```
**********************************************************
    2645 Sun Dec 16 10:46:50 2012
new/usr/src/pkg/manifests/driver-serial-usbftdi.mf
3419 usbftdi needs to support the BeagleBone
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
#

#
# The default for payload-bearing actions in this package is to appear in the
# global zone only.  See the include file for greater detail, as well as
# information about overriding the defaults.
#
<include global_zone_only_component>
set name=pkg.fmri value=pkg:/driver/serial/usbftdi@$(PKGVERS)
set name=pkg.description value="FTDI FT232R USB serial driver"
set name=pkg.summary value="FT232R USB UART"
set name=info.classification value=org.opensolaris.category.2008:Drivers/Ports
set name=variant.arch value=$(ARCH)
dir path=kernel group=sys
dir path=kernel/drv group=sys
dir path=kernel/drv/$(ARCH64) group=sys
dir path=usr/share/man
dir path=usr/share/man/man7d
driver name=usbftdi perms="* 0666 root sys" \
    alias=usb403,6001 \
    alias=usb403,cc48 \
    alias=usb403,cc49 \
    alias=usb403,cc4a \
    alias=usb403,e888 \
    alias=usb403,e889 \
    alias=usb403,e88b \
    alias=usb403,e88c \
    alias=usb403,fa00 \
    alias=usb403,fa01 \
    alias=usb403,fa02 \
    alias=usb403,fa03 \
    alias=usb403,fa04 \
    alias=usb403,fc08 \
    alias=usb403,fc09 \
    alias=usb403,fc0b \
    alias=usb403,fc0c \
    alias=usb403,fc0d \
    alias=usb403,fc82 \
```



```
    alias=usb411,00b3 \
    alias=usb7cc,0421 \
    alias=usb856,ac01 \
    alias=usb93c,0601 \
    alias=usb93c,0701 \
    alias=usbif403,6010.config1.1 \
    alias=usbif9e88,9e8f.config1.1
file path=kernel/drv/$(ARCH64)/usbftdi group=sys
$(i386_ONLY)file path=kernel/drv/usbftdi group=sys
file path=kernel/drv/usbftdi.conf group=sys \
    original_name=SUNWuftdi:kernel/drv/usbftdi.conf preserve=true
file path=usr/share/man/man7d/usbftdi.7d
legacy pkg=SUNWuftdi desc="FTDI FT232R USB serial driver" \
    name="FT232R USB UART"
license cr_Sun license=cr_Sun
license lic_CDDL license=lic_CDDL
```

```
**********************************************************
   48130 Sun Dec 16 10:46:51 2012
new/usr/src/uts/common/io/usb/clients/usbser/usbftdi/uftdi_dsd.c
3419 usbftdi needs to support the BeagleBone
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Copyright 2012 Hans Rosenfeld <rosenfeld@grumpf.hope-2000.org>
 */

/*
 * FTDI FT232R USB UART device-specific driver
 *
 * May work on the (many) devices based on earlier versions of the chip.
