TMS320 DSP
DESIGNER'S NOTEBOOK

Using a TMS320C30 Serial Port as an Asynchronous RS-232 Port

APPLICATION BRIEF: SPRA240

Corey Minyard
Bell Northern Research

Texas Instruments
May 1994

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

Contents

Figures

Examples

Using a TMS320C30 Serial Port as an Asynchronous RS-232 Port

Abstract

Although the TMS320C30 serial ports were designed to be used as synchronous ports, they can be used as asynchronous ports under software control. This application note describes the hardware and software to use a TMS320C30 serial port as an asynchronous port. A schematic diagram and a lengthy code listing are provided to illustrate the solution.

Design Problem

Although the TMS320C30 serial ports were designed to be used as synchronous ports, they can be used as asynchronous ports with a little creative software. This application note describes the hardware and software to use a TMS320C30 serial port as an asynchronous port.

Solution

How it works

This design relies on the fact that received RS-232 signals always start with a “start bit” that is not part of the data and end with one or more “stop bits” that are also not part of the data. This design keeps the receiver turned off and an interrupt (also tied to the receive line) turned on when not receiving a character. When the interrupt goes off, this signals a start bit on the line. The code then turns the interrupt off and the receiver on; the data comes in as a normal 8-bit character. The stop bits assure the TMS320C30 has time to handle the data before the next character.

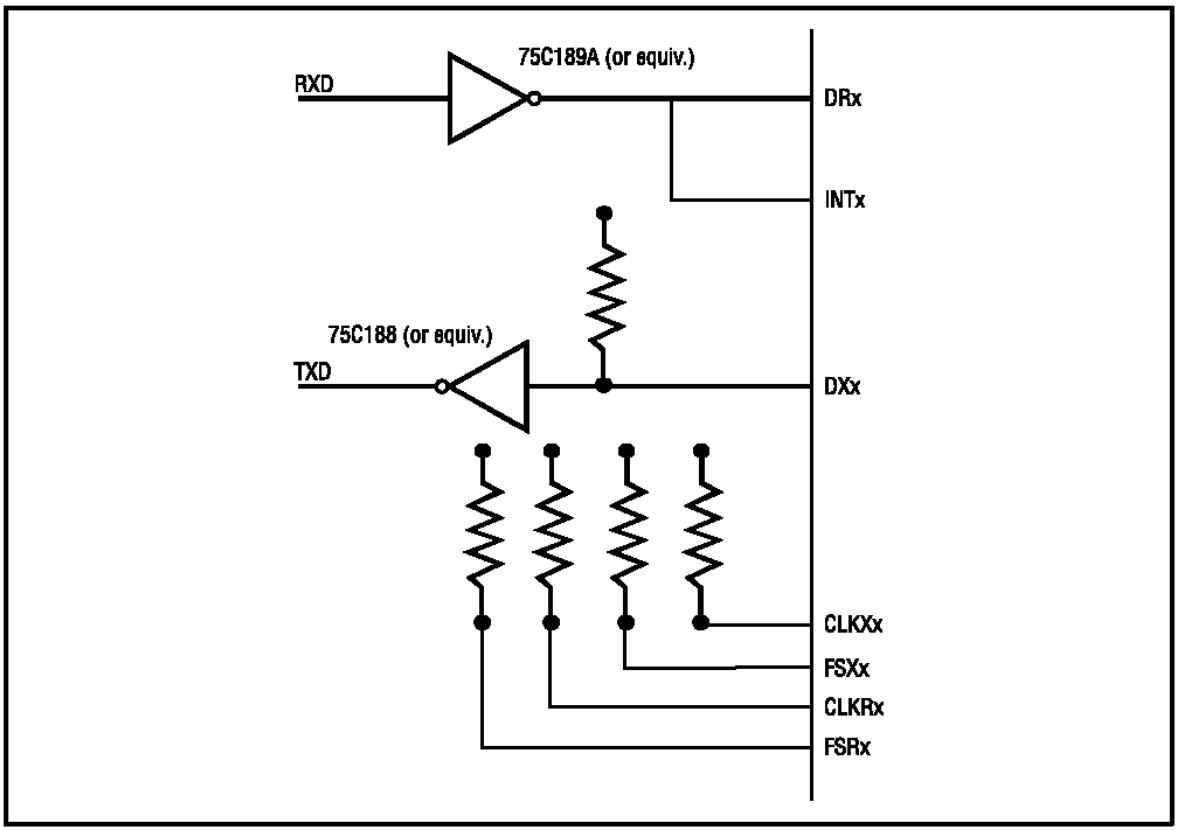
The transmitter basically frames the data into 16-bit words, adding a start bit, the character to send, and the stop bits. This will result in up to 6 clock cycles (RS-232 clock rate) where bandwidth on the channel is “wasted.” (Think of it as having 7 stop bits. That’s kind of how it works.) A more efficient (but more complicated) design could be done, but was not necessary for my project. Characters go in and out the serial port “backwards” from the RS-232 method; they must be bit-swapped to be correct.

