

new/usr/src/pkg/manifests/driver-serial-usbftdi.mf

1

```
*****
2645 Sun Dec 16 10:46:50 2012
new/usr/src/pkg/manifests/driver-serial-usbftdi.mf
3419 usbftdi needs to support the BeagleBone
*****
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 #
22 #
23 # Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
24 #
25 #
26 #
27 # The default for payload-bearing actions in this package is to appear in the
28 # global zone only. See the include file for greater detail, as well as
29 # information about overriding the defaults.
30 #
31 <include global_zone_only_component>
32 set name=pkg.fmri value=pkg:/driver/serial/usbftdi@(PKGVERS)
33 set name=pkg.description value="FTDI FT232R USB serial driver"
34 set name=pkg.summary value="FT232R USB UART"
35 set name=info.classification value=org.opensolaris.category.2008:Drivers/Ports
36 set name=variant.arch value=$(ARCH)
37 dir path=kernel group=sys
38 dir path=kernel/drv group=sys
39 dir path=kernel/drv/$(ARCH64) group=sys
40 dir path=usr/share/man
41 dir path=usr/share/man/man7d
42 driver name=usbftdi perms="* 0666 root sys" \
43 alias=usb403,6001 \
44 alias=usb403,cc48 \
45 alias=usb403,cc49 \
46 alias=usb403,cc4a \
47 alias=usb403,e888 \
48 alias=usb403,e889 \
49 alias=usb403,e88b \
50 alias=usb403,e88c \
51 alias=usb403,fa00 \
52 alias=usb403,fa01 \
53 alias=usb403,fa02 \
54 alias=usb403,fa03 \
55 alias=usb403,fa04 \
56 alias=usb403,fc08 \
57 alias=usb403,fc09 \
58 alias=usb403,fc0b \
59 alias=usb403,fc0c \
60 alias=usb403,fc0d \
61 alias=usb403,fc82 \
```

new/usr/src/pkg/manifests/driver-serial-usbftdi.mf

2

```
62 alias=usb411,00b3 \
63 alias=usb7cc,0421 \
64 alias=usb856,ac01 \
65 alias=usb93c,0601 \
66 alias=usb93c,0701 \
67 alias=usbif403,6010.config1.1 \
68 alias=usbif9e88,9e8f.config1.1
69 file path=kernel/drv/$(ARCH64)/usbftdi group=sys
70 $(i386_ONLY)file path=kernel/drv/usbftdi group=sys
71 file path=kernel/drv/usbftdi.conf group=sys \
72 original_name=SUNWuftdi:kernel/drv/usbftdi.conf preserve=true
73 file path=usr/share/man/man7d/usbftdi.7d
74 legacy pkg=SUNWuftdi desc="FTDI FT232R USB serial driver" \
75 name="FT232R USB UART"
76 license cr_Sun license=cr_Sun
77 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
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25  */

27 /*
28  * Copyright 2012 Hans Rosenfeld <rosenfeld@grumpf.hope-2000.org>
29  */

31 /*
32  * FTDI FT232R USB UART device-specific driver
33  *
34  * May work on the (many) devices based on earlier versions of the chip.
35  */

37 #include <sys/types.h>
38 #include <sys/param.h>
39 #include <sys/conf.h>
40 #include <sys/stream.h>
41 #include <sys/strsun.h>
42 #include <sys/termio.h>
43 #include <sys/termiox.h>
44 #include <sys/ddi.h>
45 #include <sys/sunddi.h>

47 #define USBDRV_MAJOR_VER    2
48 #define USBDRV_MINOR_VER    0

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

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

58 #include <sys/usb/usbdevs.h>

60 /*
61  * DSD operations

