

```

/*****
/*
/*          S E R U T I L . C
**-----**
/* Task      : Functions for direct access to
/*            serial port
**-----**
/* Author     : Michael Tischer / Bruno Jennrich
/* Developed on : 04/08/1994
/* Last update  : 04/07/1995
**-----**
/* COMPILER    : Borland C++ 3.1, Microsoft Visual C++ 1.5
/*****
#endifdef __SERUTIL_C          /* Can also be #included */
#define __SERUTIL_C

```

```

#include <conio.h>
#include "serutil.h"
#include "irqutil.h"
#include "win.h"

```

```

/*****
/* ser_UARTType : Get type of UART chip
**-----**
/* Input : iSerPort      - base port of interface being tested
/* Output : 0 (NOSER)    - no UART chip found
/*          1 (INS8250)   - INS8250 or INS8250-B chip
/*          2 (NS16450)   - INS8250A, INS82C50A, NS16450, NS16C450
/*          3 (NS16550A) - NS16550A chip
/*          4 (NS16C552) - NS16C552 chip
/*****

```

```

INT ser_UARTType( INT iSerPort )
{
    /*- Check base capabilities ----- */

    outp( iSerPort + SER_LINE_CONTROL, 0xAA ); /* Divisor latch set */
    if( inp( iSerPort + SER_LINE_CONTROL ) != 0xAA ) return NOSER;

    outp( iSerPort + SER_DIVISOR_MSB, 0x55 ); /* Specify divisor */
    if( inp( iSerPort + SER_DIVISOR_MSB ) != 0x55 ) return NOSER;

    outp( iSerPort + SER_LINE_CONTROL, 0x55 ); /* Clear divisor latch */
    if( inp( iSerPort + SER_LINE_CONTROL ) != 0x55 ) return NOSER;

    outp( iSerPort + SER_IRQ_ENABLE, 0x55 );
    if( inp( iSerPort + SER_IRQ_ENABLE ) != 0x05 ) return NOSER;

    outp( iSerPort + SER_FIFO, 0 ); /* Clear FIFO and IRQ */
    outp( iSerPort + SER_IRQ_ENABLE, 0 );
    if( inp( iSerPort + SER_IRQ_ID ) != 1 ) return NOSER;

    outp( iSerPort + SER_MODEM_CONTROL, 0xF5 );
    if( inp( iSerPort + SER_MODEM_CONTROL ) != 0x15 ) return NOSER;

    outp( iSerPort + SER_MODEM_CONTROL, SER_MCR_LOOP ); /* Looping */
    inp( iSerPort + SER_MODEM_STATUS );
    if( ( inp( iSerPort + SER_MODEM_STATUS ) & 0xF0 ) != 0 ) return NOSER;

    outp( iSerPort + SER_MODEM_CONTROL, 0x1F );
    if( ( inp( iSerPort + SER_MODEM_STATUS ) & 0xF0 ) != 0xF0 )
        return NOSER;

    outp( iSerPort + SER_MODEM_CONTROL, SER_MCR_DTR | SER_MCR_RTS );
}

```

```

outp( iSerPort + SER_SCRATCH, 0x55 ); /* Scratch register detected? */
if( inp( iSerPort + SER_SCRATCH ) != 0x55 ) return INS8250;
outp( iSerPort + SER_SCRATCH, 0 );

outp( iSerPort + SER_FIFO, 0xCF ); /* FIFO detected ? */
if( ( inp( iSerPort + SER_IRQ_ID ) & 0xC0 ) != 0xC0 ) return NS16450;
outp( iSerPort + SER_FIFO, 0 );
/* Alternate function register detected? */
outp( iSerPort + SER_LINE_CONTROL, SER_LCR_SETDIVISOR );
outp( iSerPort + SER_2FUNCTION, 0x07 );
if( inp( iSerPort + SER_2FUNCTION ) != 0x07 )
{
    outp( iSerPort + SER_LINE_CONTROL, 0 );
    return NS16550A;
}
outp( iSerPort + SER_LINE_CONTROL, 0 ); /* Reset */
outp( iSerPort + SER_2FUNCTION, 0 );
return NS16C552;
}

```

```

/*****
/* ser_Init : Initialize serial port */
/**-----**/
/* Input : iSerPort - base port of interface */
/*          being initialized. */
/*          lBaud - baud rate ( from 1 - 115200 ) */
/*          bParams - bit mask of remaining parameters */
/*          (s. SER_LCR... bits) */
/* Output : TRUE - port initialized successfully */
/*          FALSE - no port found */
*****/

