NAME

bhnd_erom, **bhnd_erom_alloc**, **bhnd_erom_dump**, **bhnd_erom_fini_static**, **bhnd_erom_free**, **bhnd_erom_free_core_table**, **bhnd_erom_get_core_table**, **bhnd_erom_init_static**, **bhnd_erom_io**, **bhnd_erom_io_fini**, **bhnd_erom_io_map**, **bhnd_erom_io_read**, **bhnd_erom_iobus_init**, **bhnd_erom_iores_new**, **bhnd_erom_lookup_core**, **bhnd_erom_lookup_core_addr**, **bhnd_erom_probe**, **bhnd_erom_probe_driver_classes** - BHND device enumeration table parsing

SYNOPSIS

#include <dev/bhnd/bhnd.h>
#include <dev/bhnd/bhnd_erom.h>

typedef struct bhnd_erom bhnd_erom_t;
typedef struct kobj_class bhnd_erom_class_t;
typedef struct bhnd_erom_static bhnd_erom_static_t;

int
bhnd_erom_probe(*bhnd_erom_class_t *cls, struct bhnd_erom_io *eio, const struct bhnd_chipid *hint, struct bhnd_chipid *cid*);

*bhnd_erom_class_t **
bhnd_erom_probe_driver_classes(*devclass_t bus_devclass, struct bhnd_erom_io *eio, const struct bhnd_chipid *hint, struct bhnd_chipid *cid*);

*bhnd_erom_t **
bhnd_erom_alloc(*bhnd_erom_class_t *cls, const struct bhnd_chipid *cid, struct bhnd_erom_io *eio*);

void
bhnd_erom_free(*bhnd_erom_t *erom*);

int
bhnd_erom_init_static(*bhnd_erom_class_t *cls, bhnd_erom_t *erom, size_t esize, const struct bhnd_chipid *cid, struct bhnd_erom_io *eio*);

void
bhnd_erom_fini_static(*bhnd_erom_t *erom*);

int
bhnd_erom_dump(*bhnd_erom_t *erom*);

int

bhnd_erom_get_core_table(*bhnd_erom_t *erom, struct bhnd_core_info **cores, u_int *num_cores*);

void
bhnd_erom_free_core_table(*bhnd_erom_t *erom, struct bhnd_core_info *cores*);

int
bhnd_erom_lookup_core(*bhnd_erom_t *erom, const struct bhnd_core_match *desc, struct bhnd_core_info *core*);

int
bhnd_erom_lookup_core_addr(*bhnd_erom_t *erom, const struct bhnd_core_match *desc, bhnd_port_type type, u_int port, u_int region, struct bhnd_core_info *core, bhnd_addr_t *addr, bhnd_size_t *size*);

Bus Space I/O

*struct bhnd_erom_io **
bhnd_erom_iores_new(*device_t dev, int rid*);

int
bhnd_erom_iobus_init(*struct bhnd_erom_iobus *iobus, bhnd_addr_t addr, bhnd_size_t size, bus_space_tag_t bst, bus_space_handle_t bsh*);

void
bhnd_erom_io_fini(*struct bhnd_erom_io *eio*);

int
bhnd_erom_io_map(*struct bhnd_erom_io *eio, bhnd_addr_t addr, bhnd_size_t size*);

uint32_t
bhnd_erom_io_read(*struct bhnd_erom_io *eio, bhnd_size_t offset, u_int width*);

#include <dev/bhnd/bhnd_eromvar.h>

```
struct bhnd_erom_io {
        bhnd_erom_io_map_t      *map;
        bhnd_erom_io_read_t     *read;
        bhnd_erom_io_fini_t     *fini;
};
```

typedef int
(bhnd_erom_io_map_t)(*struct bhnd_erom_io *eio, bhnd_addr_t addr, bhnd_size_t size*);

typedef uint32_t
(bhnd_erom_io_read_t)(*struct bhnd_erom_io *eio, bhnd_size_t offset, u_int width*);

typedef void
(bhnd_erom_io_fini_t)(*struct bhnd_erom_io *eio*);

DESCRIPTION

The **bhnd_erom** framework provides a common parser interface to the BHND device enumeration table formats supported by bhnd(4) bus drivers.

The **bhnd_erom_probe**() function is used to identify a bhnd(4) bus device and determine whether the erom class *cls* is capable of parsing its device enumeration table. If successful, the probed chip identification is written to the location pointed to by *cid*.

A pointer to a bus I/O instance mapping the device registers of the first hardware core must be provided using the *eio* argument. The registers can be mapped using bhnd_erom_io_map(9).

On devices that do not provide standard bhnd_chipc(4) chip identification registers via the first hardware core, a pointer to chip information for the device must be specified using the *hint* argument. Otherwise, the *hint* argument should be NULL.

The **bhnd_erom_probe_driver_classes**() function is a convenience wrapper for **bhnd_erom_probe**(). This function will iterate over all drivers instances in the device class *bus_devclass*, using bhnd_driver_get_erom_class(9) to fetch each driver's erom class and probe the hardware core mapped by *eio*. A pointer to the erom class with the highest probe priority is returned on success. If there are no successful probe results from the erom classes, NULL is returned.