The serial port is set up as a continuous transmit normal port. Frame syncs are not used and internal clocks are used for the serial port timing; these are timed of the TMS320C30 clock and need to be adjusted if clock rates change.

Hardware

Little hardware design needs to be done to handle this, basically just wire the serial port to the TMS320C30 properly.

Figure 1. Schematic Diagram



Notice that the received signal is tied to DRx (Data Receive) and also INTx (any of the TMS320C30 interrupts). DXx (Data TRANSmit) must be pulled high to avoid having to have the TMS320C30 constantly supply “1” on the line when it has nothing to transmit. The clocks and frame syncs are not used.

Software

The real meat of this design lies in the software. It must handle the interrupts and port setups and the queuing of the data. Actual code and descriptions follow in this article. The code ran under an operating system written by me, but the operation of the OS routines should be obvious.

Transmitter

The transmitter does not do very much; just frames the data properly, waits for the transmitter to be free, and sends the data. The transmitter interrupt also drove the OS timer tick; therefore the transmitter was constantly driven with data even when idle.

Receiver

The receiver does a lot more than the transmitter; some interrupt tricks supply the necessary “sync to async” conversion. Normally the serial port receiver is turned off. An interrupt comes in (the rec_coming interrupt) when a start bit comes in the receiver. This will turn off the rec_coming interrupt and start the receiver. The next 8 bits coming in the serial port should be the character desired. After these 8 bits are received the rec0 interrupt goes off. This will handle the received character; turn off the receiver, and turn back on the rec_coming interrupt to wait for the next start bit.

