NAME

uftdi - Future Technology Devices International USB to serial UART driver

SYNOPSIS

device usb
device ucom
device uftdi

In rc.conf(5):
kld_list="uftdi"

In sysctl.conf(5):
hw.usb.uftdi.debug=1
hw.usb.uftdi.skip_jtag_interfaces=0

DESCRIPTION

The **uftdi** driver supports FTDI USB to serial UART devices. If the appropriate hardware is detected, the driver will be loaded automatically by devmatch(8). If devmatch(8) is not available, load the driver manually in rc.conf(5) or with kldload(8) at runtime. The device is accessed through the ucom(4) driver which makes it behave like a tty(4).

Call out through this interface with applications like cu(1) or tip(1).

HARDWARE

The **uftdi** driver supports the following USB to serial UART controllers:

- FTDI FT4232H
- FTDI FT232R
- FTDI FT230X
- FTDI FT2232H
- FTDI FT2232D
- FTDI FT2232C
- FTDI FT8U232BM
- FTDI FT8U232AM
- FTDI FT8U100AX

SYSCTL VARIABLES

These settings can be entered in the loader(8) prompt, set in loader.conf(5), sysctl.conf(5), or changed at runtime with sysctl(8):

hw.usb.uftdi.debug Enable debugging messages, default ‘0’

hw.usb.uftdi.skip_jtag_interfaces
Ignore JTAG interfaces, default ‘1’

IOCTLS

Many of the supported chips provide additional functionality such as bitbang mode and the MPSSE engine for serial bus emulation. The **uftdi** driver provides access to that functionality with the following ioctl(2) calls, defined in <*dev/usb/uftdiio.h*>:

UFTDIIOC_RESET_IO (*int*)
Reset the channel to its default configuration, flush RX and TX FIFOs.

UFTDIIOC_RESET_RX (*int*)
Flush the RX FIFO.

UFTDIIOC_RESET_TX (*int*)
Flush the TX FIFO.

UFTDIIOC_SET_BITMODE (*struct uftdi_bitmode*)
Put the channel into the operating mode specified in *mode*, and set the pins indicated by ones in *iomask* to output mode. The *mode* must be one of the *uftdi_bitmodes* values. Setting *mode* to UFTDI_BITMODE_NONE returns the channel to standard UART mode.

```
enum uftdi_bitmodes
{
        UFTDI_BITMODE_ASYNC = 0,
        UFTDI_BITMODE_MPSSE = 1,
        UFTDI_BITMODE_SYNC = 2,
        UFTDI_BITMODE_CPU_EMUL = 3,
        UFTDI_BITMODE_FAST_SERIAL = 4,
        UFTDI_BITMODE_CBUS = 5,
        UFTDI_BITMODE_NONE = 0xff,
};

struct uftdi_bitmode
{
        uint8_t mode;
        uint8_t iomask;
};
```

Manuals and application notes published by FTDI describe these modes in detail. To use most of these modes, you first put the channel into the desired mode, then you read(2) and write(2) data which either reflects pin state or is interpreted as MPSSE commands and parameters, depending on the mode.

UFTDIIOC_GET_BITMODE (*struct uftdi_bitmode*)

Return the current bitbang mode in the *mode* member, and the state of the DBUS0..DBUS7 pins at the time of the call in the *iomask* member. The pin state can be read while the chip is in any mode, including UFTDI_BITMODE_NONE (UART) mode.

UFTDIIOC_SET_ERROR_CHAR (*int*)

Set the character which is inserted into the buffer to mark the point of an error such as FIFO overflow.

UFTDIIOC_SET_EVENT_CHAR (*int*)

Set the character which causes a partial FIFO full of data to be returned immediately even if the FIFO is not full.

UFTDIIOC_SET_LATENCY (*int*)

Set the amount of time to wait for a full FIFO, in milliseconds. If more than this much time elapses without receiving a new character, any characters in the FIFO are returned.

UFTDIIOC_GET_LATENCY (*int*)

Get the current value of the latency timer.

UFTDIIOC_GET_HWREV (*int*)

Get the hardware revision number. This is the *bcdDevice* value from the *usb_device_descriptor*.

UFTDIIOC_READ_EEPROM (*struct uftdi_eeio*)

Read one or more words from the configuration eeprom. The FTDI chip performs eeprom I/O in 16-bit words. Set *offset* and *length* to values evenly divisible by two before the call, and the *data* array will contain the requested values from eeprom after the call.

```
struct uftdi_eeio
{
        uint16_t offset;
        uint16_t length;
        uint16_t data[64];
};
```

The FT232R chip has an internal eeprom. An external serial eeprom is optional on other FTDI chips. The eeprom may contain 64, 128, or 256 words, depending on the part used. Multiple calls may be needed to read or write the larger parts. When no eeprom is present, all words in the returned data are 0xffff. An erased eeprom also reads as all 0xffff.

UFTDIIOC_WRITE_EEPROM (*struct uftdi_eeio*)

Write one or more words to the configuration eeprom. The *uftdi_eeio* values are as described for UFTDIIOC_READ_EEPROM.

The FTDI chip does a blind write to the eeprom, and it will appear to succeed even when no eeprom is present. To ensure a good write you must read back and verify the data. It is *not* necessary to erase before writing. Any position within the eeprom can be overwritten at any time.

UFTDIIOC_ERASE_EEPROM (*int*)

Erase the entire eeprom. This is useful primarily for test and debugging, as there is no need to erase before writing. To help prevent accidental erasure caused by calling the wrong ioctl, you must pass the special value UFTDI_CONFIRM_ERASE as the argument to this ioctl.

FILES

*/dev/ttyU** for callin ports
/dev/ttyU.init*
/dev/ttyU.lock*
corresponding callin initial-state and lock-state devices

*/dev/cuaU** for callout ports
/dev/cuaU.init*
/dev/cuaU.lock*
corresponding callout initial-state and lock-state devices

DIAGNOSTICS

ftdi*: allocating USB transfers failed
ftdi*: skipping JTAG interface...
ftdi*: Warning: unknown FTDI device type...

SEE ALSO

cu(1), tty(4), ucom(4), usb(4)

HISTORY

The **uftdi** driver appeared in FreeBSD 4.8 from NetBSD 1.5.