```

```

62 */
63 static int    uftdi_attach(ds_attach_info_t *);
64 static void    uftdi_detach(ds_hdl_t);
65 static int    uftdi_register_cb(ds_hdl_t, uint_t, ds_cb_t *);
66 static void    uftdi_unregister_cb(ds_hdl_t, uint_t);
67 static int    uftdi_open_port(ds_hdl_t, uint_t);
68 static int    uftdi_close_port(ds_hdl_t, uint_t);

70 /* power management */
71 static int    uftdi_usb_power(ds_hdl_t, int, int, int *);
72 static int    uftdi_suspend(ds_hdl_t);
73 static int    uftdi_resume(ds_hdl_t);
74 static int    uftdi_disconnect(ds_hdl_t);
75 static int    uftdi_reconnect(ds_hdl_t);

77 /* standard UART operations */
78 static int    uftdi_set_port_params(ds_hdl_t, uint_t, ds_port_params_t *);
79 static int    uftdi_set_modem_ctl(ds_hdl_t, uint_t, int, int);
80 static int    uftdi_get_modem_ctl(ds_hdl_t, uint_t, int, int *);
81 static int    uftdi_break_ctl(ds_hdl_t, uint_t, int);

83 /* data xfer */
84 static int    uftdi_tx(ds_hdl_t, uint_t, mblk_t *);
85 static mblk_t *uftdi_rx(ds_hdl_t, uint_t);
86 static void    uftdi_stop(ds_hdl_t, uint_t, int);
87 static void    uftdi_start(ds_hdl_t, uint_t, int);
88 static int    uftdi_fifo_flush(ds_hdl_t, uint_t, int);
89 static int    uftdi_fifo_drain(ds_hdl_t, uint_t, int);

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

95 /*
96  * Sub-routines
97  */

99 /* configuration routines */
100 static void    uftdi_cleanup(uftdi_state_t *, int);
101 static int    uftdi_dev_attach(uftdi_state_t *);
102 static int    uftdi_open_hw_port(uftdi_state_t *, int);

104 /* hotplug */
105 static int    uftdi_restore_device_state(uftdi_state_t *);
106 static int    uftdi_restore_port_state(uftdi_state_t *);

108 /* power management */
109 static int    uftdi_create_pm_components(uftdi_state_t *);
110 static void    uftdi_destroy_pm_components(uftdi_state_t *);
111 static int    uftdi_pm_set_busy(uftdi_state_t *);
112 static void    uftdi_pm_set_idle(uftdi_state_t *);
113 static int    uftdi_pwrlv10(uftdi_state_t *);
114 static int    uftdi_pwrlv11(uftdi_state_t *);
115 static int    uftdi_pwrlv12(uftdi_state_t *);
116 static int    uftdi_pwrlv13(uftdi_state_t *);

118 /* pipe operations */
119 static int    uftdi_open_pipes(uftdi_state_t *);
120 static void    uftdi_close_pipes(uftdi_state_t *);
121 static void    uftdi_disconnect_pipes(uftdi_state_t *);
122 static int    uftdi_reconnect_pipes(uftdi_state_t *);

124 /* pipe callbacks */
125 static void    uftdi_bulk_in_cb(usb_pipe_handle_t, usb_bulk_req_t *);
126 static void    uftdi_bulk_out_cb(usb_pipe_handle_t, usb_bulk_req_t *);

