NAME

ddi_device_acc_attr – data access attributes structure

SYNOPSIS

#include <sys/ddi.h>
#include <sys/sunddi.h>

INTERFACE LEVEL

Solaris DDI specific (Solaris DDI).

DESCRIPTION

The **ddi_device_acc_attr** structure describes the data access characteristics and requirements of the device.

STRUCTURE MEMBERS

```
ushort_t    devacc_attr_version;
uchar_t     devacc_attr_endian_flags;
uchar_t     devacc_attr_dataorder;
```

The **devacc_attr_version** member identifies the version number of this structure. The current version number is **DDI_DEVICE_ATTR_V0**.

The **devacc_attr_endian_flags** member describes the endian characteristics of the device. Specify one of the following values.

DDI_NEVERSWAP_ACC
: data access with no byte swapping.

DDI_STRUCTURE_BE_ACC
: structural data access in big endian format.

DDI_STRUCTURE_LE_ACC
: structural data access in little endian format.

DDI_STRUCTURE_BE_ACC and **DDI_STRUCTURE_LE_ACC** describes the endian characteristics of the device as big endian or little endian, respectively. Even though most of the devices will have the same endian characteristics as their buses, there are examples of devices with I/O an processor that has opposite endian characteristics of the buses. When **DDI_STRUCTURE_BE_ACC** or **DDI_STRUCTURE_LE_ACC** is set, byte swapping will automatically be performed by the system if the host machine and the device data formats have opposite endian characteristics. The implementation may take advantage of hardware platform byte swapping capabilities.

When **DDI_NEVERSWAP_ACC** is specified, byte swapping will not be invoked in the data access functions.

The **devacc_attr_dataorder** member describes order in which the CPU will reference data. Specify one of the following values.

DDI_STRICTORDER_ACC
: The data references must be issued by a CPU in program order. Strict ordering is the default behavior.

DDI_UNORDERED_OK_ACC
: The CPU may re-order the data references. This includes all kinds of re-ordering. (i.e. a load followed by a store may be replaced by a store followed by a load).

DDI_MERGING_OK_ACC
The CPU may merge individual stores to consecutive locations. For example, the CPU may turn two consecutive byte stores into one halfword store. It may also batch individual loads. For example, the CPU may turn two consecutive byte loads into one halfword load. **DDI_MERGING_OK_ACC** also implies re-ordering.

DDI_LOADCACHING_OK_ACC
The CPU may cache the data it fetches and reuse it until another store occurs. The default behavior is to fetch new data on every load. **DDI_LOADCACHING_OK_ACC** also implies merging and re-ordering.

DDI_STORECACHING_OK_ACC
The CPU may keep the data in the cache and push it to the device (perhaps with other data) at a later time. The default behavior is to push the data right away. **DDI_STORECACHING_OK_ACC** also implies load caching, merging, and re-ordering.

These values are advisory, not mandatory. For example, data can be ordered without being merged or cached, even though a driver requests unordered, merged and cached together.

EXAMPLES

The following examples illustrate the use of device register address mapping setup functions and different data access functions.

Example 1

This example demonstrates the use of the **ddi_device_acc_attr** structure in **ddi_regs_map_setup**(9F). It also shows the use of **ddi_getw**(9F) and **ddi_putw**(9F) functions in accessing the register contents.

```
dev_info_t *dip;
uint_t    rnumber;
ushort_t  *dev_addr;
offset_t  offset;
offset_t  len;
ushort_t  dev_command;
ddi_device_acc_attr_t dev_attr;
ddi_acc_handle_t handle;

...

/*
 * setup the device attribute structure for little endian,
 * strict ordering and 16-bit word access.
 */
dev_attr.devacc_attr_version = DDI_DEVICE_ATTR_V0;
dev_attr.devacc_attr_endian_flags = DDI_STRUCTURE_LE_ACC;
dev_attr.devacc_attr_dataorder = DDI_STRICTORDER_ACC;
```

```
/*
 * set up the device registers address mapping
 */
ddi_regs_map_setup(dip, rnumber, (caddr_t *)&dev_addr, offset, len,
        &dev_attr, &handle);

/* read a 16-bit word command register from the device*/
dev_command = ddi_getw(handle, dev_addr);

dev_command |= DEV_INTR_ENABLE;
/* store a new value back to the device command register*/
ddi_putw(handle, dev_addr, dev_command);
```

Example 2 The following example illustrates the steps used to access a device with different apertures. We assume that several apertures are grouped under one single "reg" entry. For example, the sample device has four different apertures each 32K in size. The apertures represent YUV little-endian, YUV big-endian, RGB little-endian, and RGB big-endian. This sample device uses entry 1 of the "reg" property list for this purpose. The size of the address space is 128K with each 32K range as a separate aperture. In the register mapping setup function, the sample driver uses the *offset* and *len* parameters to specify one of the apertures.

```
ulong_t *dev_addr;
ddi_device_acc_attr_t dev_attr;
ddi_acc_handle_t handle;
uchar_t buf[256];

. . .

/*
 * setup the device attribute structure for never swap,
 * unordered and 32-bit word access.
 */
dev_attr.devacc_attr_version = DDI_DEVICE_ATTR_V0;
dev_attr.devacc_attr_endian_flags = DDI_NEVERSWAP_ACC;
dev_attr.devacc_attr_dataorder = DDI_UNORDERED_OK_ACC;

/*
 * map in the RGB big-endian aperture
 * while running in a big endian machine
 *  - offset 96K and len 32K
 */
ddi_regs_map_setup(dip, 1, (caddr_t *)&dev_addr, 96*1024, 32*1024,
        &dev_attr, &handle);

/*
```

```
 * Write to the screen buffer
 *  first 1K bytes words, each size 4 bytes
 */
ddi_rep_putl(handle, buf, dev_addr, 256, DDI_DEV_AUTOINCR);
```

Example 3 The following example illustrates the use of the functions that explicitly call out the data word size to override the data size in the device attribute structure.

```
struct device_blk {
        ushort   d_command;      /* command register */
        ushort   d_status;       /* status register */
        ulong    d_data;         /* data register */
} *dev_blkp;

dev_info_t *dip;
caddr_t  dev_addr;
ddi_device_acc_attr_t dev_attr;
ddi_acc_handle_t handle;
uchar_t buf[256];

. . .

/*
 * setup the device attribute structure for never swap,
 * strict ordering and 32-bit word access.
 */
dev_attr.devacc_attr_version = DDI_DEVICE_ATTR_V0;
dev_attr.devacc_attr_endian_flags = DDI_NEVERSWAP_ACC;
dev_attr.devacc_attr_dataorder= DDI_STRICTORDER_ACC;

ddi_regs_map_setup(dip, 1, (caddr_t *)&dev_blkp, 0, 0,
        &dev_attr, &handle);

/* write command to the 16-bit command register */
ddi_putw(handle, &dev_blkp->d_command, START_XFER);

/* Read the 16-bit status register */
status = ddi_getw(handle, &dev_blkp->d_status);

if (status & DATA_READY)
        /* Read 1K bytes off the 32-bit data register */
        ddi_rep_getl(handle, buf, &dev_blkp->d_data,
                256, DDI_DEV_NO_AUTOINCR);
```

SEE ALSO

ddi_getw(9F), **ddi_putw**(9F), **ddi_regs_map_setup**(9F),
Writing Device Drivers