The **bhnd_erom_alloc**() function allocates and returns a new parser instance of the device enumeration class *cls* for the chip identified by *cid*, using the bus I/O instance *eio* to map and read the device table. On success, the returned *bhnd_erom_t* assumes ownership of *eio*.

The **bhnd_erom_free**() function releases all resources held by an erom parser successfully allocated using **bhnd_erom_alloc**().

Clients can manage the allocation of memory themselves with **bhnd_erom_init_static**(). This is useful in cases like performing device enumeration before malloc(9) initialization. **bhnd_erom_init_static**() is called with *erom* set to a pointer to the memory for the instance, and the total available bytes in *esize*.

The *bhnd_erom_static* structure is large enough to statically allocate any supported parser class instance state. Pointers to a *bhnd_erom_static* structure can be cast to *bhnd_erom_t*.

The **bhnd_erom_fini_static**() function releases all resources held by an erom parser successfully initialized using **bhnd_erom_init_static**().

The **bhnd_erom_dump**() function enumerates and prints all device table entries in *erom*.

The **bhnd_erom_get_core_table**() function enumerates all device table entries in *erom*, returning a table of core information structures in *cores* and the count in *num_cores*. The memory allocated for the table must be freed using **bhnd_erom_free_core_table**().

The **bhnd_erom_free_core_table**() function frees any memory allocated in a previous call to **bhnd_erom_get_core_table**().

The **bhnd_erom_lookup_core**() function locates the first device table entry in *erom* that matches core match descriptor *desc*, writing the core information of the matching entry to *core*.

The **bhnd_erom_lookup_core_addr**() function locates the first device table entry in *erom* that matches core match descriptor *desc*, fetching the base address and size of the memory region *region* mapped to the port *port* of type *type*. On success, the core information of the matching entry is written to *core*, the base address of the port region is written to *addr*, and the total size of the port region is written to *size*. If the core information is not desired, set *core* to NULL.

Bus Space I/O

The *bhnd_erom_io* structure provides a set of I/O callbacks used by **bhnd_erom** to map and read the device enumeration table. Clients may either use the existing **bhnd_erom_iores_new**() or **bhnd_erom_iobus_init**() functions to allocate a bus I/O instance, or implement the *bhnd_erom_io* callbacks directly.

The *bhnd_erom_io* structure contains these required fields:

map
: A function implementing **bhnd_erom_io_map**().

read A function implementing **bhnd_erom_io_read**().

fini A function implementing **bhnd_erom_io_fini**().

The **bhnd_erom_iores_new**() function allocates and returns a new bus I/O instance that will perform mapping by using bhnd_alloc_resource(9) to allocate SYS_RES_MEMORY bus resources on demand from the device *dev* using a resource ID of *rid*.

The **bhnd_erom_iobus_init**() function initializes a caller-allocated bus I/O instance *iobus* that will perform bus I/O using the bus space tag *bst* and handle *bsh*. The base address and total size mapped by *bsh* should be specified using the *addr* and *size* arguments.

The **bhnd_erom_io_fini**() function frees all resources held by the bus I/O instance *eio*.

The **bhnd_erom_io_map**() function is used to request that the bus I/O instance *eio* map bhnd(4) bus space at bus address *addr* with a mapping of size *size*.

The **bhnd_erom_io_read**() function is used to read a data item of *width* bytes from the bus I/O instance *eio* at *offset*, relative to the bus address previously mapped using **bhnd_erom_io_map**().

The *width* must be one of 1, 2, or 4 bytes.

RETURN VALUES

The **bhnd_erom_probe**() function returns a standard DEVICE_PROBE(9) result.

A return value equal to or less than zero indicates success. Values greater than zero indicates an error, and will be an appropriate error code. For values less than or equal to zero, the erom class returning the highest value should be used to parse the erom table. ENXIO is returned if the device is not supported by the parser.

The **bhnd_erom_probe_driver_classes**() function returns a pointer to the probed *bhnd_erom_class_t* instance on success, a null pointer otherwise.

The **bhnd_erom_alloc**() function returns a pointer to *bhnd_erom_t* on success, or NULL if an error occurred allocating or initializing the EROM parser.

The **bhnd_erom_init_static**() function returns 0 on success, ENOMEM if the allocation size is smaller than required by the erom class, or an appropriate error code if initialization otherwise fails.

The **bhnd_erom_lookup_core**() function returns 0 on success, ENOENT if no matching core is found, or an appropriate error code if parsing the device table otherwise fails.

The **bhnd_erom_dump**(), **bhnd_erom_get_core_table**(), **bhnd_erom_iobus_init**(), **bhnd_erom_io_map**(), functions return 0 on success, otherwise an appropriate error code is returned.

SEE ALSO

bhnd(4), bhnd(9), bhnd_alloc_resource(9), bhnd_driver_get_erom_class(9), bus_space(9)

AUTHORS

The **bhnd_erom** framework and this manual page were written by Landon Fuller <*landonf@FreeBSD.org*>.