A Driver for the NS16550 DUART Which Runs on an NS32GX32 CPU

National Semiconductor
Application Note 660
Coby Hanoch
December 1989

1.0 INTRODUCTION

This application note supplies a software program which serves as a driver for the NS16550 DUART. The driver runs on the NS32GX32 CPU, and should run with no modifications on all other Embedded System Processors.

The driver is useful for anyone who is writing a monitor or system kernel which will run on a system with an NS32GX32 and NS16550. It is well documented and can be modified according to the user's needs.

Drivers of this sort are hard to develop, since they are very hard to debug. In order to debug such a driver there needs to be some sort of a communication line to the developed system. This line is used in order to load the driver into the memory of the developed system. But, in this case the communication line is controlled by the NS16550, which doesn't have a driver yet.

The NS16550 is a popular standard DUART which has 2 communication ports and is useful for serial I/O operations. It also contains a FIFO on each port allowing synchronous communication without losing data.

The driver has the following features:

— It controls both ports of the DUART.

— It can be told to echo its input, and if CR should be echoed as CRLF. Echoing is a required feature by many systems.

— It can be told to ignore echo of its output, and if CR is echoed as CRLF.

— It can be told to treat ^S (XOFF), ^Q (XON) and ^H (Backspace). These control characters are the standard communication signals to pause and restart communication (^S and ^Q respectively), and to erase the last character sent (^H).

— It can be told to wait until input arrives or only till a specified timeout has passed. These are the two popular modes of operation in communication systems.

— It can read/write a single character or a whole line.

— It can be told to read a specified number of characters or till a CR (and return the length).

The driver is designed for synchronous communication.

The driver is written in a very portable way so that it can be transported to other Embedded System Processors with no modifications.

It is compiled and linked by the GNX package supplied by NSC.

2.0 USAGE

The driver consists of 3 files:

1. An assembler source file containing the initialization routine (int_16550).
2. A C source file containing the functions to read/write from/to the DUART.
3. A CPP include file used by both source files, defining global constants. These constants define the addresses of the DUART on the board and the requested baud rate.

In addition the driver's makefile (for UNIX systems) is supplied, and a demo program which runs it.

The user should perform the following steps in order to use the driver:

1. Modify the include file to define the DUART address on his board. These are the lines:
   #define USART_0_ADDR <address of port 0>
   #define USART_1_ADDR <address of port 1>
2. Modify the include file to define the baud rate on his board. The baud rate is defined according to the NS16550 data sheet. For example: 12 signals 9600 baud at 1.8432 MHz crystal. This is the line:
   #define BAUD_RATE <baud rate>
3. Modify the include file to define the timeout constant according to his needs. The read operations can be instructed to wait for input. This constant defines the number of times a read will be attempted before a failure is reported in this mode. This is the line:
   #define IO_TIMEOUT <number of attempts>
4. In his program he should add a call for the init_16550 routine as one of the first things it does. This must, of course, come before any attempt to read or write from the NS16550.
5. The driver uses 4 global variables which should be initialized:
   world_echo— Specifies if the world (the hardware on the other side of the communication line) echoes what it receives.
   world_crlf— Specifies if the world echoes a CR as CRLF.
   board_echo— Specifies if the board (on which the driver and the NS16550 reside) should echo what it receives.
   board_crlf— Specifies if the board should echo CR as CRLF.
   Each of these variables is a 2 element array, whose first element refers to port 0 and the second to port 1. A value of 0 specifies FALSE and a value of 1 specifies TRUE.
   Example: world_echo[0] = 0;
   This instruction informs the driver that the world echoes any character it receives from port 0.
6. The user can call the following functions from his program: read_line, write_line, read_char, write_char. They are defined in the next section.
7. Compile the driver (no special switches needed) and link it to his program.

Note: If the user calls the init_16550 function from a reset function (when there is still no confidence that the memory is operative), the function can be jumped to, and passed the return address in a register. The ret instruction at the end of the function should be replaced by a jump 0(r7) for example.