```

```

128 /* data transfer routines */
129 static int    uftdi_rx_start(uftdi_state_t *);
130 static void   uftdi_tx_start(uftdi_state_t *, int *);
131 static int    uftdi_send_data(uftdi_state_t *, mblk_t *);
132 static int    uftdi_wait_tx_drain(uftdi_state_t *, int);

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

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

143 /*
144  * DSD ops structure
145  */
146 ds_ops_t uftdi_ds_ops = {
147     DS_OPS_VERSION,
148     uftdi_attach,
149     uftdi_detach,
150     uftdi_register_cb,
151     uftdi_unregister_cb,
152     uftdi_open_port,
153     uftdi_close_port,
154     uftdi_usb_power,
155     uftdi_suspend,
156     uftdi_resume,
157     uftdi_disconnect,
158     uftdi_reconnect,
159     uftdi_set_port_params,
160     uftdi_set_modem_ctl,
161     uftdi_get_modem_ctl,
162     uftdi_break_ctl,
163     NULL, /* no loopback support */
164     uftdi_tx,
165     uftdi_rx,
166     uftdi_stop,
167     uftdi_start,
168     uftdi_fifo_flush,
169     uftdi_fifo_drain,
170     uftdi_out_pipe,
171     uftdi_in_pipe
172 };

174 /* debug support */
175 static uint_t    uftdi_errlevel = USB_LOG_L4;
176 static uint_t    uftdi_errmask = DPRINT_MASK_ALL;
177 static uint_t    uftdi_instance_debug = (uint_t)-1;
178 static uint_t    uftdi_attach_unrecognized = B_FALSE;

180 /*
181  * ds_attach
182  */
183 static int
184 uftdi_attach(ds_attach_info_t *aip)
185 {
186     uftdi_state_t *uf;
187     usb_dev_descr_t *dd;
188     int recognized;

190     uf = kmem_zalloc(sizeof(*uf), KM_SLEEP);
191     uf->uf_dip = aip->ai_dip;
192     uf->uf_usb_events = aip->ai_usb_events;
193     *aip->ai_hdl = (ds_hdl_t)uf;

```

```

195     /* only one port */
196     *aip->ai_port_cnt = 1;

198     if (usb_client_attach(uf->uf_dip, USBDRV_VERSION, 0) != USB_SUCCESS) {
199         uftdi_cleanup(uf, 1);
200         return (USB_FAILURE);
201     }

203     if (usb_get_dev_data(uf->uf_dip,
204                         &uf->uf_dev_data, USB_PARSE_LVL_IF, 0) != USB_SUCCESS) {
205         uftdi_cleanup(uf, 2);
206         return (USB_FAILURE);
207     }

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

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

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

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

219     /*
220      * This device and its clones has numerous physical instantiations.
221      */
222     recognized = B_TRUE;
223     dd = uf->uf_dev_data->dev_descr;
224     switch (dd->idVendor) {
225     case USB_VENDOR_FTDI:
226         switch (dd->idProduct) {
227             case USB_PRODUCT_FTDI_SERIAL_2232C:
228             case USB_PRODUCT_FTDI_SERIAL_8U232AM:
229             case USB_PRODUCT_FTDI_SEMC_DSS20:
230             case USB_PRODUCT_FTDI_CFA_631:
231             case USB_PRODUCT_FTDI_CFA_632:
232             case USB_PRODUCT_FTDI_CFA_633:
233             case USB_PRODUCT_FTDI_CFA_634:
234             case USB_PRODUCT_FTDI_CFA_635:
235             case USB_PRODUCT_FTDI_USBSERIAL:
236             case USB_PRODUCT_FTDI_MX2_3:
237             case USB_PRODUCT_FTDI_MX4_5:
238             case USB_PRODUCT_FTDI_LK202:
239             case USB_PRODUCT_FTDI_LK204:
240             case USB_PRODUCT_FTDI_TACTRIX_OPENPORT_13M:
241             case USB_PRODUCT_FTDI_TACTRIX_OPENPORT_13S:
242             case USB_PRODUCT_FTDI_TACTRIX_OPENPORT_13U:
243             case USB_PRODUCT_FTDI_EISCOU:
244             case USB_PRODUCT_FTDI_UOPTBR:
245             case USB_PRODUCT_FTDI_EMCU2D:
246             case USB_PRODUCT_FTDI_PCMSFU:
247             case USB_PRODUCT_FTDI_EMCU2H:
248                 break;
249             default:
250                 recognized = B_FALSE;
251                 break;
252         }
253     }
254     case USB_VENDOR_SIIG2:
255         switch (dd->idProduct) {
256             case USB_PRODUCT_SIIG2_US2308:
257                 break;
258             default:
259                 recognized = B_FALSE;