```

```

INT ser_Init( INT  iSerPort, LONG lBaudRate, BYTE bParams )
{ WORD uDivisor;
  if( ser_UARTType( iSerPort ) != NOSER )
  {
    /* Calculate baud rate divisor */
    uDivisor = ( WORD )( SER_MAXBAUD / lBaudRate );
    outp( iSerPort + SER_LINE_CONTROL, /* Enable divisor access */
          inp( SER_LINE_CONTROL ) | SER_LCR_SETDIVISOR );
    /* Set baud rate divisor */
    outp( iSerPort + SER_DIVISOR_LSB, LOBYTE( uDivisor ) );
    outp( iSerPort + SER_DIVISOR_MSB, HIBYTE( uDivisor ) );
    /* Disable divisor access */
    outp( iSerPort + SER_LINE_CONTROL,
          inp( SER_LINE_CONTROL ) & ~SER_LCR_SETDIVISOR );

    /* Set other parameters only after resetting baud rate latch, */
    /* because this operation clears all */
    /* port parameters! */
    /* Set transmission parameters other than baud rate */
    outp( iSerPort + SER_LINE_CONTROL, bParams );
    /* Read a byte, to reverse possible error */
    inp( iSerPort + SER_TXBUFFER );
    return TRUE;
  }
  return FALSE;
}

/*****
/* ser_FIFOLevel : Set FIFO buffer size */
/**-----*/
/* Input : 0 - FIFO buffer size = 0, disable and reset (1 byte) */
/*          SER_FIFO_TRIGGER4/8/14 - size = 4, 8 or 14 bytes */
/*****

```

```

VOID ser_FIFOLevel( INT iSerPort, BYTE bLevel )
{
    if( bLevel ) outp( iSerPort + SER_FIFO, bLevel | SER_FIFO_ENABLE );
    else         outp( iSerPort + SER_FIFO, SER_FIFO_RESETPRECEIVE |
                      SER_FIFO_RESETPRETRANSMIT );
}

/*****
/* ser_IsDataAvailable : Is data available to be read?                */
/*-----**
/* Input : iSerPort - base port of interface being checked.          */
/* Output : == 0 : No byte available to be read                      */
/*          != 0 : Byte is available                                  */
/*-----**
/* Info : A byte is sent bit by bit, and becomes                    */
/*        a complete byte again only when the receiving port        */
/*        has combined the individual bits. This is what is         */
/*        being checked by this function.                            */
*****/
INT ser_IsDataAvailable( INT iSerPort )
{
    return inp( iSerPort + SER_LINE_STATUS ) & SER_LSR_DATA RECEIVED;
}

/*****
/* ser_IsWritingPossible : Can port send next byte ?                */
/*-----**
/* Input : iSerPort - base port of interface being checked.          */
/* Output : == 0 : Byte cannot be sent.                              */
/*          != 0 : Port ready to send.                                */
/*-----**
/* Info : A serial port should not be used                          */
*****/

```

```

/*          to send a byte in the following cases:                                */
/*          1. A received byte has not yet been "retrieved"                      */
/*          by the port.                                                           */
/*          2. An old send request has not yet been completed.                   */
/*****
INT ser_IsWritingPossible( INT iSerPort )
{
    return ( inp( iSerPort + SER_LINE_STATUS ) & SER_LSR_TSREMPY );
}

/*****
/* ser_IsModemStatusSet : Check status of input lines                            */
/**-----*/
/* Input : iSerPort      - base port of interface.                             */
/*          bTestStatus - Bit pattern of lines being tested                     */
/*          (CTS, DSR, RI, CD)                                                  */
/*****
INT ser_IsModemStatusSet( INT iSerPort, BYTE bTestStatus )
{
    return ( ( BYTE )inp( iSerPort + SER_MODEM_STATUS ) & bTestStatus )
           == bTestStatus;
}

/*****
/* ser_SetModemControl : Set signal lines for communication with                */
/*          modem etc.                                                           */
/**-----*/
/* Input : iSerPort      - base port of interface.                             */
/*          bNewControl - New status of DTR, RTS etc. lines                     */
/*****
VOID ser_SetModemControl( INT iSerPort, BYTE bNewControl )
{

```

```

    outp( iSerPort + SER_MODEM_CONTROL, bNewControl );
}

/*****
/* ser_WriteByte : Send a byte */
/*-----*/
/* Input : iSerPort - base port of interface through which */
/*          a byte is being sent. */
/*          bData    - byte to be sent */
/*          uTimeout - number of passes through loop */
/*                  after which a timeout error occurs */
/*                  if the send was unsuccessful (if */
/*                  iTimeout = 0 the system waits "forever".) */
/*          bSigMask - bit mask of signal lines being tested */
/*                  (RTS, CTS, CD, RI) */
/*          bSigVals - signal line status after applying */
/*                  above mask. */
/* Output : == 0 - byte was sent */
/*          != 0 - error */
*****/
INT ser_WriteByte( INT iSerPort, BYTE bData, UINT uTimeout,
                  BYTE bSigMask, BYTE bSigVals )
{
    if( uTimeout ) /* Timeout loop */
    {
        while( !ser_IsWritingPossible( iSerPort ) && uTimeout ) uTimeout--;
        if( !uTimeout ) return SER_ERRTIMEOUT;
    }
    else while( !ser_IsWritingPossible( iSerPort ) ); /* Wait! */

    /* Test signal lines */
    if( ( ( BYTE ) inp( iSerPort + SER_MODEM_STATUS ) & bSigMask ) ==
        bSigVals )

