This chapter contains reference pages for driver support routines that are external to all CDIOs.
Functions, Structures and Macro Commands
NAME

dma_sync(CDIO3) – Synchronize processor and device views of memory

SYNOPSIS

#include <sys/dma.h>

#define dma_sync (Addr_Type, Addr, Length, Hints)

PARAMETERS

Addr-type

Space ID corresponding to Addr.

Addr

Virtual address (processor view) of memory object.

Length

Size of the memory object, in bytes, pointed to by addr.

Hints

Bit-wise OR of hints that change the behavior of dma_sync(). If no hints are given, the call results in a SYNC instruction on noncoherent platforms and a SYNCDMA on coherent and semicoherent platforms. The defined hints are:

- IO_ACCESSED  Perform function only if the page has been accessed by a processor.
- IO_MODIFIED  Perform function only if the page has been modified by a processor.
- IO_NO_SYNC   Inhibit execution of SYNC or SYNCDMA instructions.
- IO_PREFETCHED  Perform function only if the processor prefetches data.
- IO_READ  Purge processor caches for inbound data on noncoherent systems.
- IO_SYNC_FORCPU  Same as IO_READ.
- IO_SYNC_FORDEV  Same as IO_WRITE.
- IO_SYNC_MEM  Synchronize processor caches with host memory: caches are flushed to memory when used with IO_WRITE (even on coherent platforms).
DESCRIPTION

The dma_sync() CDIO macro has been superseded by the dma_sync_IO() function. New drivers are encouraged to call dma_sync_IO() which provides the following benefits:

- Eliminates the overhead of the SYNCDMA instruction on semicoherent platforms when the IO_READ_START hint is used. dma_sync() does not recognize the IO_READ_START hint.
- Eliminates the overhead of the SYNCDMA instruction on fully coherent platforms. dma_sync() does not distinguish between coherent and semicoherent platforms and issues unnecessary SYNCDMA instructions on coherent platforms.

Legacy drivers call dma_sync() to synchronize the processor caches with DMA transactions mastered by their devices.

There are three cases to consider where drivers must call dma_sync(). These cases are prior to starting a write transaction, prior to starting a read transaction and after completing a read transaction.

- Prior to starting a write transaction:
  
  For each buffer that is to be written out, the driver must call dma_sync() with the IO_WRITE hint set. On noncoherent platforms, this will cause the associated processor caches to be flushed. For all but the last buffer, the IO_NO_SYNC hint should also be set to reduce the performance penalty of synchronizing the cache flushes on noncoherent platforms.

- Prior to starting a read transaction:
  
  For each buffer that is to be read into, the driver must call dma_sync() with the IO_READ hint set. On noncoherent platforms, this will cause the associated processor caches to be purged. For all but the last buffers, the IO_NO_SYNC hint should also be set to reduce the performance penalty of synchronizing the cache purges on noncoherent platforms.
After completing a read transaction:

For each buffer that has been read into, the drive must call `dma_sync()` with the `IO_READ` hint set. On noncurrent platforms, this will cause the associated processor caches to be purged of data that may have been perfected. For all but the last buffer, the `IO_NO_SYNC` hint should also be set to reduce the performance penalty of synchronizing the cache purges on noncoherent platforms. On semicoherent platforms, the processor caches will be made to synchronize with the data read when the `IO_NO_SYNC` hint is not set.

**CONSTRAINTS**

**WARNINGS**

Do not use the `IO_READ_START` hint with `dma_sync()`. `IO_READ_START` is new to `dma_sync_IO()`.

**SEE ALSO**

`dma_sync_IO(CDIO3)`
NAME
dma_sync_IO(CDIO3) – Synchronize processor and device views of memory.

