```
/* (-lgl
 *      The information contained herein is a trade secret of Mark Williams
 *      Company, and  is confidential information.  It is provided  under a
 *      license agreement,  and may be  copied or disclosed  only under the
 *      terms of  that agreement. Any  reproduction or disclosure  of this
 *      material without the express written authorization of Mark Williams
 *      Company or persuant to the license agreement is unlawful.
 *
 *      COHERENT Version 0.7.3
 *      Copyright (c) 1982, 1983, 1984.
 *      An unpublished work by Mark Williams Company, Chicago.
 *      All rights reserved.
 -lgl) */
/*
 *      Async line driver for Zilog Z8030 Serial Comm Controller (SCC)
 *      using interrupts and modem control. Multiple line support is included
 *
 *      Version 1.2, April 1985
 */

#include <coherent.h>
#include <con.h>
#include <errno.h>
#include <stat.h>
#include <tty.h>
#include <uproc.h>
#include <sched.h>
#include <signal.h>
#include <romconf.h>

#define NMINOR   6                  /* Number of serial ports */
#define NSCC     3                  /* Number of actual SCC devices */
#define MODEM_CTL 0x80              /* Mask in minor device for modem control */
#define VECTOR   0x10               /* Base vector number */
#define endof(x)            (&x[sizeof(x)/sizeof(x[0])])
#define alindex(x)          (((x)-VECTOR)>>3)       /* index in tty struct */

#define BAUD(x,y) (int)((x)/((y)*32L))-2L  /* calc counter value */

extern struct romconf romconf;

/*
 * Functions.
 */
int     alRxintr();                 /* Receive interrupt handler */
int     alTxintr();                 /* Transmit interrupt handler */
int     alESintr();                 /* External status interrupt handler */
int     alSCintr();                 /* Special condition interrupt handler */
int     alstart();
int     alparam();
int     alopen();
int     alclose();
int     alread();
int     alwrite();
int     alioctl();
int     alload();
int     aluload();
int     nulldev();
int     nonedev();

/*
 * Configuration table.
 */
CON alcon ={
        DFCHR,                          /* Flags */
        5,                              /* Major index */
        alopen,                         /* Open */
        alclose,                        /* Close */
        nulldev,                        /* Block */
        alread,                         /* Read */
        alwrite,                        /* Write */
        alioctl,                        /* Ioctl */
        nulldev,                        /* Powerfail */
        nulldev,                        /* Timeout */
        alload,                         /* Load */
        aluload                         /* Unload */
};

#define albase(tp)      ((int)((tp)->t_ddp))

/*
 *      Z8030 register offsets. Note that these are relative to the
 *      base address of the port extracted from the altty[] structure.
 */

#define WR0     0x01                    /* Command Register */
#define WR1     0x03                    /* Tx & Rx Interrupts */
#define WR2     0x05                    /* Interrupt Vector */
#define WR3     0x07                    /* Rx Params & Control */
#define WR4     0x09                    /* Tx & Rx Misc Params & Modes */
#define WR5     0x0b                    /* Tx Params & Control */
#define WR6     0x0d                    /* 1st Sync Byte & SDLC garf */
#define WR7     0x0f                    /* 2nd Sync Byte & SDLC garf */
#define WR8     0x11                    /* Tx buffer */
#define WR9     0x13                    /* Master Interrupt Control */
#define WR10    0x15                    /* Misc TX & Rx Control */
#define WR11    0x17                    /* Clock Mode Control */
#define WR12    0x19                    /* Baud Rate Gen Low Byte */
#define WR13    0x1b                    /* Baud Rate Gen High Byte */
#define WR14    0x1d                    /* Misc Control */
#define WR15    0x1f                    /* External/Status Int Control */

#define RR0     0x01                    /* Tx & Rx Buffer & Misc Status */
#define RR1     0x03                    /* Special Recv Condition Status */
#define RR2     0x05                    /* Interrupt Vector */
#define RR3     0x07                    /* Interrupt Pending Reg */
#define RR8     0x11                    /* Rx buffer */
#define RR10    0x15                    /* Misc Status */
```

```
#define RR12    0x19                            /* Baud Rate Gen Low Byte */
#define RR13    0x1b                            /* Baud Rate Gen High Byte */
#define RR15    0x1f                            /* External/Status Int Control */

/*
 * Bits in registers.