RRD-B30M75/Printed in U. S. A.

3.0 INTERFACE

The interface to the driver is done via calls to a set of functions. Following is a description of each of these functions and its parameters.

Note: The type 'str' is defined to be 'char*'

1. void read__line (port, line, len, err);
   int port;
   str line, err;
   unsigned *len;

Description: Reads a line from a port.

Parameters: port—The port number (0 or 1) to be read from.
line—A pointer to a character array in which the line read will be returned.
len—The number of characters to be read. 0 specifies to read until a CR.
err—Contains an error message, if an error occurred during the read. If it is null, no error occurred.

2. void write__line (port, line);
   int port;
   str line;

Description: Writes a line to a port.

Parameters: port—The port number (0 or 1) to be written to.
line—A pointer to a character array in which the line to be written is placed. The line should end with a CR.

3. void read__char (port, ch, wait, err);
   int port;
   char *ch;
   boolean wait;
   str err;

Description: Reads a character from a port.

Parameters: port—The port number (0 or 1) to be read from.
ch—A pointer to a character in which the character read will be returned.
wait— If TRUE, will wait until a character arrives (if one is not present in the FIFO already). If FALSE, will attempt to read IO__TIMEOUT times before giving up and returning an error.
err—Contains an error message, if an error occurred during the read. If it is null, no error occurred.

4. void write__char (port, ch);
   int port;
   char ch;

Description: Writes a character to a port.

Parameters: port—The port number (0 or 1) to be read from.
ch—A pointer to a character in which the character to be written is at.

4.0 THE FILES

Attached are the source files.

4.1 16550.h—The CPP Include File

```
/*****************************************************************************
 *
 *                                16550 definitions
 *
 * This module contains the CPP constants definitions for the 16550 driver.
 *
 *****************************************************************************/

/*
 * Useful characters
 */

#define CR            '\r'
#define LF            '\n'
#define CTRL_Q        '\021'
#define CTRL_S        '\023'
#define CTRL_H        '\010'

/*
 * Maximum line size
 */

#define LINE_SIZE       270

/*
 * The USART addresses on the board.
 * NOTE: These addresses should be modified by the user to the addresses on
 *       the tested board.
 */

#define USART_0_ADDR    0xF00000
#define USART_1_ADDR    0xF00008

/*
 * The maximum time to wait for character read (number of tries).
 */

#define IO_TIMEOUT    ((int) 0x90000)

/*
 * The baud rate as defined in the NS16550 data sheet.
 * (12 signals 9600 baud at 1.8432 MHz crystal)
 * NOTE: The baud rate should be modified by the user to fit his needs.
 */

#define BAUD_RATE     12
```

TL/EE/10601–1

4.2 16550_asm.s—The Assembly File

```
#*****************************************************************************
#
#                                16550_asm.s
#
# This file contains the init_16550 routine.
#
# Note that comment lines must begin with a blank so cpp will disregard
# them.
#
#*****************************************************************************

#include "16550.h"

        .set    usrt0,  USART_0_ADDR    # usrt0 start address
        .set    usrt1,  USART_1_ADDR    # usrt1 start address
        .set    ier,    1               # interrupt enable register   (IER)
        .set    ffc,    2               # fifo control register       (FCR)
        .set    com,    3               # usrt line control register  (LCR)
        .set    modem,  4               # modem control register      (MCR)
        .set    ck,     0               # baud rate counter      (DLL, DLM)

        .set    cke,    0x80            # clock enable(dlab=1)
        .set    modr1,  0x3             # async, 8bits, no parity, 1 stop bit
        .set    modr2,  0x3             # activate receive, transmit signals

        .set    baud,   BAUD_RATE       # the baud rate

#-----------------------------------------------------------------------------
#
#                                   Init_16550
#
# This is the init routine for the NS16550 UART.
# This routine must be written in assembly. It doesn't use the RAM (or stack
# since it is still RAM), since it is usually called by the diagnostics
# routine. For this reason you would usually like to jump to it directly
# and have it jump back. A register can be used to contain the return address.
# The baud rate (in the format requested by the UART) can be passed as a
# parameter in R6 instead of being loaded directly.
# The only register used by this routine is R6.
#
#-----------------------------------------------------------------------------

_init_16550::
        movb    $cke,@usrt0+com         # enable clock divisor accsess
        movb    $cke,@usrt1+com         # same for usrt1
        movw    $baud,@usrt0+ck         # set baud rate
        movw    $baud,@usrt1+ck
        movw    @usrt0+ck,r6
        movb    $modr1,@usrt0+com       # init line mode register
        movb    $modr1,@usrt1+com
        movqb   0,@usrt0+ier            # disable interrupts
        movqb   0,@usrt1+ier
        movqb   3,@usrt0+ffc            # clear fifo
        movqb   3,@usrt1+ffc
        movqb   1,@usrt0+ffc            # enable fifo
        movqb   1,@usrt1+ffc
        movb    $modr2,@usrt0+modem     # activate receive transmit
        movb    $modr2,@usrt1+modem
        ret     0
#       jump    0(r7)                   # suggested return via register
```

