```
**********************************************************
   17146 Wed Sep  3 13:02:44 2014
new/usr/src/uts/common/sys/ecppvar.h
5084 struct ecppunit's e_busy is multi-value, can't be a boolean_t
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (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 2004 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_ECPPVAR_H
#define _SYS_ECPPVAR_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/note.h>

#ifdef  __cplusplus
extern "C" {
#endif

struct ecppunit;

/*
 * Hardware-abstraction structure
 */
struct ecpp_hw {
        int     (*map_regs)(struct ecppunit *);         /* map registers */
        void    (*unmap_regs)(struct ecppunit *);       /* unmap registers */
        int     (*config_chip)(struct ecppunit *);      /* configure SuperIO */
        void    (*config_mode)(struct ecppunit *);      /* config new mode */
        void    (*mask_intr)(struct ecppunit *);        /* mask interrupts */
        void    (*unmask_intr)(struct ecppunit *);      /* unmask interrupts */
        int     (*dma_start)(struct ecppunit *);        /* start DMA transfer */
        int     (*dma_stop)(struct ecppunit *, size_t *); /* stop DMA xfer */
        size_t  (*dma_getcnt)(struct ecppunit *);       /* get DMA counter */
        ddi_dma_attr_t  *attr;                          /* DMA attributes */
};
```

*unchanged_portion_omitted_*

```
/* ecpp e_busy states */
typedef enum {
        ECPP_IDLE = 1,  /* No ongoing transfers */
        ECPP_BUSY = 2,  /* Ongoing transfers on the cable */
        ECPP_DATA = 3,  /* Not used */
        ECPP_ERR = 4,   /* Bad status in Centronics mode */
        ECPP_FLUSH = 5  /* Currently flushing the q */
} ecpp_busy_t;

#endif /* ! codereview */
/*
 * ecpp soft state structure
 */
struct ecppunit {
        kmutex_t        umutex;         /* lock for this structure */
        int             instance;       /* instance number */
        dev_info_t      *dip;           /* device information */
        ddi_iblock_cookie_t ecpp_trap_cookie;   /* interrupt cookie */
        ecpp_busy_t     e_busy;         /* ecpp busy flag */
        boolean_t       e_busy;         /* ecpp busy flag */
        kcondvar_t      pport_cv;       /* cv to signal idle state */
        /*
         * common SuperIO registers
         */
        struct info_reg         *i_reg;         /* info registers */
        struct fifo_reg         *f_reg;         /* fifo register */
        ddi_acc_handle_t        i_handle;
        ddi_acc_handle_t        f_handle;
        /*
         * DMA support
         */
        ddi_dma_handle_t        dma_handle;     /* DMA handle */
        ddi_dma_cookie_t        dma_cookie;     /* current cookie */
        uint_t                  dma_cookie_count;       /* # of cookies */
        uint_t                  dma_nwin;       /* # of DMA windows */
        uint_t                  dma_curwin;     /* current window number */
        uint_t                  dma_dir;        /* transfer direction */
        /*
         * hardware-dependent stuff
         */
        struct ecpp_hw  *hw;            /* operations/attributes */
        union {                         /* hw-dependent data */
                struct ecpp_ebus        ebus;
                struct ecpp_m1553       m1553;
#if defined(__x86)
                struct ecpp_x86         x86;
#endif
        } uh;
        /*
         * DDI/STREAMS stuff
         */
        boolean_t       oflag;          /* instance open flag */
        queue_t         *readq;         /* pointer to readq */