SYNOPSIS

void dma_sync_IO (uint32_t addr_type, void * addr, int length, uint32_t hints)

PARAMETERS

addr-type
addr
length
hints

Space ID corresponding to addr.
Virtual address (processor view) of memory object.
Size of the memory object, in bytes, pointed to by addr.
Bit-wise OR of hints that change the behavior of
dma_sync(). If no hints are given, the call results in a
SYNC instruction on noncoherent platforms and a
SYNCDMA on semicoherent platforms; nothing is done on
fully coherent platforms. The defined hints are:
IO_ACCESSED Perform function only if the page has
been accessed by a processor.
IO_MODIFIED Perform function only if the page has
been modified by a processor.
IO_NO_SYNC Inhibit execution of SYNC or SYNCDMA
instructions.
IO_PREFETCHED Perform function only if the processor
prefetches data.
IO_READ Purge processor caches for inbound
data on noncoherent platforms. Done
after completing the DMA data
transfer.
IO_READ_START Purge processor caches for inbound
data on noncoherent platforms and
inhibit the SYNCDMA instruction on
semicoherent platforms. Done prior
to starting the DMA data transfer.
Drivers call `dma_sync_IO()` to synchronize the processor caches with DMA transactions mastered by their devices. `dma_sync_IO()` is sensitive to the underlying coherency of the platform. If the platform is coherent, `dma_sync_IO()` does nothing; the hardware provides the coherency functionality. If the platform is semicoherent, `dma_sync_iO()` handles the special case where the processor caches must be synchronized with data that have been read into host memory. If the platform is noncoherent, `dma_sync_iO()` flushes (or purges) and synchronizes the processor caches to maintain a consistent view of memory between processors and devices.

There are three cases to consider where drivers must call `dma_sync_IO()`. These cases are prior to starting a write transaction, prior to starting a read transaction and after completing a read transaction.

- Prior to starting a write transaction:
  
  For each buffer that is to be written out, the driver must call `dma_sync_IO()` with the `IO_WRITE` hint set. On noncoherent platforms, this will cause the associated processor caches to be flushed. For all but the last buffer, the `IO_NO_SYNC` hint should also be set to reduce the performance penalty of synchronizing the cache flushes on noncoherent platforms.

- Prior to starting a read transaction:
  
  For each buffer that is to be read into, the driver must call `dma_sync_IO()` with the `IO_READ_START` hint set. On noncoherent platforms, this will cause the associated processor caches to be...
purged. For all but the last buffers, the \texttt{IO\_NO\_SYNC} hint should also be set to reduce the performance penalty of synchronizing the cache purges on noncoherent platforms.

After completing a read transaction:

For each buffer that has been read into, the drive must call \texttt{dma\_sync\_IO()} with the \texttt{IO\_READ} hint set. On noncoherent platforms, this will cause the associated processor caches to be purged of data that may have been prefetched. For all but the last buffer, the \texttt{IO\_NO\_SYNC} hint should also be set to reduce the performance penalty of synchronizing the cache purges on noncoherent platforms. On semicoherent platforms, the processor caches will be made to synchronize with the data read when the \texttt{IO\_NO\_SYNC} hint is not set.

**CONSTRAINTS**

**SEE ALSO**

\texttt{dma\_sync(CDIO3)}
NAME

drv_info(CDIO4) – Driver information structure

SYNOPSIS

#include <sys/conf.h>

PARAMETERS

typedef struct drv_info
{
    char   *name;    /* Name of driver */
    char   *class;   /* Device class (see below)*/
    ubit32 flags;   /* Device flags (see below)*/
    int    b_major; /* Block device major number */
    int    c_major; /* Character device major number */
    cdio_t *cdio;   /* Drivers set this to NULL */
    void   *gio_private; /* Drivers set this to NULL */
    void   *cdio_private; /* Drivers set this to NULL */
} drv_info_t;

DESCRIPTION

All CDIOs use the driver-specific fields in the drv_info_t CDIO structure type, defined in <sys/conf.h>, to describe certain parameters of the driver. A drv_info_t structure must be statically allocated.