TL/EE/10601–2

4.3 16550.c—The Driver Itself

```
/*****************************************************************************
 *
 *                                  16550_driver
 *
 * This module contains a serial I/O driver for the NS16550.
 * Throughout this source, "world" refers to the outer world, "board" refers to
 * the board running this program.
 *
 *****************************************************************************/


#include <strings.h>
#include "16550.h"

/*---------------------------------------------------------------------------
 *
 *                              Constants definition
 *
 *---------------------------------------------------------------------------*/

/*
 * Bit macroes
 */

#define BIT_ON(bit, reg)   ((boolean) (((reg) & (unsigned) (1 << (bit))) != 0))
#define BIT_OFF(bit, reg)  ((boolean) (((reg) & (unsigned) (1 << (bit))) == 0))

/*
 * Boolean operators
 */

#define NOT    !
#define AND    &&
#define OR     ||

/*---------------------------------------------------------------------------
 *
 *                              Types definition
 *
 *---------------------------------------------------------------------------*/

typedef unsigned char  byte;
typedef char *str;
typedef char l_str[LINE_SIZE];

typedef enum {
        FALSE,
        TRUE
} boolean;
```

TL/EE/10601–3

```
/*---------------------------------------------------------------------------
 *
 *                              Variables definition
 *
 *---------------------------------------------------------------------------*/

/*
 * Communications control boolean variables, per port.
 */

char world_echo[2],             /* The world echoes what it receives        */
     world_crlf[2],             /* The world echoes a CR as CRLF            */
     board_echo[2],             /* The board should echo what it receives   */
     board_crlf[2];             /* The board should echo CR as CRLF         */

/*
 * The structure of the NS16550 register file. Note that in case of a collision
 * between two registers, the first register is defined here and the second
 * register is defined to be the same.
 */

typedef volatile struct {
        byte    iop;            /* Receive/Transmit register     (RBR, THR) */
        byte    ier;            /* Interrupt enable register          (IER) */
        byte    fcr;            /* Fifo  control register             (FCR) */
        byte    lcr;            /* Line  control register             (LCR) */
        byte    mcr;            /* Modem control register             (MCR) */
        byte    lsr;            /* Line  status  register             (LSR) */
        byte    msr;            /* Modem status  register             (MSR) */
        byte    scr;            /* Scratch register                   (SCR) */
} uart_regs_type;

#define iir     fcr             /* Interrupt ident register           (IIR) */
#define dll     iop             /* Divisor latch LSB (dlab=1)         (DLL) */
#define dlm     ier             /* Divisor latch MSB (dlab=1)         (DLM) */

/*
 * Frequently used bits of different registers.
 */

#define OUT_READY    5          /* Transmit ready bit of LSRx               */
#define IN_READY     0          /* Receive  ready bit of LSRx               */

#define PORT_0_STATUS  uart_0_regs->lsr
#define PORT_0_IO      uart_0_regs->iop
#define PORT_1_STATUS  uart_1_regs->lsr
#define PORT_1_IO      uart_1_regs->iop

/*
 * The USART registers are memory mapped to structures.
 */

static uart_regs_type  *uart_0_regs = (uart_regs_type *) USART_0_ADDR;
static uart_regs_type  *uart_1_regs = (uart_regs_type *) USART_1_ADDR;
```