 */

#define ALMODE  0x4c                            /* x16 CLK, 2SB, no parity */
#define RxPARAM 0xc0                            /* 8 bits/char, no enable */
#define TxPARAM 0x60                            /* 8 bits/char, no enable */
#define NRZ     0x00                            /* NRZ mode */
#define BRGINIT 0x56                            /* Tx & Rx CLK = BRG output */
#define DEFBAUD 16    /* Default baud rate (index in albaud[]) */
#define BRGEN   0x03                            /* BRG in=PCLK, enable BRG */
#define RxEN    (RxPARAM|0x01)                  /* 8 bits/char, enable */
#define TxEN    (TxPARAM|0x08)                  /* 8 bits/char, enable */
#define RxENABLE 0x01                           /* Rx enable bit in WR3 */
#define TxENABLE 0x08                           /* Tx enable bit in WR5 */
#define RESEXTINT 0x10                          /* reset exteral/status ints */
#define PEVEN   0x03                            /* even parity + enable */
#define PODD    0x01                            /* odd parity + enable */
#define PNONE   0x00                            /* no parity */
#define Rx8BPC  0xc0                            /* Rx 8 bits/char */
#define Rx7BPC  0x40                            /* Rx 7 bits/char */
#define Tx8BPC  0x60                            /* Tx 8 bits/char */
#define Tx7BPC  0x20                            /* Tx 7 bits/char */
#define RESET   0xc0                            /* hard reset chip */
#define VIS     0x01                            /* vector indicates status */
#define MIE     0x08                            /* master interrupt enable */
#define RESTxI  0x28                            /* reset pending Tx interrupt */
#define ERRESET 0x30                            /* special cond. error reset */
#define RESIUS  0x38                            /* reset highest IUS */
#define BREAK   0x80                            /* BREAK bit position */
#define CTS     0x20                            /* CTS bit position */
#define DCD     0x08                            /* DCD bit position */
#define DTR     0x80                            /* DTR bit position */
#define RTS     0x02                            /* RTS bit position */
#define BREAKIE 0x80                            /* break interrupt enable */
#define CTSIE   0x20                            /* CTS interrupt enable */
#define DCDIE   0x08                            /* DCD interrupt enable */
#define RxASCIE 0x10                            /* Rx on all or spec int enable */
#define TxIE    0x02                            /* Tx interrupt enable */
#define EXTIE   0x01                            /* external interrupt enable */
#define RxAVAIL 0x01                            /* char available */
#define TxEMPTY 0x04                            /* transmit buffer empty */


/*
 * Terminal structure.
 * NOTE: These entries must be in the SAME order as the ports are within
 * the SCC chip in order for the alindex() macro to work correctly !!
 */
TTY     altty[NMINOR] = {
        { {0}, {0}, 0x100, alstart, alparam, B9600, B9600 },
        { {0}, {0}, 0x120, alstart, alparam, B9600, B9600 },
        { {0}, {0}, 0x300, alstart, alparam, B9600, B9600 },
        { {0}, {0}, 0x320, alstart, alparam, B9600, B9600 },
        { {0}, {0}, 0x380, alstart, alparam, B9600, B9600 },
        { {0}, {0}, 0x3A0, alstart, alparam, B9600, B9600 }
};

/*
 * Baud rate table.
 * Indexed by ioctl bit rates.
 * These start out as bit rates and are converted
 * to something sendable to the chip (for the counter)
 * by the load routine.
 */
int albaud[] ={
        0,                                      /* 0 */
        50,                                     /* 50 */
        75,                                     /* 75 */
        110,                                    /* 110 */
        134,                                    /* 134 */
        150,                                    /* 150 */
        200,                                    /* 200 */
        300,                                    /* 300 */
        600,                                    /* 600 */
        1200,                                   /* 1200 */
        1800,                                   /* 1800 */
        2000,                                   /* 2000 */
        2400,                                   /* 2400 */
        3600,                                   /* 3600 */
        4800,                                   /* 4800 */
        7200,                                   /* 7200 */
        9600,                                   /* 9600 */
        19200,                                  /* 19200 */
        0,                                      /* EXTA */
        0                                       /* EXTB */
};

struct  al_init {
        char    port;                           /* port offset */
        char    val;                            /* initial value */
};

struct  al_init alinit[] = {
        {       WR4,    ALMODE   },,            /* initial mode */
        {       WR3,    RxPARAM },,             /* receive params */
        {       WR5,    TxPARAM },,             /* transmit params */
        {       WR10,   NRZ      },,            /* nrz encoding */
        {       WR11,   BRGINIT },,             /* baud rate source inits */
        {       WR12,   0        },,            /* low byte initial value */
        {       WR13,   0        },,            /* high byte initial value */
        {       WR14,   BRGEN    },,            /* use PCLK in and enable */
        {       WR3,    RxEN     },,            /* params + enable */
        {       WR5,    TxEN     },,            /* params + enable */
        {       WR15,   0         },      /* interrupt enables */
        {       WR0,    RESEXTINT },      /* reset interrupts */
        {       WR0,    RESEXTINT },      /* reset interrupts */
};

int     al_ercnt;                         /* chip error count */

/*
 * Upon loading driver, reset all SCC's using hardware reset, program
 * the interrupt vector for each chip, program each channel using the
 * port/value pairs found in the alinit[] struct (repeated for all
 * channels of all SCC's), and then go away until the first open.