```

```

260         break;
261     }
262     break;
263 case USB_VENDOR_INTREPIDCS:
264     switch (dd->idProduct) {
265     case USB_PRODUCT_INTREPIDCS_VALUECAN:
266     case USB_PRODUCT_INTREPIDCS_NEOVI:
267         break;
268     default:
269         recognized = B_FALSE;
270         break;
271     }
272     break;
273 case USB_VENDOR_BBELECTRONICS:
274     switch (dd->idProduct) {
275     case USB_PRODUCT_BBELECTRONICS_USOTL4:
276         break;
277     default:
278         recognized = B_FALSE;
279         break;
280     }
281     break;
282 case USB_VENDOR_MELCO:
283     switch (dd->idProduct) {
284     case USB_PRODUCT_MELCO_PCOPRS1:
285         break;
286     default:
287         recognized = B_FALSE;
288         break;
289     }
290     break;
291 case USB_VENDOR_MARVELL:
292     switch (dd->idProduct) {
293     case USB_PRODUCT_MARVELL_SHEEVAPLUG_JTAG:
294         break;
295     default:
296         recognized = B_FALSE;
297         break;
298     }
299     break;
300 default:
301     recognized = B_FALSE;
302     break;
303 }
304
305 /*
306  * Set 'uftdi_attach_unrecognized' to non-zero to
307  * experiment with newer devices ..
308  */
309 if (!recognized && !uftdi_attach_unrecognized) {
310     uftdi_cleanup(uf, 3);
311     return (USB_FAILURE);
312 }
313
314 USB_DPRINTF_L3(DPRINT_ATTACH, uf->uf_lh,
315 "uftdi: matched vendor 0x%x product 0x%x port %d",
316 dd->idVendor, dd->idProduct, uf->uf_hwport);
317
318 uf->uf_def_ph = uf->uf_dev_data->dev_default_ph;
319
320 mutex_enter(&uf->uf_lock);
321 uf->uf_dev_state = USB_DEV_ONLINE;
322 uf->uf_port_state = UFTDI_PORT_CLOSED;
323 mutex_exit(&uf->uf_lock);
324
325 if (uftdi_create_pm_components(uf) != USB_SUCCESS) {

```

```

326     uftdi_cleanup(uf, 3);
327     return (USB_FAILURE);
328 }
329
330 if (usb_register_event_cbs(uf->uf_dip,
331 uf->uf_usb_events, 0) != USB_SUCCESS) {
332     uftdi_cleanup(uf, 4);
333     return (USB_FAILURE);
334 }
335
336 if (usb_pipe_get_max_bulk_transfer_size(uf->uf_dip,
337 &uf->uf_xfer_sz) != USB_SUCCESS) {
338     uftdi_cleanup(uf, 5);
339     return (USB_FAILURE);
340 }
341
342 /*
343  * TODO: modern ftdi devices have deeper (and asymmetric)
344  * fifos than this minimal 64 bytes .. but how to tell
345  * -safely- ?
346  */
347
348 #define FTDI_MAX_XFERSIZE      64
349
350 if (uf->uf_xfer_sz > FTDI_MAX_XFERSIZE)
351     uf->uf_xfer_sz = FTDI_MAX_XFERSIZE;
352
353 if (uftdi_dev_attach(uf) != USB_SUCCESS) {
354     uftdi_cleanup(uf, 5);
355     return (USB_FAILURE);
356 }
357
358 return (USB_SUCCESS);
359 }
360
361 unchanged_portion_omitted
362
363 /*
364  * pipe operations
365  */
366 static int
367 uftdi_open_pipes(uftdi_state_t *uf)
368 {
369     int ifc, alt;
370     usb_pipe_policy_t policy;
371     usb_ep_data_t *in_data, *out_data;
372     size_t max_xfer_sz;
373
374     /* get max transfer size */
375     if (usb_pipe_get_max_bulk_transfer_size(uf->uf_dip, &max_xfer_sz)
376 != USB_SUCCESS)
377         return (USB_FAILURE);
378
379     /* get ep data */
380     ifc = uf->uf_dev_data->dev_curr_if;
381     alt = 0;
382
383     in_data = usb_lookup_ep_data(uf->uf_dip, uf->uf_dev_data, ifc, alt,
384 0, USB_EP_ATTR_BULK, USB_EP_DIR_IN);
385
386     out_data = usb_lookup_ep_data(uf->uf_dip, uf->uf_dev_data, ifc, alt,
387 0, USB_EP_ATTR_BULK, USB_EP_DIR_OUT);
388
389     if (in_data == NULL || out_data == NULL) {
390         USB_DPRINTF_L2(DPRINT_ATTACH, uf->uf_lh,
391 "uftdi_open_pipes: can't get ep data");
392     }