 */

#include <sys/types.h>
#include <sys/param.h>
#include <sys/conf.h>
#include <sys/stream.h>
#include <sys/strsun.h>
#include <sys/termio.h>
#include <sys/termiox.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>

#define USBDRV_MAJOR_VER        2
#define USBDRV_MINOR_VER        0

#include <sys/usb/usba.h>
#include <sys/usb/usba/usba_types.h>
#include <sys/usb/usba/usba_impl.h>

#include <sys/usb/clients/usbser/usbser_dsdi.h>
#include <sys/usb/clients/usbser/usbftdi/uftdi_var.h>
#include <sys/usb/clients/usbser/usbftdi/uftdi_reg.h>

#include <sys/usb/usbdevs.h>

/*
 * DSD operations
```



```
 */
static int      uftdi_attach(ds_attach_info_t *);
static void     uftdi_detach(ds_hdl_t);
static int      uftdi_register_cb(ds_hdl_t, uint_t, ds_cb_t *);
static void     uftdi_unregister_cb(ds_hdl_t, uint_t);
static int      uftdi_open_port(ds_hdl_t, uint_t);
static int      uftdi_close_port(ds_hdl_t, uint_t);

/* power management */
static int      uftdi_usb_power(ds_hdl_t, int, int, int *);
static int      uftdi_suspend(ds_hdl_t);
static int      uftdi_resume(ds_hdl_t);
static int      uftdi_disconnect(ds_hdl_t);
static int      uftdi_reconnect(ds_hdl_t);

/* standard UART operations */
static int      uftdi_set_port_params(ds_hdl_t, uint_t, ds_port_params_t *);
static int      uftdi_set_modem_ctl(ds_hdl_t, uint_t, int, int);
static int      uftdi_get_modem_ctl(ds_hdl_t, uint_t, int, int *);
static int      uftdi_break_ctl(ds_hdl_t, uint_t, int);

/* data xfer */
static int      uftdi_tx(ds_hdl_t, uint_t, mblk_t *);
static mblk_t   *uftdi_rx(ds_hdl_t, uint_t);
static void     uftdi_stop(ds_hdl_t, uint_t, int);
static void     uftdi_start(ds_hdl_t, uint_t, int);
static int      uftdi_fifo_flush(ds_hdl_t, uint_t, int);
static int      uftdi_fifo_drain(ds_hdl_t, uint_t, int);

/* polled I/O support */
static usb_pipe_handle_t uftdi_out_pipe(ds_hdl_t, uint_t);
static usb_pipe_handle_t uftdi_in_pipe(ds_hdl_t, uint_t);

/*
 * Sub-routines
 */

/* configuration routines */
static void     uftdi_cleanup(uftdi_state_t *, int);
static int      uftdi_dev_attach(uftdi_state_t *);
static int      uftdi_open_hw_port(uftdi_state_t *, int);

/* hotplug */
static int      uftdi_restore_device_state(uftdi_state_t *);
static int      uftdi_restore_port_state(uftdi_state_t *);

/* power management */
static int      uftdi_create_pm_components(uftdi_state_t *);
static void     uftdi_destroy_pm_components(uftdi_state_t *);
static int      uftdi_pm_set_busy(uftdi_state_t *);
static void     uftdi_pm_set_idle(uftdi_state_t *);
static int      uftdi_pwrlvl0(uftdi_state_t *);
static int      uftdi_pwrlvl1(uftdi_state_t *);
static int      uftdi_pwrlvl2(uftdi_state_t *);
static int      uftdi_pwrlvl3(uftdi_state_t *);

/* pipe operations */
static int      uftdi_open_pipes(uftdi_state_t *);
static void     uftdi_close_pipes(uftdi_state_t *);
static void     uftdi_disconnect_pipes(uftdi_state_t *);
static int      uftdi_reconnect_pipes(uftdi_state_t *);

/* pipe callbacks */
static void     uftdi_bulkin_cb(usb_pipe_handle_t, usb_bulk_req_t *);
static void     uftdi_bulkout_cb(usb_pipe_handle_t, usb_bulk_req_t *);
```

```c
/* data transfer routines */
static int      uftdi_rx_start(uftdi_state_t *);
static void     uftdi_tx_start(uftdi_state_t *, int *);
static int      uftdi_send_data(uftdi_state_t *, mblk_t *);
static int      uftdi_wait_tx_drain(uftdi_state_t *, int);

