NAME

nvmem, **nvmem_get_cell_len**, **nvmem_read_cell_by_name**, **nvmem_read_cell_by_idx**, **nvmem_write_cell_by_name**, **nvmem_write_cell_by_idx**

SYNOPSIS

options FDT
device nvmem
#include <sys/extres/nvmem/nvmem.h>

int
nvmem_get_cell_len(*phandle_t node*, *const char *name*);

int
nvmem_read_cell_by_name(*phandle_t node*, *const char *name*, *void *cell*, *size_t buflen*);

int
nvmem_read_cell_by_idx(*phandle_t node*, *int idx*, *void *cell*, *size_t buflen*);

int
nvmem_write_cell_by_name(*phandle_t node*, *const char *name*, *void *cell*, *size_t buflen*);

int
nvmem_write_cell_by_idx(*phandle_t node*, *int idx*, *void *cell*, *size_t buflen*);

DESCRIPTION

On some embedded boards, the manufacturer stored some data on a NVMEM (Non-Volatile Memory), this is generally stored in some eeprom or fuses.

The **nvmem** API consist of helpers functions for consumer and device methods for providers.

FUNCTIONS

nvmem_get_cell_len(*phandle_t node*, *const char *name*)
Get the size of the cell base on the reg property on the node. Return the size or ENOENT if the cell name wasn't found

nvmem_read_cell_by_name(*phandle_t node*, *const char *name*, *void *cell*, *size_t buflen*)
Get the cell content based on the name. Return 0 on sucess or ENOENT if the cell doesn't exists, ENXIO if no provider device was found, EINVAL if the size isn't correct.

nvmem_read_cell_by_idx(*phandle_t node*, *int idx*, *void *cell*, *size_t buflen*)

Get the cell content based on the id. Return 0 on sucess or ENOENT if the cell doesn’t exists, ENXIO if no provider device was found, EINVAL if the size isn’t correct.

nvmem_write_cell_by_name(*phandle_t node*, *const char *name*, *void *cell*, *size_t buflen*)
Write the cell content based on the name. Return 0 on sucess or ENOENT if the cell doesn’t exists, ENXIO if no provider device was found, EINVAL if the size isn’t correct.

nvmem_write_cell_by_idx(*phandle_t node*, *int idx*, *void *cell*, *size_t buflen*)
Write the cell content based on the id. Return 0 on sucess or ENOENT if the cell doesn’t exists, ENXIO if no provider device was found, EINVAL if the size isn’t correct.

DEVICE METHODS

nvmem_read(*device_t dev*, *uint32_t offset*, *uint32_t size*, *uint8_t *buffer*)
Provider device method to read a cell content.

nvmem_write(*device_t dev*, *uint32_t offset*, *uint32_t size*, *uint8_t *buffer*)
Provider device method to write a cell content.

EXAMPLES

Consider this DTS

```
/* Provider */
eeprom: eeprom@20000 {
        board_id: id@0 {
                reg = <0x0 0x4>;
        };
};
/* Consumer */
device@30000 {
        ...

        nvmem-cells = <&board_id>
        nvmem-cell-names = "boardid";
};
```

The device driver for eeprom@20000 needs to expose itself as a provider

```
#include "nvmem_if.h"

int
```

```
foo_nvmem_read(device_t dev, uint32_t offset, uint32_t size, uint8_t *buffer)
{
        /* Read the data */
}

int
foo_attach(device_t dev)
{
        phandle_t node;

        node = ofw_bus_get_node(dev);
        ...
        /* Registering the device so the consumers can find us */
        OF_device_register_xref(OF_xref_from_node(node), dev);

        ...
}

static device_method_t foo_methods[] = {
        ...

        /* nvmem interface */
        DEVMETHOD(nvmem_read, foo_nvmem_read),

        /* Terminate method list */
        DEVMETHOD_END
};
```

The consumer device driver for device@30000 can now read the nvmem data

```
int
bar_attach(device_t dev)
{
        phandle_t node;
        uint32_t boardid;

        ...
        node = ofw_bus_get_node(dev);
        nvmem_read_cell_by_name(node, "boardid", (void *)&boardid, sizeof(boardid));
        ...
}
```

HISTORY

The nvmem related function first appear in FreeBSD 12.0. The nvmem interface and manual page was written by Emmanuel Vadot <*manu@FreeBSD.org*>.