```

```

1512         return (USB_FAILURE);
1513     }

1515     /*
1516     * Set buffer sizes. Default to UFTDI_XFER_SZ_MAX.
1517     * Use wMaxPacketSize from endpoint descriptor if it is nonzero..
1518     * Cap at a max transfer size of host controller.
1519     */
1520     uf->uf_ibuf_sz = uf->uf_obuf_sz = UFTDI_XFER_SZ_MAX;

1522     if (in_data->ep_descr.wMaxPacketSize)
1523         uf->uf_ibuf_sz = in_data->ep_descr.wMaxPacketSize;
1524     uf->uf_ibuf_sz = min(uf->uf_ibuf_sz, max_xfer_sz);

1526     if (out_data->ep_descr.wMaxPacketSize)
1527         uf->uf_obuf_sz = out_data->ep_descr.wMaxPacketSize;
1528     uf->uf_obuf_sz = min(uf->uf_obuf_sz, max_xfer_sz);

1530     /* open pipes */
1531     policy.pp_max_async_reqs = 2;

1533     if (usb_pipe_open(uf->uf_dip, &in_data->ep_descr, &policy,
1534         USB_FLAGS_SLEEP, &uf->uf_bulkin_ph) != USB_SUCCESS)
1535         return (USB_FAILURE);

1537     if (usb_pipe_open(uf->uf_dip, &out_data->ep_descr, &policy,
1538         USB_FLAGS_SLEEP, &uf->uf_bulkout_ph) != USB_SUCCESS) {
1539         usb_pipe_close(uf->uf_dip, uf->uf_bulkin_ph, USB_FLAGS_SLEEP,
1540             NULL, NULL);
1541         return (USB_FAILURE);
1542     }

1544     mutex_enter(&uf->uf_lock);
1545     uf->uf_bulkin_state = UFTDI_PIPE_IDLE;
1546     uf->uf_bulkout_state = UFTDI_PIPE_IDLE;
1547     mutex_exit(&uf->uf_lock);

1549     return (USB_SUCCESS);
1550 }

```

unchanged portion omitted

```

1802 /*
1803 * start receiving data
1804 */
1805 static int
1806 uftdi_rx_start(uftdi_state_t *uf)
1807 {
1808     usb_bulk_req_t *br;
1809     int rval;

1811     USB_DPRINTF_L4(DPRINT_OUT_PIPE, uf->uf_lh, "uftdi_rx_start");

1813     ASSERT(mutex_owned(&uf->uf_lock));

1815     uf->uf_bulkin_state = UFTDI_PIPE_BUSY;
1816     mutex_exit(&uf->uf_lock);

1818     br = usb_alloc_bulk_req(uf->uf_dip, uf->uf_ibuf_sz, USB_FLAGS_SLEEP);
1819     br->bulk_len = uf->uf_ibuf_sz;
1820     br = usb_alloc_bulk_req(uf->uf_dip, uf->uf_xfer_sz, USB_FLAGS_SLEEP);
1821     br->bulk_len = uf->uf_xfer_sz;
1822     br->bulk_timeout = UFTDI_BULKIN_TIMEOUT;
1823     br->bulk_cb = uftdi_bulkin_cb;
1824     br->bulk_exc_cb = uftdi_bulkin_cb;
1825     br->bulk_client_private = (usb_opaque_t)uf;