/* vendor-specific commands */
static int      uftdi_cmd_vendor_write0(uftdi_state_t *,
                    uint16_t, uint16_t, uint16_t);

/* misc */
static void     uftdi_put_tail(mblk_t **, mblk_t *);
static void     uftdi_put_head(mblk_t **, mblk_t *);


/*
 * DSD ops structure
 */
ds_ops_t uftdi_ds_ops = {
        DS_OPS_VERSION,
        uftdi_attach,
        uftdi_detach,
        uftdi_register_cb,
        uftdi_unregister_cb,
        uftdi_open_port,
        uftdi_close_port,
        uftdi_usb_power,
        uftdi_suspend,
        uftdi_resume,
        uftdi_disconnect,
        uftdi_reconnect,
        uftdi_set_port_params,
        uftdi_set_modem_ctl,
        uftdi_get_modem_ctl,
        uftdi_break_ctl,
        NULL,                   /* no loopback support */
        uftdi_tx,
        uftdi_rx,
        uftdi_stop,
        uftdi_start,
        uftdi_fifo_flush,
        uftdi_fifo_drain,
        uftdi_out_pipe,
        uftdi_in_pipe
};

/* debug support */
static uint_t   uftdi_errlevel = USB_LOG_L4;
static uint_t   uftdi_errmask = DPRINT_MASK_ALL;
static uint_t   uftdi_instance_debug = (uint_t)-1;
static uint_t   uftdi_attach_unrecognized = B_FALSE;

/*
 * ds_attach
 */
static int
uftdi_attach(ds_attach_info_t *aip)
{
        uftdi_state_t *uf;
        usb_dev_descr_t *dd;
        int recognized;

        uf = kmem_zalloc(sizeof (*uf), KM_SLEEP);
        uf->uf_dip = aip->ai_dip;
        uf->uf_usb_events = aip->ai_usb_events;
        *aip->ai_hdl = (ds_hdl_t)uf;
```



```c
        /* only one port */
        *aip->ai_port_cnt = 1;

        if (usb_client_attach(uf->uf_dip, USBDRV_VERSION, 0) != USB_SUCCESS) {
                uftdi_cleanup(uf, 1);
                return (USB_FAILURE);
        }

        if (usb_get_dev_data(uf->uf_dip,
            &uf->uf_dev_data, USB_PARSE_LVL_IF, 0) != USB_SUCCESS) {
                uftdi_cleanup(uf, 2);
                return (USB_FAILURE);
        }

        uf->uf_hwport = FTDI_PIT_SIOA + uf->uf_dev_data->dev_curr_if;

        mutex_init(&uf->uf_lock, NULL, MUTEX_DRIVER,
            uf->uf_dev_data->dev_iblock_cookie);

        cv_init(&uf->uf_tx_cv, NULL, CV_DRIVER, NULL);

        uf->uf_lh = usb_alloc_log_hdl(uf->uf_dip, "uftdi",
            &uftdi_errlevel, &uftdi_errmask, &uftdi_instance_debug, 0);

        /*
         * This device and its clones has numerous physical instantiations.
         */
        recognized = B_TRUE;
        dd = uf->uf_dev_data->dev_descr;
        switch (dd->idVendor) {
        case USB_VENDOR_FTDI:
                switch (dd->idProduct) {
                case USB_PRODUCT_FTDI_SERIAL_2232C:
                case USB_PRODUCT_FTDI_SERIAL_8U232AM:
                case USB_PRODUCT_FTDI_SEMC_DSS20:
                case USB_PRODUCT_FTDI_CFA_631:
                case USB_PRODUCT_FTDI_CFA_632:
                case USB_PRODUCT_FTDI_CFA_633:
                case USB_PRODUCT_FTDI_CFA_634:
                case USB_PRODUCT_FTDI_CFA_635:
                case USB_PRODUCT_FTDI_USBSERIAL:
                case USB_PRODUCT_FTDI_MX2_3:
                case USB_PRODUCT_FTDI_MX4_5:
                case USB_PRODUCT_FTDI_LK202:
                case USB_PRODUCT_FTDI_LK204:
                case USB_PRODUCT_FTDI_TACTRIX_OPENPORT_13M:
                case USB_PRODUCT_FTDI_TACTRIX_OPENPORT_13S:
                case USB_PRODUCT_FTDI_TACTRIX_OPENPORT_13U:
                case USB_PRODUCT_FTDI_EISCOU:
                case USB_PRODUCT_FTDI_UOPTBR:
                case USB_PRODUCT_FTDI_EMCU2D:
                case USB_PRODUCT_FTDI_PCMSFU:
                case USB_PRODUCT_FTDI_EMCU2H:
                        break;
                default:
                        recognized = B_FALSE;
                        break;
                }
                break;
        case USB_VENDOR_SIIG2:
                switch (dd->idProduct) {
                case USB_PRODUCT_SIIG2_US2308:
                        break;
                default:
                        recognized = B_FALSE;
```

```
                        break;
                }
                break;
        case USB_VENDOR_INTREPIDCS:
                switch (dd->idProduct) {
                case USB_PRODUCT_INTREPIDCS_VALUECAN:
                case USB_PRODUCT_INTREPIDCS_NEOVI:
                        break;
                default:
                        recognized = B_FALSE;
                        break;
                }
                break;
        case USB_VENDOR_BBELECTRONICS:
                switch (dd->idProduct) {
                case USB_PRODUCT_BBELECTRONICS_USOTL4:
                        break;
                default:
                        recognized = B_FALSE;
                        break;
                }
                break;
        case USB_VENDOR_MELCO:
                switch (dd->idProduct) {
                case USB_PRODUCT_MELCO_PCOPRS1:
                        break;
                default:
                        recognized = B_FALSE;
                        break;
                }
                break;
        case USB_VENDOR_MARVELL:
                switch (dd->idProduct) {
                case USB_PRODUCT_MARVELL_SHEEVAPLUG_JTAG:
                        break;
                default:
                        recognized = B_FALSE;
                        break;
                }
                break;
        default:
                recognized = B_FALSE;
                break;
        }

        /*
         * Set 'uftdi_attach_unrecognized' to non-zero to
         * experiment with newer devices ..
         */
        if (!recognized && !uftdi_attach_unrecognized) {
                uftdi_cleanup(uf, 3);
                return (USB_FAILURE);
        }

        USB_DPRINTF_L3(DPRINT_ATTACH, uf->uf_lh,
            "uftdi: matched vendor 0x%x product 0x%x port %d",
            dd->idVendor, dd->idProduct, uf->uf_hwport);

        uf->uf_def_ph = uf->uf_dev_data->dev_default_ph;

        mutex_enter(&uf->uf_lock);
        uf->uf_dev_state = USB_DEV_ONLINE;
        uf->uf_port_state = UFTDI_PORT_CLOSED;
        mutex_exit(&uf->uf_lock);

        if (uftdi_create_pm_components(uf) != USB_SUCCESS) {
```



```
                uftdi_cleanup(uf, 3);
                return (USB_FAILURE);
        }

        if (usb_register_event_cbs(uf->uf_dip,
            uf->uf_usb_events, 0) != USB_SUCCESS) {
                uftdi_cleanup(uf, 4);
                return (USB_FAILURE);
        }

        if (usb_pipe_get_max_bulk_transfer_size(uf->uf_dip,
            &uf->uf_xfer_sz) != USB_SUCCESS) {
                uftdi_cleanup(uf, 5);
                return (USB_FAILURE);
        }

        /*
         * TODO: modern ftdi devices have deeper (and asymmetric)
         * fifos than this minimal 64 bytes .. but how to tell
         * -safely- ?