        queue_t         *writeq;        /* pointer to writeq */
        mblk_t          *msg;           /* current message block */
        boolean_t       suspended;      /* driver suspended status */
        /*
         * Modes of operation
         */
        int             current_mode;   /* 1284 mode */
        uchar_t         current_phase;  /* 1284 phase */
        uchar_t         backchannel;    /* backchannel mode supported */
        uchar_t         io_mode;        /* transfer mode: PIO/DMA */
        /*
         * Ioctls support
         */
        struct ecpp_transfer_parms xfer_parms;  /* transfer parameters */
        struct ecpp_regs regs;          /* control/status registers */
        uint8_t         saved_dsr;      /* store the dsr returned from TESTIO */
        boolean_t       timeout_error;  /* store the timeout for GETERR */
        uchar_t         port;           /* xfer type: dma/pio/tfifo */
```

```
167         struct prn_timeouts prn_timeouts; /* prnio timeouts */
168         /*
169          * ecpp.conf parameters
170          */
171         uchar_t         init_seq;       /* centronics init seq */
172         uint32_t        wsrv_retry;     /* delay (ms) before next wsrv */
173         uint32_t        wait_for_busy;  /* wait for BUSY to deassert */
174         uint32_t        data_setup_time; /* pio centronics handshake */
175         uint32_t        strobe_pulse_width; /* pio centronics handshake */
176         uint8_t         fast_centronics; /* DMA/PIO centronics */
177         uint8_t         fast_compat;    /* DMA/PIO 1284 compatible mode */
178         uint32_t        ecp_rev_speed;  /* rev xfer speed in ECP, bytes/sec */
179         uint32_t        rev_watchdog;   /* rev xfer watchdog period, ms */
180         /*
181          * Timeouts
182          */
183         timeout_id_t    timeout_id;     /* io transfers timer */
184         timeout_id_t    fifo_timer_id;  /* drain SuperIO FIFO */
185         timeout_id_t    wsrv_timer_id;  /* wsrv timeout */
186         /*
187          * Softintr data
188          */
189         ddi_softintr_t  softintr_id;
190         int             softintr_flags; /* flags indicating softintr task */
191         uint8_t         softintr_pending;
192         /*
193          * Misc stuff
194          */
195         caddr_t         ioblock;        /* transfer buffer block */
196         size_t          xfercnt;        /* # of bytes to transfer */
197         size_t          resid;          /* # of bytes not transferred */
198         caddr_t         next_byte;      /* next byte for PIO transfer */
199         caddr_t         last_byte;      /* last byte for PIO transfer */
200         uint32_t        ecpp_drain_counter;     /* allows fifo to drain */
201         uchar_t         dma_cancelled;  /* flushed while dma'ing */
202         uint8_t         tfifo_intr;     /* TFIFO switch interrupt workaround */
203         size_t          nread;          /* requested read */
204         size_t          last_dmacnt;    /* DMA counter value for rev watchdog */
205         uint32_t        rev_timeout_cnt; /* number of watchdog invocations */
206         /*
207          * Spurious interrupt detection
208          */
209         hrtime_t        lastspur;       /* last time spurious intrs started */
210         long            nspur;          /* spurious intrs counter */
211         /*
212          * Statistics
213          */
214         kstat_t         *ksp;           /* kstat pointer */
215         kstat_t         *intrstats;     /* kstat interrupt counter */