```

```

1824     br->bulk_attributes = USB_ATTRS_AUTOCLEARING | USB_ATTRS_SHORT_XFER_OK;

1826     rval = usb_pipe_bulk_xfer(uf->uf_bulkin_ph, br, 0);

1828     if (rval != USB_SUCCESS) {
1829         USB_DPRINTF_L2(DPRINT_IN_PIPE, uf->uf_lh,
1830             "uftdi_rx_start: xfer failed %d", rval);
1831         usb_free_bulk_req(br);
1832     }

1834     mutex_enter(&uf->uf_lock);
1835     if (rval != USB_SUCCESS)
1836         uf->uf_bulkin_state = UFTDI_PIPE_IDLE;

1838     return (rval);
1839 }

1842 /*
1843 * start data transmit
1844 */
1845 static void
1846 uftdi_tx_start(uftdi_state_t *uf, int *xferd)
1847 {
1848     int len; /* bytes we can transmit */
1849     mblk_t *data; /* data to be transmitted */
1850     int data_len; /* bytes in 'data' */
1851     mblk_t *mp; /* current msgblk */
1852     int copylen; /* bytes copy from 'mp' to 'data' */
1853     int rval;

1855     USB_DPRINTF_L4(DPRINT_OUT_PIPE, uf->uf_lh, "uftdi_tx_start");
1856     ASSERT(mutex_owned(&uf->uf_lock));
1857     ASSERT(uf->uf_port_state != UFTDI_PORT_CLOSED);

1859     if (*xferd)
1860         *xferd = 0;
1861     if ((uf->uf_port_flags & UFTDI_PORT_TX_STOPPED) ||
1862         uf->uf_tx_mp == NULL) {
1863         return;
1864     }
1865     if (uf->uf_bulkout_state != UFTDI_PIPE_IDLE) {
1866         USB_DPRINTF_L4(DPRINT_OUT_PIPE, uf->uf_lh,
1867             "uftdi_tx_start: pipe busy");
1868         return;
1869     }
1870     ASSERT(MBLKL(uf->uf_tx_mp) > 0);

1872     /* send as much data as port can receive */
1873     len = min(msgdsize(uf->uf_tx_mp), uf->uf_obuf_sz);
1874     len = min(msgdsize(uf->uf_tx_mp), uf->uf_xfer_sz);

1875     if (len <= 0)
1876         return;
1877     if ((data = allocb(len, BPRI_LO)) == NULL)
1878         return;

1880     /*
1881     * copy no more than 'len' bytes from mblk chain to transmit mblk 'data'
1882     */
1883     data_len = 0;
1884     while (data_len < len && uf->uf_tx_mp) {
1885         mp = uf->uf_tx_mp;
1886         copylen = min(MBLKL(mp), len - data_len);
1887         bcopy(mp->b_rptr, data->b_wptr, copylen);
1888         mp->b_rptr += copylen;

```

```
1889         data->b_wptr += copylen;
1890         data_len += copylen;
1892         if (MBLKL(mp) < 1) {
1893             uf->uf_tx_mp = unlinkb(mp);
1894             freeb(mp);
1895         } else {
1896             ASSERT(data_len == len);
1897         }
1898     }
1900     ASSERT(data_len > 0);
1902     uf->uf_bulkout_state = UFTDI_PIPE_BUSY;
1903     mutex_exit(&uf->uf_lock);
1905     rval = uftdi_send_data(uf, data);
1906     mutex_enter(&uf->uf_lock);
1908     if (rval != USB_SUCCESS) {
1909         uf->uf_bulkout_state = UFTDI_PIPE_IDLE;
1910         uftdi_put_head(&uf->uf_tx_mp, data);
1911     } else {
1912         if (xferd)
1913             *xferd = data_len;
1914     }
1915 }
_____unchanged_portion_omitted_____
```