TL/EE/10601–4

```
/*-------------------------------------------------------------------------
 *
 *                              Functions definition
 *
 *-------------------------------------------------------------------------*/

       void read_line  (/* port, line, len, err */);
       void write_line (/* port, line */);
       void read_char  (/* port, ch, wait, err */);
       void write_char (/* port, ch */);
static boolean get_char(/* ch, status, io_port, wait, err */);
static void get_line   (/* port, line, len, err */);


/*-------------------------------------------------------------------------
 *
 *              Read_char(port, ch, wait, err)
 *
 * This function reads one character from the port specified.
 * ^S and ^Q will be treated appropriately.
 * If wait is FALSE an error will be signaled if a character doesn't arrive
 * before timeout.
 * If the board should echo the input, this function does it.
 *
 *-------------------------------------------------------------------------*/

void read_char(port, ch, wait, err)
    int      port;
    char     *ch;
    boolean  wait;
    str      err;
{
    boolean  ch_ok;
    volatile byte  *status, *io_port;

    /*
     * Set pointers according to port
     */

    if (port == 0) {
        status  = @PORT_0_STATUS;
        io_port = @PORT_0_IO;
    }
    else {
        status  = @PORT_1_STATUS;
        io_port = @PORT_1_IO;
    }

    /*
     * Loop until we have a good character
     */

    ch_ok = FALSE;
    while (NOT ch_ok) {
        ch_ok = get_char(ch, status, io_port, wait, err);
        if (err[0] != '\0') {
            return;
        }
    }
```

TL/EE/10601–5

```
    /*
     * Echo the character if needed.
     */

    if (board_echo[port]) {
        write_char(port, *ch);
        if ((*ch == CR) AND (board_crlf[port])) {
            write_char(port, LF);
        }
    }

} /* read_char */

/*---------------------------------------------------------------------------
 *
 *           Get_char(ch, status, io_port, wait, err)
 *
 * This function reads and processes a character. The ports stauts register
 * and io_port are used.
 * The function returns TRUE if the character is ok.
 *
 *---------------------------------------------------------------------------*/

static boolean get_char(ch, status, io_port, wait, err)
    volatile byte *status, *io_port;
    char    *ch;
    boolean wait;
    str     err;
{
    int     i;

    /*
     * Wait until we are flagged that a character arrived.
     */

    if (wait) {
        while (BIT_OFF(IN_READY, *status)) {
        }
    }
    else {
        for (i = 0; i < IO_TIMEOUT; i++) {
            if (BIT_ON(IN_READY, *status)) {
                break;
            }
        }
        if (i == IO_TIMEOUT) {
            strcpy(err, "ERR TIMEOUT");
            return(FALSE);
        }
    }

    /*
     * Read the character.
     */

    *ch = *io_port;
```

TL/EE/10601–6

```
    /*
     * If it is a ^Q, ignore it.
     */

    if (*ch == CTRL_Q) {
        return(FALSE);
    }

    /*
     * If the character is a ^S, keep reading until a ^Q arrives.
     * Ignore timeout.
     * If the character is not a ^S, signal that it is ok.
     */

    if (*ch == CTRL_S) {
        while (*ch != CTRL_Q) {
            while (BIT_OFF(IN_READY, *status)) {
            }
            *ch = *io_port;
        }
        return(FALSE);
    }

    return(TRUE);

} /* get_char */

/*---------------------------------------------------------------------------
 *
 *                  Write_char(port, ch)
 *
 * This function writes one character to the port specified.
 * If the world echoes the characters, this function rereads the character.
 *
 *---------------------------------------------------------------------------*/

void write_char(port, ch)
    int   port;
    char  ch;
{
    volatile byte *status, *io_port;
    char  temp;
    l_str temp_message;

    /*
     * Set pointers according to port
     */

    if (port == 0) {
        status  = &PORT_0_STATUS;
        io_port = &PORT_0_IO;
    }
    else {
        status  = &PORT_1_STATUS;
        io_port = &PORT_1_IO;
    }
```