Example 1. Code Listing

```
/*****************************************************************************/
/*
 * io.c - The I/O routines and tasks to handle I/O to the C30 serial port.
 */

#include “monitor.h”
#include “debug.h”
#include “io.h”

Queue_Id            gets_queue;
Queue_Id            rec_int_queue;
Queue_Id            io_state;
int                 rec_ready;

static   Queue_Id   wait_rec_int[2];
static   Queue_Id   wait_rec_cmd[2];

/*
 * invert_8 - swaps the bits in the 8 bit character supplied.
 */
char invert_8(inchar)
    char inchar;
{
    char outchar;

    outchar = 0;
    if (inchar & 0x01)
    {
        outchar |= 0x80;
    }
    if (inchar & 0x02)
    {
        outchar |= 0x40;
    }
    if (inchar & 0x04)
    {
        outchar |= 0x20;
    }
    if (inchar & 0x08)
    {
        outchar |= 0x10;
    }
    if (inchar & 0x10)
    {
        outchar |= 0x08;
    }
    if (inchar & 0x20)
    {
```

```
        outchar |= 0x04;
    }
    if (inchar & 0x40)
    {
        outchar |= 0x02;
    }
    if (inchar & 0x80)
    {
        outchar |= 0x01;
    }
    return(outchar);
}
/*
 * The get string task. This receives request to receive strings then
 * receives them and sends the result back to the requesting task.
 */
void
gets_task()
{
    unsigned int my_tid;
    unsigned int msg;
    unsigned int tid;
    unsigned int qid;
    unsigned int dummy1;
    unsigned int dummy2;
    Buffer_Id    buf;
    void         *bufptr;

    char outbuf[3];

    char *out_loc;
    unsigned int count;
    unsigned int max_size;
    int finished;
    unsigned int c;

    io_state = NODEBUG_STATE;

    my_tid = 0;

    os_task_inquiry(&my_tid, NULL);  /* Get my task id (and therefore my) */
                                     /* main queue id. */
    os_create_queue(&gets_queue);    /* Create another queue for requests */
                                     /* to get data. */

    rec_int_queue = my_tid;          /* My main queue same as tid */

    wait_rec_int[0] = rec_int_queue; /* Set up queue lists for wait queues * /
    wait_rec_int[1] = END_QUEUE;

    wait_rec_cmd[0] = gets_queue;
    wait_rec_cmd[1] = END_QUEUE;

    while (TRUE)
    {
        rec_ready = FALSE;           /* Not receiving any data here */
                                     /* Wait for someone to request a string */
        os_wait_fetch(wait_rec_cmd, &msg, &buf, &bufptr, &tid, &qid);
        if (buf != NO_BUFFER)
    {
        os_free_buffer(buf);
    }
```

```
    rec_ready = TRUE;               /* Now we are receiving data */
    /*
     * The following is not 32-bit clean, but it doesn't matter for
     * 'C30s
     */
    max_size = (msg > 24) & 0xff;    /* Get the num bytes to receive */
    out_loc = ((char *) (msg & 0xffffff)); /* Get the address to put */
                                     /* the string in. */
    count = 0;
    finished = FALSE;
    while (!finished)
    {
        if (count = max_size) /* If all the data is in, send a msg */
        { /* back to the requestor */
            *out_loc = '\0';
            finished = TRUE;
            os_put_queue(REC_FINISHED, NO_BUFFER, tid);
        }
        else
        {
            /* Wait for the receiver to send me some data */
            os_wait_fetch(wait_rec_int, &c, &buf, &bufptr, &dummy1, &dummy2);
            if (buf != NO_BUFFER)
            {
                os_free_buffer(buf);
            }
            outbuf[0] = c; /* Put the received data into a buf */
            outbuf[1] = '\0'; /* so it can be echoed. */
            puts(outbuf); /* Echo the data */
            if (c == '\n') /* If a newline is received, finish */
            { /* the receive. */
                *out_loc = '\0';
                finished = TRUE;
                os_put_queue(REC_FINISHED, NO_BUFFER, tid);
            }
            else /* else put the character into the */
            { /* buffer. */
                *out_loc = c;
                out_loc++;
                count++;
            }
         }
      }
   }
}

/*
 * Receive handler. This routine is called by the interrupt handler that
 * is called when a byte is received from the com port.
 */
void
rec_hndl()
{
    int rec_char;

    regioncount = 1;

    /* Data from RS-232 is backwards, flip it around */
    rec_char = invert_8((*RECLOC) & 0xff);
    if (rec_char == 0x0d) /* Map ctrl-m to newline (No raw mode!) */
    {
        rec_char = '\n';
```

```
    }
    if (io_state == DEBUG_STATE) /* If the debugger is on, send all */
    { /* data to it. */
        os_put_queue(rec_char, NO_BUFFER, debug_q);
    }
    else if (rec_char == 0x03) /* A ctrl-c activates the debugger. */
    {
        io_state = DEBUG_STATE;
        os_start_task(debug_tid);
    }
    else if (rec_ready) /* Send data to the gets task if it wants it. */
    {
        os_put_queue(rec_char, NO_BUFFER, rec_int_queue);
    }

    regioncount = 0;
}

#define XMTLOC      ((int *) 0x808048)
#define RECLOC      ((int *) 0x80804c)
#define XMT_PRT_CTL ((int *) 0x808042)

Queue_Id            puts_queue;
Queue_Id            xmt_int_queue;

static Queue_Id     wait_xmt_int[2];
static Queue_Id     wait_xmt_cmd[2];

int                 xmt_data;

/*
 * The put string routine. This task will put strings out to the serial port.
 */
void
puts(string)
    char *string;
{
    Task_Id         my_tid;
    Queue_Id        wait_fini[2];
    unsigned int    msg;
    Task_Id         tid;