         */

#define FTDI_MAX_XFERSIZE       64

        if (uf->uf_xfer_sz > FTDI_MAX_XFERSIZE)
                uf->uf_xfer_sz = FTDI_MAX_XFERSIZE;

        if (uftdi_dev_attach(uf) != USB_SUCCESS) {
                uftdi_cleanup(uf, 5);
                return (USB_FAILURE);
        }

        return (USB_SUCCESS);
}
```

______***unchanged_portion_omitted_***

```
/*
 * pipe operations
 */
static int
uftdi_open_pipes(uftdi_state_t *uf)
{
        int ifc, alt;
        usb_pipe_policy_t policy;
        usb_ep_data_t *in_data, *out_data;
        size_t max_xfer_sz;

        /* get max transfer size */
        if (usb_pipe_get_max_bulk_transfer_size(uf->uf_dip, &max_xfer_sz)
            != USB_SUCCESS)
                return (USB_FAILURE);

        /* get ep data */
        ifc = uf->uf_dev_data->dev_curr_if;
        alt = 0;

        in_data = usb_lookup_ep_data(uf->uf_dip, uf->uf_dev_data, ifc, alt,
            0, USB_EP_ATTR_BULK, USB_EP_DIR_IN);

        out_data = usb_lookup_ep_data(uf->uf_dip, uf->uf_dev_data, ifc, alt,
            0, USB_EP_ATTR_BULK, USB_EP_DIR_OUT);

        if (in_data == NULL || out_data == NULL) {
                USB_DPRINTF_L2(DPRINT_ATTACH, uf->uf_lh,
                    "uftdi_open_pipes: can't get ep data");
```

```
                return (USB_FAILURE);
        }

        /*
         * Set buffer sizes. Default to UFTDI_XFER_SZ_MAX.
         * Use wMaxPacketSize from endpoint descriptor if it is nonzero..
         * Cap at a max transfer size of host controller.
         */
        uf->uf_ibuf_sz = uf->uf_obuf_sz = UFTDI_XFER_SZ_MAX;

        if (in_data->ep_descr.wMaxPacketSize)
                uf->uf_ibuf_sz = in_data->ep_descr.wMaxPacketSize;
        uf->uf_ibuf_sz = min(uf->uf_ibuf_sz, max_xfer_sz);

        if (out_data->ep_descr.wMaxPacketSize)
                uf->uf_obuf_sz = out_data->ep_descr.wMaxPacketSize;
        uf->uf_obuf_sz = min(uf->uf_obuf_sz, max_xfer_sz);

        /* open pipes */
        policy.pp_max_async_reqs = 2;

        if (usb_pipe_open(uf->uf_dip, &in_data->ep_descr, &policy,
            USB_FLAGS_SLEEP, &uf->uf_bulkin_ph) != USB_SUCCESS)
                return (USB_FAILURE);

        if (usb_pipe_open(uf->uf_dip, &out_data->ep_descr, &policy,
            USB_FLAGS_SLEEP, &uf->uf_bulkout_ph) != USB_SUCCESS) {
                usb_pipe_close(uf->uf_dip, uf->uf_bulkin_ph, USB_FLAGS_SLEEP,
                    NULL, NULL);
                return (USB_FAILURE);
        }

        mutex_enter(&uf->uf_lock);
        uf->uf_bulkin_state = UFTDI_PIPE_IDLE;
        uf->uf_bulkout_state = UFTDI_PIPE_IDLE;
        mutex_exit(&uf->uf_lock);

        return (USB_SUCCESS);
}
______unchanged_portion_omitted_

/*
 * start receiving data
 */
static int
uftdi_rx_start(uftdi_state_t *uf)
{
        usb_bulk_req_t *br;
        int rval;

        USB_DPRINTF_L4(DPRINT_OUT_PIPE, uf->uf_lh, "uftdi_rx_start");

        ASSERT(mutex_owned(&uf->uf_lock));

        uf->uf_bulkin_state = UFTDI_PIPE_BUSY;
        mutex_exit(&uf->uf_lock);

        br = usb_alloc_bulk_req(uf->uf_dip, uf->uf_ibuf_sz, USB_FLAGS_SLEEP);
        br->bulk_len = uf->uf_ibuf_sz;
        br = usb_alloc_bulk_req(uf->uf_dip, uf->uf_xfer_sz, USB_FLAGS_SLEEP);
        br->bulk_len = uf->uf_xfer_sz;
        br->bulk_timeout = UFTDI_BULKIN_TIMEOUT;
        br->bulk_cb = uftdi_bulkin_cb;