 */
alload()
{
        register TTY *tp;
        register struct al_init *p;
        register int b;
        register unsigned int vec;

        /*
         * Convert albaud[] depending the the clock rate
         */
        for (b = 0; b < sizeof(albaud)/sizeof(albaud[0]); b++)
                if (albaud[b])
                        albaud[b] = BAUD(romconf.rom_ctype ?
                        6000000L : 4000000L, albaud[b]);
        alinit[5].val = albaud[DEFBAUD] & 0xff; /* Default baudrate low */
        alinit[6].val = albaud[DEFBAUD] >> 8;  /* Default baudrate high */
        al_ercnt = 0;                     /* reset count */
        for (tp = &altty[0], vec = VECTOR; tp < &altty[NMINOR]; tp += 2) {
                b = albase(tp);
                outb(b+WR9, RESET|VIS); /* reset chip */
                outb(b+WR2, vec);       /* set chip's base vector */
                setivec(vec+0, alTxintr);       /* channel B interrupt vecs *
                setivec(vec+2, alESintr);
                setivec(vec+4, alRxintr);
                setivec(vec+6, alSCintr);
                setivec(vec+8, alTxintr);       /* channel A interrupt vecs *
                setivec(vec+10, alESintr);
                setivec(vec+12, alRxintr);
                setivec(vec+14, alSCintr);
                vec += 16;
        }
        for (tp = &altty[0]; tp < &altty[NMINOR]; tp++) {
                b = albase(tp);
                for (p = &alinit[0]; p < endof(alinit); p++)
                        outb(b+(p->port), p->val);
        }
}

aluload()
{
        register int i;
        /* outb(BASE+WR9, RESET|VIS);*/          /* reset chip */
        for (i = VECTOR; i < (VECTOR+NSCC*16); i+=2)
                clrivec(i);
}

alopen(dev, mode)
dev_t dev;
int mode;
{
        register TTY *tp;
        register int s;
        register int b;
        register int m;

        m = minor(dev)&~MODEM_CTL;
        if (m >= NMINOR) {
                u.u_error = ENXIO;
                return;
        }
        tp = &altty[m];
        if ((tp->t_flags&T_EXCL)!=0 && super()==0) {
                u.u_error = ENODEV;
                return;
        }
        if ((minor(dev) & MODEM_CTL) == 1)
                tp->t_flags |= T_MODC;
        if (tp->t_open == 0) {
                s = sphi();
                b = albase(tp);
                outb(b+WR3, RxEN);
                outb(b+WR5, (TxEN|RTS|DTR));
                outb(b+WR9, (MIE|VIS));
                if (tp->t_flags & T_MODC) {
                        outb(b+WR15, (BREAKIE|CTSIE|DCDIE));
                        outb(b+WR1, EXTIE);
                        while ((inb(b+RR0) & (DCD|CTS)) == 0)
                                sleep((char *)(&tp->t_open), CVTTIN, IVTTIN,
                                        SVTTIN);
                } else {
                        outb(b+WR15, BREAKIE);
                }
                ttopen(tp);
                tp->t_flags |= T_CARR;
                outb(b+WR1, (EXTIE|TxIE|RxASCIE));
                spl(s);
        }
        tp->t_open++;
        ttsetgrp(tp, dev);
}

alclose(dev, mode)
dev_t dev;
int mode;
```

```
{
        register TTY *tp;
        register int s;
        register int b;

        s = sphi();
        tp = &altty[minor(dev)&~MODEM_CTL];
        if (--tp->t_open == 0) {
                ttclose(tp);
                b = albase(tp);
                outb(b+WR1, 0);         /* disable Rx, Tx, ES, and SC ints */
                outb(b+WR5, tp->t_flags&T_HPCL ? TxEN : (TxEN|RTS|DTR));
        }
        spl(s);
}

/*
 * For these next two routines, the MFSYS
 * is equivalent to passing a priority of
 * 'sphi()' (i.e. interrupts disabled).