new/usr/src/uts/common/sys/usb/clients/usbser/usbftdi/uftdi\_var.h 1

```
*****
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
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */
25 /*
26 * Copyright 2012 Hans Rosenfeld <rosenfeld@grumpf.hope-2000.org>
27 */

29 #ifndef _USBSER_USBFTDI_UFTDI_VAR_H
30 #define _USBSER_USBFTDI_UFTDI_VAR_H

32 /*
33  * USB UFTDI definitions
34  */

36 #include <sys/types.h>
37 #include <sys/dditypes.h>
38 #include <sys/note.h>

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

42 #ifdef __cplusplus
43 extern "C" {
44 #endif

46 /*
47  * PM support
48  */
49 typedef struct uftdi_pm {
50     uint8_t    pm_wakeup_enabled;    /* remote wakeup enabled */
51     uint8_t    pm_pwr_states;    /* bit mask of power states */
52     boolean_t  pm_raise_power;    /* driver is about to raise power */
53     uint8_t    pm_cur_power;    /* current power level */
54     uint_t     pm_busy_cnt;    /* number of set_busy requests */
55 } uftdi_pm_t;
unchanged portion omitted

64 _NOTE(SCHEME_PROTECTS_DATA("uftdi_regs", uftdi_regs))

66 /*
67  * per device state structure
```

new/usr/src/uts/common/sys/usb/clients/usbser/usbftdi/uftdi\_var.h 2

```
68 */
69 typedef struct uftdi_state {
70     kmutex_t    uf_lock;    /* structure lock */
71     dev_info_t  *uf_dip;    /* device info */
72     int         uf_dev_flags;    /* device flags */
73     int         uf_hwport;    /* hw port number */
74     int         uf_port_state;    /* port state */
75     int         uf_port_flags;    /* port flags */
76     ds_cb_t     uf_cb;    /* DSD callbacks */

78     /*
79      * USB
80      */
81     usb_client_dev_data_t *uf_dev_data;    /* registration data */
82     usb_event_t  *uf_usb_events;    /* usb events */
83     usb_pipe_handle_t uf_def_ph;    /* default pipe hdl */
84     usb_pipe_handle_t uf_bulkin_ph;    /* in pipe hdl */
85     int          uf_bulkin_state;    /* in pipe state */
86     usb_pipe_handle_t uf_bulkout_ph;    /* in pipe hdl */
87     int          uf_bulkout_state;    /* out pipe state */
88     usb_log_handle_t uf_lh;    /* USB log handle */
89     int          uf_dev_state;    /* USB device state */
90     size_t       uf_ibuf_sz;    /* input buffer size */
91     size_t       uf_obuf_sz;    /* output buffer size */
92     size_t       uf_xfer_sz;    /* HCI bulk xfer size */

93     uftdi_pm_t   *uf_pm;    /* PM support */

95     /*
96      * data receive and transmit
97      */
98     mblk_t        *uf_rx_mp;    /* rx data */
99     mblk_t        *uf_tx_mp;    /* tx data */
100    kcondvar_t     uf_tx_cv;    /* tx completion */

102     /*
103      * soft registers
104      */
105     uftdi_regs_t  uf_softr;    /* config registers */
106     uint16_t      uf_mctl;    /* modem control */
107     uint8_t       uf_msr;    /* modem status */
108     uint8_t       uf_lsr;    /* line status register */

110 } uftdi_state_t;

112 _NOTE(MUTEX_PROTECTS_DATA(uftdi_state::uf_lock, uftdi_state))
113 _NOTE(DATA_READABLE_WITHOUT_LOCK(uftdi_state::{
114     uf_dip
115     uf_dev_data
116     uf_usb_events
117     uf_def_ph
118     uf_lh
119     uf_ibuf_sz
120     uf_obuf_sz
121     uf_xfer_sz
122     uf_pm
123     uf_port_state
124     uf_cb
125     uf_bulkin_ph
126     uf_bulkout_ph
127     uf_hwport
128 })))
unchanged portion omitted

175 #endif

177 #endif /* _USBSER_USBFTDI_UFTDI_VAR_H */
```