TL/EE/10601–7

```
    /*
     * Check if there is a character waiting on the input port.
     * If there is, and it is a ^S, keep reading until a ^Q is read.
     * Otherwise, ignore it.
     */

    if (BIT_ON(IN_READY, *status)) {
        temp = *io_port;
        if (temp == CTRL_S) {
            while (temp != CTRL_Q) {
                while (BIT_OFF(IN_READY, *status)) {
                }
                temp = *io_port;
            }
        }
    }

    /*
     * Write the character.
     */

    while (BIT_OFF(OUT_READY, *status)) {
    }
    *io_port = ch;

    /*
     * If the world echoes the character, reread it.
     */

    if (world_echo[port]) {
        read_char(port, &temp, FALSE, temp_message);
        if ((ch == CR) AND (world_crlf)) {
            read_char(port, &temp, FALSE, temp_message);
        }
    }

} /* write_char */

/*-------------------------------------------------------------------------
 *
 *             Read_line(port, line, len, err)
 *
 * This function controls the reading of a line from the requested port.
 * len = 0 specifies that the reading should continue till a CR is reached.
 * It returns the line and the port from which the command was read.
 *
 *-------------------------------------------------------------------------*/

void read_line(port, line, len, err)
    int      port;
    str      line, err;
    unsigned *len;
{

    err[0] = '\0';
```

TL/EE/10601–8

```
    /*
     * Keep trying to read a line, until success.
     */

    get_line(port, line, len, err);

    while (err[0] != '\0') {
        write_line(port, err);
        get_line(port, line, len, err);
    }

} /* read_line */

/*---------------------------------------------------------------------------
 *
 *              Get_line(port, line, len, err)
 *
 * This function reads a line from the port specified.
 * Length = 0 specifies that read should continue to a CR.
 * In any case, no more characters than the size of line are read.
 * Upon return, length will contain the number of characters read.
 *
 *---------------------------------------------------------------------------*/

static void get_line(port, line, len, err)
    int      port;
    str      line, err;
    unsigned *len;
{
    boolean  stop_on_cr, wait;
    char     ch;
    int      i;

    /*
     * Check the length specified.
     */

    stop_on_cr = FALSE;
    if (*len == 0) {
        stop_on_cr = TRUE;
        *len       = LINE_SIZE;
    }

    /*
     * Read the line. Treat backspace (^H) on the way.
     */
```

TL/EE/10601–9

```
    line[0] = '\0';
    for (i = 0; i < *len ; i++) {
        if (i == 0) {
            wait = TRUE;
        }
        else {
            wait = FALSE;
        }
        read_char(port, &ch, wait, err);
        if (err[0] != '\0') {
            *len = i;
            return;
        }
        if (ch == CTRL_H) {
            if (i > 0) {
                i -= 2;
                if (board_echo[port]) {
                    write_char(port, ch);
                    write_char(port, ' ');
                    write_char(port, ch);
                }
                continue;
            }
            continue;
        }
        if ((ch == CR) AND (stop_on_cr)) {
            line[i] = '\0';
            *len = i;
            return;
        }
        line[i] = ch;
    }

} /* get_line */

/*-------------------------------------------------------------------------
 *
 *                      Write_line(port, line)
 *
 * This function writes a line to the requested communication port.
 *
 *-------------------------------------------------------------------------*/

void write_line(port, line)

    int port;
    str line;
{
    int i;

    for (i = 0; i < strlen(line); i++) {
        write_char(port, line[i]);
    }

    write_char(port, CR);

} /* write_line */
```