    Queue_Id        qid;
    Buffer_Id       buf;
    void            *bufptr;

    my_tid = 0;
    os_task_inquiry(&my_tid, NULL);  /* Get my task id. */
    wait_fini[0] = my_tid;           /* Use my task id as the queue to */
    wait_fini[1] = END_QUEUE;        /* receive xmit ready messages. */

    /* Send a pointer to the string to the transmit task. */
    os_put_queue((unsigned int) string, NO_BUFFER, puts_queue);

    /* Wait for it to respond. */
    os_wait_fetch(wait_fini, &msg, &buf, &bufptr, &tid, &qid);
    if (buf != NO_BUFFER)
    {
        os_free_buffer(buf);
    }
    /* Ignore all messages that are not a send finished from the xmit task */
    while (msg != SEND_FINISHED)
    {
```

```
        os_wait_fetch(wait_fini, &msg, &buf, &bufptr, &tid, &qid);
        if (buf != NO_BUFFER)
        {
            os_free_buffer(buf);
        }
    }
}

/*
 * The put string task. This task will wait for strings on its input queue
 * and transmit them to the serial port.
 */
puts_task()
{
    Task_Id         my_tid;
    char            *msg;
    Task_Id         tid;
    Queue_Id        qid;
    unsigned int    dummy1, dummy2, dummy3;
    int             newline_flag;
    Buffer_Id       buf;
    void            *bufptr;

    my_tid = 0;

    os_task_inquiry(&my_tid, NULL);  /* Get my task id. */
    os_create_queue(&puts_queue);    /* Create a queue to get send requests*/
    xmt_int_queue = my_tid;          /* My queue to get transmitter */
                                     /* interrupt messages. */

    wait_xmt_int[0] = xmt_int_queue; /* Set up receive queues. */
    wait_xmt_int[1] = END_QUEUE;

    wait_xmt_cmd[0] = puts_queue;
    wait_xmt_cmd[1] = END_QUEUE;

    newline_flag = FALSE;

    xmt_data = FALSE;

    while(TRUE)
    {
        /* Wait for a string to transmit. */
        os_wait_fetch(wait_xmt_cmd, (unsigned int *) &msg, &buf, &bufptr, &tid,
&qid);
        if (buf != NO_BUFFER)
        {
            os_free_buffer(buf);
        }

        /*
         * Ok, now I am transmitting. Wait for the transmitter to tell me
         * that I can send some data.
         */
        xmt_data = TRUE;
        os_wait_fetch(wait_xmt_int, &dummy1, &buf, &bufptr, &dummy2, &dummy3);
        if (buf != NO_BUFFER)
        {
            os_free_buffer(buf);
        }
        /*
         * Send the whole message. Make sure to send the last new line
```

```
         * even if currently pointing to the EOS character.
         */
        while ((*msg != '\0') || (newline_flag))
        {
            if (newline_flag)    /* If transmitting a newline, (ctrl-j), also */
            {                    /* send a cariage return (ctrl-m). */
            *XMTLOC = (((int) invert_8((char)0x0d)) & 0xfeff) | 0xfe00;
            newline_flag = FALSE;
            }
            else
            {
                if (*msg == '\n') /* If a newline, set up to send a */
                { /* ctrl-m next. */
                newline_flag = TRUE;
                }
                    /*
                     * Put the character into the output buffer. The first
                     * 7 bits are transmitted as 1, the next is the start bit,
                     * the rest is the character.
                     */
                    *XMTLOC = (((int) invert_8(*msg)) & 0xfeff) | 0xfe00;
                    msg++;
                }
                /*Wait for the transmitter to tell me I can send the next char*/
                os_wait_fetch(wait_xmt_int,&dummy1,&buf,&bufptr,&dummy2,&dummy3);
                if (buf != NO_BUFFER)
                {
                    os_free_buffer(buf);
                }
            }
            xmt_data = FALSE; /* No longer receiveing data. */
            *XMTLOC = 0xffff; /* Prime the transmitter to send ones. */

            /* Inform the requestor that the send is finished. */
            os_put_queue(SEND_FINISHED, NO_BUFFER, tid);
        }
    }
    /*
     * transmit interrupt handler. This routine is called whenever the transmit
     * interrupt for the serial port goes off. It continuously keeps the
     * transmitter primed because the transmit interrupt is also used as the
     * clock interrupt.
     */
void
xmt_hndl()
{
    if (xmt_data) /* If the puts task is waiting interrupt info... */
    {
        regioncount = 1;   /* Interrupts should already be turned off, */
                           /* set the critical region count to */
                           /* reflect that. */
        /*
         * Send a message to the puts task to tell it to send the next
         * char. If the send fails, go ahead and prime the transmitter.
         */
        if (os_put_queue(0, NO_BUFFER, xmt_int_queue) != 0)
        {
            *XMTLOC = 0xffff;
        }
        regioncount = 0;
    }
    else            /* If the puts task is not sending, prime the transmitter */
```

```
    {
        *XMTLOC = 0xffff;
    }
}

/*************************************************************************/

;ioasm.asm - the assembly langage support routines for I/O handling for the
;            C30 serial port.

             .global   xmt0
             .global   rec0
             .global   rec_coming
             .global   _init_io
             .global   _rec_hndl
             .global   _xmt_hndl
             .global   _os_tick, save_task, restore_task

;* xmt0 - handle an interrupt from the serial port transmitter. This also