216         /*
217          * number of bytes, transferred in and out in each mode
218          */
219         uint32_t        ctxpio_obytes;
220         uint32_t        obytes[ECPP_EPP_MODE+1];
221         uint32_t        ibytes[ECPP_EPP_MODE+1];
222         /*
223          * other stats
224          */
225         uint32_t        to_mode[ECPP_EPP_MODE+1]; /* # transitions to mode */
226         uint32_t        xfer_tout;      /* # transfer timeouts */
227         uint32_t        ctx_cf;         /* # periph check failures */
228         uint32_t        joblen;         /* of bytes xfer'd since open */
229         uint32_t        isr_reattempt_high;     /* max times isr has looped */
230         /*
231          * interrupt stats
232          */
```



```
233         uint_t          intr_hard;
234         uint_t          intr_spurious;
235         uint_t          intr_soft;
236         /*
237          * identify second register set for ecp mode on Sx86
238          */
239         int             noecpregs;
240 };

242 _NOTE(MUTEX_PROTECTS_DATA(ecppunit::umutex, ecppunit))
243 _NOTE(DATA_READABLE_WITHOUT_LOCK(ecppunit::dip))
244 _NOTE(DATA_READABLE_WITHOUT_LOCK(ecppunit::instance))
245 _NOTE(DATA_READABLE_WITHOUT_LOCK(ecppunit::i_reg))
246 _NOTE(DATA_READABLE_WITHOUT_LOCK(ecppunit::f_reg))
247 _NOTE(DATA_READABLE_WITHOUT_LOCK(ecppunit::i_handle))
248 _NOTE(DATA_READABLE_WITHOUT_LOCK(ecppunit::f_handle))
249 _NOTE(DATA_READABLE_WITHOUT_LOCK(ecppunit::ecpp_trap_cookie))
250 _NOTE(DATA_READABLE_WITHOUT_LOCK(ecppunit::readq))
251 _NOTE(DATA_READABLE_WITHOUT_LOCK(ecppunit::writeq))

253 /*
254  * current_phase values
255  */
256 #define ECPP_PHASE_INIT         0x00    /* initialization */
257 #define ECPP_PHASE_NEGO         0x01    /* negotiation */
258 #define ECPP_PHASE_TERM         0x02    /* termination */
259 #define ECPP_PHASE_PO           0x03    /* power-on */

261 #define ECPP_PHASE_C_FWD_DMA    0x10    /* cntrx/compat fwd dma xfer */
262 #define ECPP_PHASE_C_FWD_PIO    0x11    /* cntrx/compat fwd PIO xfer */
263 #define ECPP_PHASE_C_IDLE       0x12    /* cntrx/compat idle */

265 #define ECPP_PHASE_NIBT_REVDATA 0x20    /* nibble/byte reverse data */
266 #define ECPP_PHASE_NIBT_AVAIL   0x21    /* nibble/byte reverse data available */
267 #define ECPP_PHASE_NIBT_NAVAIL  0x22    /* nibble/byte reverse data not avail */
268 #define ECPP_PHASE_NIBT_REVIDLE 0x22    /* nibble/byte reverse idle */
269 #define ECPP_PHASE_NIBT_REVINTR 0x23    /* nibble/byte reverse interrupt */

271 #define ECPP_PHASE_ECP_SETUP    0x30    /* ecp setup */
272 #define ECPP_PHASE_ECP_FWD_XFER 0x31    /* ecp forward transfer */
273 #define ECPP_PHASE_ECP_FWD_IDLE 0x32    /* ecp forward idle */
274 #define ECPP_PHASE_ECP_FWD_REV  0x33    /* ecp forward to reverse */
275 #define ECPP_PHASE_ECP_REV_XFER 0x34    /* ecp reverse transfer */
276 #define ECPP_PHASE_ECP_REV_IDLE 0x35    /* ecp reverse idle */
277 #define ECPP_PHASE_ECP_REV_FWD  0x36    /* ecp reverse to forward */

279 #define ECPP_PHASE_EPP_INIT_IDLE 0x40   /* epp init phase */
280 #define ECPP_PHASE_EPP_IDLE     0x41    /* epp all-round phase */

282 #define FAILURE_PHASE           0x80
283 #define UNDEFINED_PHASE         0x81

285 /* ecpp return values */
286 #define SUCCESS         1
287 #define FAILURE         2

273 /* ecpp e_busy states */
274 #define ECPP_IDLE       1 /* No ongoing transfers */
275 #define ECPP_BUSY       2 /* Ongoing transfers on the cable */
276 #define ECPP_DATA       3 /* Not used */
