_attributes()` is a `void` function.

CONSTRAINTS

SEE ALSO

`panic(KER2)`, `wsio_fastmap(WSIO3)`, `wsio_map(WSIO3)`, `wsio_remap(WSIO3)`, `wsio_unmap(WSIO3)`
Deprecated WSIO Interfaces

Functions

NAME

*wsio_unmap*(WSIO3) – Unmap an I/O virtual address range

SYNOPSIS

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

void wsio_unmap (struct isc_table_type *isc, struct iovec *io_range);
```

PARAMETERS

- **isc**
  A pointer to the *isc* struct holding bus-specific information used by the mapping services.

- **io_range**
  A pointer/length pair, representing the I/O virtual range to be unmapped.

DESCRIPTION

On coherent systems, the resources associated with the mapping are released. On noncoherent systems, this function does nothing. In addition to *wsio_unmap()* the caller must call *dma_sync_IO()* during post-DMA cleanup for inbound data.

When multiple objects (I/Os) are mapped with a single map context, *wsio_unmap()* must not be called for ANY of the mapped objects until ALL the I/Os for that context have completed. Failure to ensure that all I/Os have completed may result in data corruption.

RETURN VALUES

None.

CONSTRAINTS
EXAMPLE

The following function cleans up after an inbound DMA:

```c
#define SYNC() dma_sync(0,0,0,0)

void my_inbound_dma_cleanup(isc, vec_cnt, host_vec, dma_desc)
struct isc_table_type *isc;
int vec_cnt;
struct iovec *host_vec;
struct my_dma_type *dma_desc;
{
    struct iovec *io_vec;
    int dma_cnt;

    /* Point to the beginning of the DMA Vector area */
    io_vec = &dma_desc->iov;

    /* Unmap each DMA vector */
    for (dma_cnt=dma_desc->iov_cnt; dma_cnt>0; dma_cnt, 
      io_vec++) wsio_unmap(my_isc, io_vec);

    /* Inbound data so synchronize each host range */
    for (; vec_cnt; host_vec++)
      dma_sync(KERNELSPACE, host_vec->iov_base,
               host_vec->iov_len,
               IO_SYNC_FORCPU|IO_NO_SYNC|IO_PREFETCHED);

    SYNC();
    FREE(dma_desc);
}
```

SEE ALSO
dma_sync(CDIO3), init_map_context(CDIO3), wsio_fastmap(WSIO3), 
wsio_map(WSIO3), wsio_remap(WSIO3)
Deprecated WSIO Interfaces

Functions