TL/EE/10601–10

4.4 demo.c—The Demo Program

```
/*****************************************************************************
 *
 *                                  16550_demo
 *
 * This module contains an example of how to use the driver for the NS16550.
 * Throughout this source, "world" refers to the outer world, "board" refers to
 * the board running this program.
 *
 *****************************************************************************/

/*
 * Include the 16550 include file.
 */

#include "16550.h"

/*
 * Declare the used functions and global variables in the driver.
 */

void read_line (/* port, line, len, response */);
void write_line(/* port, line */);

char board_echo[2],              /* The board should echo what it receives   */
     board_crlf[2];              /* The board should echo CR as CRLF         */


main () {

static char     ln[LINE_SIZE],   /* The line read/written                    */
                resp[LINE_SIZE]; /* The response (error message)              */
static unsigned len=0;           /* The length of the line to be read        */

/*
 * Call the drivers init routine and initialize.
 */

init_16550();
board_echo[0] = 0;
board_crlf[0] = 0;
board_echo[1] = 0;
board_crlf[1] = 0;

/*
 * Write messages to port 0.
 */

write_line(0, "hello world\n");
write_line(0, "This is the 16550 driver test\n");
```

TL/EE/10601–11

```
/*
 * Read a line from port 0, until a CR.
 */

read_line(0, ln, &len, resp);
if (resp[0] != '\0') exit(1);
write_line(0, "line received\n");
write_line(0, ln);

/*
 * Ask the driver to echo all its input from port 0, and echo CR as CRLF.
 */

board_echo[0] = 1;
board_crlf[0] = 1;

/*
 * Read a line from port 0. Note that len contains the number of characters
 * read previously (and not 0), so the read will read the same number of
 * characters without reguarding CR.
 */

read_line(0, ln, &len, resp);
if (resp[0] != '\0') exit(1);

/*
 * Ask the driver to stop echoing CR as CRLF in port 0, and read a line
 * until a CR.
 */

board_crlf[0] = 0;
len = 0;
read_line(0, ln, &len, resp);
if (resp[0] != '\0') exit(1);

/*
 * Write a line to port 1, and read an answer.
 */

write_line(1, "hello port 1\n");
len = 0;
read_line(0, ln, &len, resp);
if (resp[0] != '\0') exit(1);

}
```

TL/EE/10601–12

4.5 makefile—The Makefile

```
#
# This makefile compiles the 16550 driver with the demo program.
# Type "make" to produce the executable called "16550".
#

OBJS    = demo.o 16550.o

SOURCES = demo.c 16550.c

H_DIR   = .


CC      = nmcc
AS      = nasm -c
CFLAGS  = -KCGX32 -g

16550: 16550_asm.o $(OBJS) 16550.h
        $(CC)  $(CFLAGS) -o 16550 16550_asm.o $(OBJS)

demo.o: demo.c

16550.o: 16550.c

16550_asm.0: 16550_asm.s 16550.h
        $(AS)  -o 16550_asm.o $(AS_FLAGS) 16550_asm.s
```

TL/EE/10601–13

Lit. # 100660

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-8090
Tel: 1(800) 272-9959
TWX: (910) 339-9240

National Semiconductor GmbH
Livry-Gargan-Str. 10
D-82256 Fürstenfeldbruck
Germany
Tel: (81-41) 35-0
Telex: 527649
Fax: (81-41) 35-1

National Semiconductor Japan Ltd.
Sumitomo Chemical
Engineering Center
Bldg. 7F
1-7-1, Nakase, Mihama-Ku
Chiba-City,
Ciba Prefecture 261
Tel: (043) 299-2300
Fax: (043) 299-2500

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductores Do Brazil Ltda.
Rue Deputado Lacorda Franco
120-3A
Sao Paulo-SP
Brazil 05418-000
Tel: (55-11) 212-5066
Telex: 391-1131931 NSBR BR
Fax: (55-11) 212-1181

National Semiconductor (Australia) Pty, Ltd.
Building 16
Business Park Drive
Monash Business Park
Nottinghill, Melbourne
Victoria 3168 Australia
Tel: (3) 558-9999
Fax: (3) 558-9998