        br->bulk_exc_cb = uftdi_bulkin_cb;
        br->bulk_client_private = (usb_opaque_t)uf;
```



```
        br->bulk_attributes = USB_ATTRS_AUTOCLEARING | USB_ATTRS_SHORT_XFER_OK;

        rval = usb_pipe_bulk_xfer(uf->uf_bulkin_ph, br, 0);

        if (rval != USB_SUCCESS) {
                USB_DPRINTF_L2(DPRINT_IN_PIPE, uf->uf_lh,
                    "uftdi_rx_start: xfer failed %d", rval);
                usb_free_bulk_req(br);
        }

        mutex_enter(&uf->uf_lock);
        if (rval != USB_SUCCESS)
                uf->uf_bulkin_state = UFTDI_PIPE_IDLE;

        return (rval);
}


/*
 * start data transmit
 */
static void
uftdi_tx_start(uftdi_state_t *uf, int *xferd)
{
        int             len;            /* bytes we can transmit */
        mblk_t          *data;          /* data to be transmitted */
        int             data_len;       /* bytes in 'data' */
        mblk_t          *mp;            /* current msgblk */
        int             copylen;        /* bytes copy from 'mp' to 'data' */
        int             rval;

        USB_DPRINTF_L4(DPRINT_OUT_PIPE, uf->uf_lh, "uftdi_tx_start");
        ASSERT(mutex_owned(&uf->uf_lock));
        ASSERT(uf->uf_port_state != UFTDI_PORT_CLOSED);

        if (xferd)
                *xferd = 0;
        if ((uf->uf_port_flags & UFTDI_PORT_TX_STOPPED) ||
            uf->uf_tx_mp == NULL) {
                return;
        }
        if (uf->uf_bulkout_state != UFTDI_PIPE_IDLE) {
                USB_DPRINTF_L4(DPRINT_OUT_PIPE, uf->uf_lh,
                    "uftdi_tx_start: pipe busy");
                return;
        }
        ASSERT(MBLKL(uf->uf_tx_mp) > 0);

        /* send as much data as port can receive */
        len = min(msgdsize(uf->uf_tx_mp), uf->uf_obuf_sz);
        len = min(msgdsize(uf->uf_tx_mp), uf->uf_xfer_sz);

        if (len <= 0)
                return;
        if ((data = allocb(len, BPRI_LO)) == NULL)
                return;

        /*
         * copy no more than 'len' bytes from mblk chain to transmit mblk 'data'
         */
        data_len = 0;
        while (data_len < len && uf->uf_tx_mp) {
                mp = uf->uf_tx_mp;
                copylen = min(MBLKL(mp), len - data_len);
                bcopy(mp->b_rptr, data->b_wptr, copylen);
                mp->b_rptr += copylen;
```

```
		data->b_wptr += copylen;
		data_len += copylen;

		if (MBLKL(mp) < 1) {
			uf->uf_tx_mp = unlinkb(mp);
			freeb(mp);
		} else {
			ASSERT(data_len == len);
		}
	}

	ASSERT(data_len > 0);

	uf->uf_bulkout_state = UFTDI_PIPE_BUSY;
	mutex_exit(&uf->uf_lock);

	rval = uftdi_send_data(uf, data);
	mutex_enter(&uf->uf_lock);

	if (rval != USB_SUCCESS) {
		uf->uf_bulkout_state = UFTDI_PIPE_IDLE;
		uftdi_put_head(&uf->uf_tx_mp, data);
	} else {
		if (xferd)
			*xferd = data_len;
	}
}
```

______***unchanged_portion_omitted_***

```
**********************************************************
    4514 Sun Dec 16 10:46:52 2012
new/usr/src/uts/common/sys/usb/clients/usbser/usbftdi/uftdi_var.h
3419 usbftdi needs to support the BeagleBone
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright 2012 Hans Rosenfeld <rosenfeld@grumpf.hope-2000.org>
 */

#ifndef _USBSER_USBFTDI_UFTDI_VAR_H
#define _USBSER_USBFTDI_UFTDI_VAR_H

/*
 * USB UFTDI definitions
 */

#include <sys/types.h>
#include <sys/dditypes.h>
#include <sys/note.h>