;*        calls the OS tick routine. Note that the save and restore tasks
;*        are called because this can result in a task switch.
             .text
xmt0
             CALL   save_task

             CALL   _xmt_hndl

             CALL   _os_tick

             CALL   restore_task

             RETI

rec_ser_cnt .word   808040h      ; Address of serial port status register
s_recc_int  .word   000000002h   ; Mask for the receive interrupt
c_recc_int  .word   0fffffffdh   ; Inverted mask to clear the rec int.
reset_rec   .word   0f7ffffffh   ; Mask to write a 0 to the rec reset
unreset_rec .word   008000000h   ; Unreset the receiver.

;* rec0 - Handle an interrupt from the serial port receiver to inform it
;*        of the receipt of a byte on the serial port. This routine will
;*        turn off the receiver and restore the interrupt telling it that
;*        a byte is about to come.
rec0
             CALL   save_task

             LDP    @rec_ser_cnt,DP
             LDI    @rec_ser_cnt,AR0

             LDI    *+AR0(0),R0         ;Reset the receiver
             AND    @reset_rec,R0
             STI    R0,*+AR0(0)
             AND    @c_recc_int,IF      ;clear receive coming interrupt
             OR     @s_recc_int,IE      ;enable the interrupt for the next byte
             CALL   _rec_hndl

             CALL   restore_task
             RETI
;* rec_coming - This interrupt handle is called by INT1. It is tied to the
;*      receive data line, it will be called when the start bit is received
;*      for a character of information. It will turn on the serial receiver
;*      (and the serial receiver interrupt) and turn its own interrupt off.
```

```
rec_coming
            PUSH   ST
            PUSH   R0
            PUSH   DP
            PUSH   AR0

            LDP    @rec_ser_cnt,DP
            LDI    @rec_ser_cnt,AR0

            AND    @c_recc_int,IE   ;do not allow the receive coming interrupt

            LDI    *+AR0(0),R0      ;Ready the receiver
            OR     @unreset_rec,R0
            STI    R0,*+AR0(0)

            POP    AR0
            POP    DP
            POP    R0
            POP    ST
            RETI

gl_prt_cnt  .word   0068400c4h       ; Initial setup for the serial port
                                     ;  status register. This sets the
                                     ;  following things:
                                     ;    FSX is output.
                                     ;    Fixed data rate signalling
                                     ;    Standard frame sync mode
                                     ;    Internal xmit clk
                                     ;    Internal rec clk
                                     ;    Active high DX and DR
                                     ;    XLEN - 16 bits
                                     ;    RLEN - 8 bits
                                     ;    Transmitter interrupt enabled
                                     ;    Receive interrupt enabled
                                     ;    Activate the transmitter
                                     ;    Deactivate the receiver

x_prt_cnt   .word   000000111h       ; Setup for the transmit port control
                                     ;  register. Set all the transmit
                                     ;  pins as serial port pins.

r_prt_cnt   .word   000000111h       ; Setup for the receive port control
                                     ;  register. Set all the receive
                                     ;  pins as serial port pins.

tmr_cnt     .word   0000003cfh       ; Setup for the timer control reg.
                                     ;    Starts the timer
                                     ;    Free run the timer (no hold)
                                     ;    Clock mode
                                     ;    Internal clock source

tmr_per     .word   00434042Ah       ; Timer periods. This is 1076 for
                                     ;  the receiver, which is a little
                                     ;  slow. This makes sure we don't
                                     ;  shift in time before the bits.
                                     ;  These also assume a 20.48MHZ clock
                                     ;  in the C30; these values will have
                                     ;  to be adjusted for different clock
                                     ;  rates.

enab_int    .word   000000032h       ; Enable serial xmit, serial recieve,
                                     ;  and int 1 for serial port stuff.
```

```
first_xmt   .word   00000ffffh

_init_io
            LDP     @rec_ser_cnt,DP
            LDI     @rec_ser_cnt,AR0

            LDI     @x_prt_cnt,R0    ; Set up the transmit control port.
            STI     R0,*+AR0(2)

            LDI     @r_prt_cnt,R0    ; Set up the receive control port.
            STI     R0,*+AR0(3)

            LDI     @tmr_per,R0      ; Set the timer period register.
            STI     R0,*+AR0(6)

            LDI     @tmr_cnt,R0      ; Set the timer control register.
            STI     R0,*+AR0(4)

            LDI     @gl_prt_cnt,R0   ; Set the global serial control register.
            STI     R0,*+AR0(0)

            OR      @enab_int,IE     ; Enable interrupts.

            LDI     @first_xmt,R0    ; Start the transmitter sending 1's
            STI     R0,*+AR0(8)

            RETS
```