The relevant fields are described below. All other fields in a drv_info_t should be NULL.
### STRUCTURE MEMBERS

<table>
<thead>
<tr>
<th>Field</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Pointer to a string containing the name of the driver. This is the name you use in the system file (usually <code>/stand/system</code>) in the $DRIVER_INSTALL section of a master file in <code>/usr/conf/master.d</code>, and as the value for <code>driver</code> in the <code>driver_install()</code> function name. See <code>config (1M)</code> and <code>master (4)</code>. The current implementation of kernel functions that access <code>name</code> require that the string be less than 16 characters long.</td>
</tr>
<tr>
<td>class</td>
<td>Pointer to a string containing the name of the class that the driver is in. Interface drivers typically use <code>ext_bus</code>. Device drivers use classes that describe the general type of device, e.g., <code>disk</code>, <code>tape</code>, <code>pseudo</code>, etc. For interface drivers, instances of a card are enumerated within each class as they are identified by the kernel at boot time.</td>
</tr>
<tr>
<td>flags</td>
<td>The bit-wise OR of flag values that describe the driver, taken from:</td>
</tr>
<tr>
<td></td>
<td><strong>DRV_CHAR</strong> Character device driver.</td>
</tr>
<tr>
<td></td>
<td><strong>DRV_BLOCK</strong> Block device driver.</td>
</tr>
<tr>
<td></td>
<td><strong>DRV_PSEUDO</strong> Pseudo driver.</td>
</tr>
<tr>
<td></td>
<td><strong>DRV_SCAN</strong> Driver supports bus scanning.</td>
</tr>
<tr>
<td></td>
<td><strong>DRV_MP_SAFE</strong> Driver provides its own multiprocessing protection. This flag and C_MGR_IS_MP in drv_ops_t must be consistent or the kernel services will treat the driver as if it were not MPSAFE.</td>
</tr>
<tr>
<td></td>
<td><strong>DRV_SAVE_CONF</strong> Save configuration information to <code>/etc/ioconfig</code>. This file retains potentially volatile information, such as dynamic major numbers and card instance numbers, across reboots.</td>
</tr>
<tr>
<td>b_major</td>
<td>The major number if this is a block device. Set it to -1 for dynamic assignment or if it is not a block device.</td>
</tr>
</tbody>
</table>
c_major

The major number if this is a character device. Set it to -1 for dynamic assignment or if it is not a character device.

NOTE

The values you specify above for b_major and c_major override the values you enter in a master file in /usr/conf/master.d (see master (4)).

SEE ALSO

config(1M), driver_install(WSIO_DRV), drv_ops(CDIO4), wsio_drv_info(WSIO4), master(4)
NAME

\texttt{drv\_ops}(CDIO4) – Structure to specify driver entry points

SYNOPSIS

\texttt{
#include <sys/conf.h>
}

PARAMETERS

\texttt{
typedef struct drv\_ops
{
    int (*d\_open)(); /* block and character */
    int (*d\_close)(); /* block and character */
    int (*d\_strategy)(); /* block */
    int (*d\_dump)(); /* NULL (obsolete) */
    int (*d\_psize)(); /* block */
    int (*reserved0)(); /* NULL */
    int (*d\_read)(); /* character */
    int (*d\_write)(); /* character */
    int (*d\_ioctl)(); /* character */
    int (*d\_select)(); /* character */
    int (*d\_option1)(); /* NULL */
    pfilter\_t \*pfilter; /* block and character */
    int (*reserved1)(); /* NULL */
    int (*reserved2)(); /* NULL */
    int (*reserved3)(); /* NULL */
    int d\_flags; /* block and character */
} drv\_ops\_t;
}

DESCRIPTION

The \texttt{drv\_ops\_t} CDIO structure type, defined in \texttt{<sys/conf.h>}, contains pointers to all driver entry points. A \texttt{drv\_ops\_t} structure must be statically allocated.

The relevant fields are described in the STRUCTURE MEMBERS section. All other fields in \texttt{drv\_ops\_t} should be \texttt{NULL}. Except as noted, entry points that don't apply to your driver or that your driver does not provide should be \texttt{NULL} (for example, \texttt{d\_read()} has no meaning for a printer).
STRUCTURE MEMBERS

The Device Type column indicates whether the field applies to character-only, block-only, or both types of drivers.