277 #define ECPP_ERR        4 /* Bad status in Centronics mode */
278 #define ECPP_FLUSH      5 /* Currently flushing the q */

289 #define TRUE            1
290 #define FALSE           0
```

```
292 /* message type */
293 #define ECPP_BACKCHANNEL        0x45

295 /* transfer modes */
296 #define ECPP_DMA                0x1
297 #define ECPP_PIO                0x2

299 /* tuneable timing defaults */
300 #define CENTRONICS_RETRY        750     /* 750 milliseconds */
301 #define WAIT_FOR_BUSY           1000    /* 1000 microseconds */
302 #define SUSPEND_TOUT            10      /* # seconds before suspend fails */

304 /* Centronics hanshaking defaults */
305 #define DATA_SETUP_TIME         2       /* 2 uSec Data Setup Time (2x min) */
306 #define STROBE_PULSE_WIDTH      2       /* 2 uSec Strobe Pulse (2x min) */

308 /* 1284 Extensibility Request values */
309 #define ECPP_XREQ_NIBBLE        0x00    /* Nibble Mode Rev Channel Transfer */
310 #define ECPP_XREQ_BYTE          0x01    /* Byte Mode Rev Channel Transfer */
311 #define ECPP_XREQ_ID            0x04    /* Request Device ID */
312 #define ECPP_XREQ_ECP           0x10    /* Request ECP Mode */
313 #define ECPP_XREQ_ECPRLE        0x30    /* Request ECP Mode with RLE */
314 #define ECPP_XREQ_EPP           0x40    /* Request EPP Mode */
315 #define ECPP_XREQ_XLINK         0x80    /* Request Extensibility Link */

317 /* softintr flags */
318 #define ECPP_SOFTINTR_PIONEXT   0x1     /* write next byte in PIO mode */

320 /* Stream  defaults */
321 #define IO_BLOCK_SZ     1024 * 128      /* transfer buffer size */
322 #define ECPPHIWAT       32 * 1024  * 6
323 #define ECPPLOWAT       32 * 1024  * 4

325 /* Loop timers */
326 #define ECPP_REG_WRITE_MAX_LOOP 100     /* cpu is faster than superio */
327 #define ECPP_ISR_MAX_DELAY      30      /* DMAC slow PENDING status */

329 /* misc constants */
330 #define ECPP_FIFO_SZ            16      /* FIFO size */
331 #define FIFO_DRAIN_PERIOD       250000  /* max FIFO drain period in usec */
332 #define NIBBLE_REV_BLKSZ        1024    /* send up to # bytes at a time */
333 #define FWD_TIMEOUT_DEFAULT     90      /* forward xfer timeout in seconds */
334 #define REV_TIMEOUT_DEFAULT     0       /* reverse xfer timeout in seconds */

336 /* ECP mode constants */
337 #define ECP_REV_BLKSZ           1024    /* send up to # bytes at a time */
338 #define ECP_REV_BLKSZ_MAX       (4 * 1024)      /* maximum of # bytes */
339 #define ECP_REV_SPEED           (1 * 1024 * 1024)       /* bytes/sec */
340 #define ECP_REV_MINTOUT         5       /* min ECP rev xfer timeout in ms */
341 #define REV_WATCHDOG            100     /* poll DMA counter every # ms */

343 /* spurious interrupt detection */
344 #define SPUR_CRITICAL           100     /* number of interrupts... */
345 #define SPUR_PERIOD             1000000000 /* in # ns */

347 /*
348  * Copyin/copyout states
349  */
350 #define ECPP_STRUCTIN           0
351 #define ECPP_STRUCTOUT          1
352 #define ECPP_ADDRIN             2
353 #define ECPP_ADDROUT            3

355 /*
356  * As other ioctls require the same structure, put inner struct's into union
357  */
```



```
358 struct ecpp_copystate {
359         int     state;          /* see above */
360         void    *uaddr;         /* user address of the following structure */
361         union {
362                 struct ecpp_device_id           devid;
363                 struct prn_1284_device_id       prn_devid;
364                 struct prn_interface_info       prn_if;
365         } un;
366 };
```

_____***unchanged_portion_omitted_***