 */
alread(dev, iop)
dev_t dev;
IO *iop;
{
        ttread(&altty[minor(dev)&~MODEM_CTL], iop, SFCW);
}

alwrite(dev, iop)
dev_t dev;
IO *iop;
{
        ttwrite(&altty[minor(dev)&~MODEM_CTL], iop, SFCW);
}

alioctl(dev, com, vec)
dev_t dev;
struct sgttyb *vec;
{
        register int s;

        s = sphi();
        ttioctl(&altty[minor(dev)&~MODEM_CTL], com, vec);
        spl(s);
}

alstart(tp)
TTY *tp;
{
        register int c;
        register int b;

        b = albase(tp);
        while ((inb(b+RR0)&TxEMPTY)!=0 && (c=ttout(tp))>=0)
```

```
                        outb(b+WR8, c);
}

alparam(tp)
TTY *tp;
{
        register int b;
        register int baud;

        b = albase(tp);
        if (tp->t_sgttyb.sg_ispeed!=tp->t_sgttyb.sg_ospeed ||
            (baud=albaud[tp->t_sgttyb.sg_ispeed]) == 0) {
                u.u_error = ENODEV;
                return;
        }
        outb(b+WR12, (baud&0xff));
        outb(b+WR13, baud>>8);
        switch (tp->t_sgttyb.sg_flags & (EVENP|ODDP|RAW)) {
        case EVENP:
                outb(b+WR3, (Rx7BPC|RxENABLE));
                outb(b+WR5, (Tx7BPC|DTR|RTS|TxENABLE));
                outb(b+WR4, (ALMODE|PEVEN));
                break;

        case ODDP:
                outb(b+WR3, (Rx7BPC|RxENABLE));
                outb(b+WR5, (Tx7BPC|DTR|RTS|TxENABLE));
                outb(b+WR4, (ALMODE|PODD));
                break;

        default:
                outb(b+WR3, (Rx8BPC|RxENABLE));
                outb(b+WR5, (Tx8BPC|DTR|RTS|TxENABLE));
                outb(b+WR4, (ALMODE|PNONE));
        }
}

/*
 * Special condition interrupt handler for SCC. Currently, since parity
 * errors are not set up as special conditions, only framing and overrun
 * errors will cause this interrupt. For now, simply log the error,
 * clear it, and return.
 */

alSCintr(id)
{
        register TTY *tp;
        register int b;
        register int s;

        tp = &altty[alindex(id)];
        b = albase(tp);
        s = sphi();
        if ((inb(b+RR1)&0x70) != 0) {
```

```
                        outb(b+WR0, ERRESET);
                        inb(b+RR8);             /* read & discard bad data */
                        al_ercnt++;
                }
                spl(s);
                outb(b+WR0, RESIUS);    /* reset interrupt */
}

/*
 * External status interrupt handler for the SCC. The handler is called
 * upon changes in the DCD or CTS modem control lines, or upon a break
 * (line spacing) condition on the Rx line. Note that this only applies
 * if the given line is opened (or attempting to be) with modem control
 * enabled.
 */

alESintr(id)
{
        register TTY *tp;
        register int b;
        register int what;
        register int s;

        tp = &altty[alindex(id)];
        b = albase(tp);
        s = sphi();
        what = inb(b+RR0);
        outb(b+WR0, RESEXTINT);
/*      if ((what & BREAK) != 0) {
                inb(b+RR8);             read + dismiss
                ttsignal(tp, SIGINT);
                ttflush(tp);
        } else {
                if ((tp->t_flags & T_MODC) == 1)
                        if ((what & (DCD|CTS)) != 0)
                                if (tp->t_open == 0)
                                        wakeup((char *)(&tp->t_open));
        }               */

        if ((tp->t_flags & T_MODC) == 1)
                if ((what & (DCD|CTS)) != 0)
                        if (tp->t_open == 0)
                                wakeup((char *)(&tp->t_open));

        spl(s);
        outb(b+WR0, RESIUS);            /* reset interrupt */
}

/*
 * Receive interrupt handler for the SCC.
 */

alRxintr(id)
{
	register TTY *tp;
	register int b;
	register int s;

	tp = &altty[alindex(id)];
	b = albase(tp);
	s = sphi();
	if ((inb(b+RR0) & RxAVAIL) != 0) {
		do {
			ttin(tp, inb(b+RR8));
		} while ((inb(b+RR0) & RxAVAIL) != 0);
	}
	spl(s);
	outb(b+WR0, RESIUS);
}

/*
 * Transmit interrupt handler for the SCC.
 */

alTxintr(id)
{
	register TTY *tp;
	register int b;
	register int s;

	tp = &altty[alindex(id)];
	b = albase(tp);
	s = sphi();
	if ((inb(b+RR0) & TxEMPTY) != 0) {
		outb(b+WR0, RESTxI);	/* reset Tx ints just in case */
		ttstart(tp);
	}
	spl(s);
	outb(b+WR0, RESIUS);
}
```