#include <sys/usb/clients/usbser/usbser_dsdi.h>

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * PM support
 */
typedef struct uftdi_pm {
        uint8_t         pm_wakeup_enabled;      /* remote wakeup enabled */
        uint8_t         pm_pwr_states;  /* bit mask of power states */
        boolean_t       pm_raise_power; /* driver is about to raise power */
        uint8_t         pm_cur_power;   /* current power level */
        uint_t          pm_busy_cnt;    /* number of set_busy requests */
} uftdi_pm_t;
______unchanged_portion_omitted_

_NOTE(SCHEME_PROTECTS_DATA("uftdi_regs", uftdi_regs))

/*
 * per device state structure
```



```
 */
typedef struct uftdi_state {
        kmutex_t                uf_lock;                /* structure lock */
        dev_info_t              *uf_dip;                /* device info */
        int                     uf_dev_flags;           /* device flags */
        int                     uf_hwport;              /* hw port number */
        int                     uf_port_state;          /* port state */
        int                     uf_port_flags;          /* port flags */
        ds_cb_t                 uf_cb;                  /* DSD callbacks */

        /*
         * USBA
         */
        usb_client_dev_data_t   *uf_dev_data;           /* registration data */
        usb_event_t             *uf_usb_events;         /* usb events */
        usb_pipe_handle_t       uf_def_ph;              /* default pipe hdl */
        usb_pipe_handle_t       uf_bulkin_ph;           /* in pipe hdl */
        int                     uf_bulkin_state;        /* in pipe state */
        usb_pipe_handle_t       uf_bulkout_ph;          /* in pipe hdl */
        int                     uf_bulkout_state;       /* out pipe state */
        usb_log_handle_t        uf_lh;                  /* USBA log handle */
        int                     uf_dev_state;           /* USB device state */
        size_t                  uf_ibuf_sz;             /* input buffer size */
        size_t                  uf_obuf_sz;             /* output buffer size */
        size_t                  uf_xfer_sz;             /* HCI bulk xfer size */

        uftdi_pm_t              *uf_pm;                 /* PM support */

        /*
         * data receive and transmit
         */
        mblk_t                  *uf_rx_mp;              /* rx data */
        mblk_t                  *uf_tx_mp;              /* tx data */
        kcondvar_t              uf_tx_cv;               /* tx completion */

        /*
         * soft registers
         */
        uftdi_regs_t            uf_softr;       /* config registers */
        uint16_t                uf_mctl;        /* modem control */
        uint8_t                 uf_msr;         /* modem status */
        uint8_t                 uf_lsr;         /* line status register */

} uftdi_state_t;

_NOTE(MUTEX_PROTECTS_DATA(uftdi_state::uf_lock, uftdi_state))
_NOTE(DATA_READABLE_WITHOUT_LOCK(uftdi_state::{
        uf_dip
        uf_dev_data
        uf_usb_events
        uf_def_ph
        uf_lh
        uf_ibuf_sz
        uf_obuf_sz
        uf_xfer_sz
        uf_pm
        uf_port_state
        uf_cb
        uf_bulkin_ph
        uf_bulkout_ph
        uf_hwport
}))
______unchanged_portion_omitted_
#endif

#endif  /* _USBSER_USBFTDI_UFTDI_VAR_H */
```