Table 3-1 Device Driver Fields in drv_ops_t Structure Type

<table>
<thead>
<tr>
<th>Field</th>
<th>Device Type</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>d_open()</td>
<td>both</td>
<td>Pointer to your driver_open() routine, which enables a device for subsequent operations.</td>
</tr>
<tr>
<td>d_close()</td>
<td>both</td>
<td>Pointer to your driver_close() routine, which performs the tasks required when a device is closed.</td>
</tr>
<tr>
<td>d_strategy()</td>
<td>block</td>
<td>Pointer to your driver_strategy() routine, which queues I/O requests for either reading or writing.</td>
</tr>
<tr>
<td>d_psize()</td>
<td>block</td>
<td>Pointer to your driver_psize() routine. For a swapping device, it should return the size of the swap partition.</td>
</tr>
<tr>
<td>d_read()</td>
<td>character</td>
<td>Pointer to your driver_read() routine, which should return the requested data transferred from the device.</td>
</tr>
<tr>
<td>d_write()</td>
<td>character</td>
<td>Pointer to your driver_write() routine, which should write the requested data to the device.</td>
</tr>
<tr>
<td>d_ioctl()</td>
<td>character</td>
<td>Pointer to your driver_ioctl() routine, which sends control information to, or gets it from, a device.</td>
</tr>
</tbody>
</table>
Table 3-1  Device Driver Fields in `drv_ops_t` Structure Type (Continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Device Type</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>d_select()</td>
<td>character</td>
<td>Pointer to your driver_select() routine, which you can use to test for I/O completion and driver-dependent exception conditions. If your device is always ready for reading or writing, you can put seltrue in the d_select() field. If you do, calls to select() always return true without invoking your driver.</td>
</tr>
<tr>
<td>pfilter</td>
<td>both</td>
<td>Pointer to a pfilter_t structure. Use the &amp;cpd_pfilter pointer. This structure provides backward compatible routines for disk structures with fixed partitions, such as the Series 800 computers before the availability of the Logical Volume Manager (LVM). The &amp;cpd_pfilter pointer is required for such disks; it is ignored under other conditions (or you can use NULL).</td>
</tr>
<tr>
<td>d_flags</td>
<td>both</td>
<td>The bit-wise OR of flag values that indicate special features of the device. The flags give information about the device to the kernel. Drivers receive this information, but usually only validate it. Use 0 if no flags are set.</td>
</tr>
</tbody>
</table>

The flag bit defines for d_flags are:

- **C_ALLCLOSES**  Force a call to driver_close() on every closing of the device. (The default action is to call the driver's close routine only on the last close of the device.)
C_NODELAY

Tell the kernel to not wait for a write request to complete on this device. The default action is to wait for a write request to complete before returning control to the calling process.

C_MGR_IS_MP

Identify the driver as safe for use in a multiprocessing environment. This flag and the DRV_MP_SAFE flag in drv_info_t must be consistent or the kernel services will treat the driver as if it were not MP SAFE.

C_MAP_BUFFER_TO_KERNEL

Identify that the device driver needs physio() to remap a user buffer to kernel space prior to calling the driver strategy() routine. This flag also identifies that after the associated buf structure has been marked iodone, physio() will remap the buffer to user space.

SEE ALSO

driver_close(WSIO_DRV), driver_ioctl(WSIO_DRV),
driver_open(WSIO_DRV), driver_psize(WSIO_DRV),
driver_read(WSIO_DRV), driver_select(WSIO_DRV),
driver_strategy(WSIO_DRV), driver_write(WSIO_DRV),
wsio_drv_info(WSIO_DRV), drv_info(CDIO4), physio(KER2),
select(2)
**NAME**

`install_driver(CDIO3)` – Install a driver's header structure into the CDIO.

**SYNOPSIS**

```c
#include <gio.h>

int install_driver (drv_info_t *drv_info, drv_ops_t *drv_ops);
```

**PARAMETERS**

- `drv_info` Pointer to the driver's `drv_info_t` structure.
- `drv_ops` Pointer to the driver's `drv_ops_t` structure.

**DESCRIPTION**

The `install_driver()` CDIO function installs a driver's header structure outside any specific CDIO, typically for pseudo drivers. WSIO drivers must call `wsio_install_driver (WSIO3)`.

**RETURN VALUES**

- 0 Successful completion.
- -1 Error. The major number specified for the driver is already in use. The following message is displayed on the system console and in the error-log file:

```
install_driver: Install of driver drv-info->name failed.
```

**CONSTRAINTS**

**SEE ALSO**

`wsio_install_driver (WSIO3)`