```

```

{
    /* Pass byte being sent to port */
    outp( iSerPort + SER_TXBUFFER, bData );
    /* Return port error */
    return inp( iSerPort + SER_LINE_STATUS ) & SER_LSR_ERRORMSK;
}
else return SER_ERRSIGNALS;
}

/*****
/* ser_ReadByte : Receive byte */
/**-----*/
/* Input : iSerPort - base port of interface through which */
/*           a byte is being received. */
/*           pData - address of byte accepting */
/*           received byte. */
/*           uTimeOut - number of passes through loop */
/*           after which a timeout error occurs */
/*           if the send was unsuccessful. (If */
/*           iTimeOut = 0 the system waits "forever".) */
/*           bSigMask - bit mask of signal lines being tested */
/*           (RTS, CTS, CD, RI) */
/*           bSigVals - signal line status after applying */
/*           above mask. */
/* Output : == 0 - byte was sent */
/*           != 0 - error */
*****/
INT ser_ReadByte( INT iSerPort, PBYTE pData, UINT uTimeOut,
                  BYTE bSigMask, BYTE bSigVals )
{
    if( uTimeOut ) /* Timeout loop */
    {
        while( !ser_IsDataAvailable( iSerPort ) && uTimeOut ) uTimeOut--;
        if( !uTimeOut ) return SER_ERRTIMEOUT;
    }
}

```



```

}
else while( !ser_IsDataAvailable( iSerPort ) );           /* Wait! */

/* Test signal lines */
if( ( ( BYTE ) inp( iSerPort + SER_MODEM_STATUS ) & bSigMask ) ==
    bSigVals )
{
    /* Read byte received by port */
    *pData = ( BYTE )inp( iSerPort + SER_RXBUFFER );
    return inp( iSerPort + SER_LINE_STATUS ) & SER_LSR_ERRORMSK;
}
else return SER_ERRSIGNALS;
}

/*****
/* ser_WritePacket : Send data packet                               */
/*-----**/
/* Input : iSerPort - base port of interface through which          */
/*          data is being sent.                                       */
/*          pData    - address of data being sent                   */
/*          iLen     - >= 0 : Number of bytes being sent.           */
/*                  < 0 : Buffer size = strlen( pData )             */
/*          uTimeOut - number of passes through loop                 */
/*                  after which a timeout error occurs              */
/*                  if the send was unsuccessful. (If               */
/*                  iTimeOut = 0 the system waits "forever".)        */
/*          bSigMask - bit mask of signal lines being tested         */
/*                  (RTS, CTS, CD, RI)                               */
/*          bSigVals - signal line status after applying             */
/*                  above mask.                                       */
/*          Output : == 0 - byte was sent                             */
/*                  != 0 - error                                       */
*****/

```

```

INT ser_WritePacket( INT iSerPort, PBYTE pData, INT iLen,
                    UINT uTimeout, BYTE bSigMask, BYTE bSigVals )
{
    INT i, e;
    if( iLen < 0 ) { iLen = 0; while( pData[ iLen ] ) iLen++; }
    for( i = 0; i < iLen; i++ )
        if( ( e = ser_WriteByte( iSerPort,
                                pData[ i ],
                                uTimeout,
                                bSigMask,
                                bSigVals ) ) != 0 ) return e;

    return SER_SUCCESS;
}

/*****
/* ser_ReadPacket : Receive data packet
/*-----*/
/* Input : iSerPort - base port of interface through which
/*           data is being received.
/*           pData    - address of data being sent
/*           iLen     - Size of receive buffer
/*           uTimeout - number of passes through loop
/*                   after which a timeout error occurs
/*                   if the send was unsuccessful. (If
/*                   iTimeout = 0 the system waits "forever".)
/*           bSigMask - bit mask of signal lines being tested
/*                   (RTS, CTS, CD, RI)
/*           bSigVals - signal line status after applying
/*                   above mask.
/* Output : == 0 - byte was sent
/*           != 0 - error
*****/
INT ser_ReadPacket( INT iSerPort, PBYTE pData, INT iLen,

```

```

        UINT uTimeOut, BYTE bSigMask, BYTE bSigVals )
{
    INT i, e;
    for( i = 0; i < iLen; i++ )
        if( ( e = ser_ReadByte( iSerPort,
                                &pData[ i ],
                                uTimeOut,
                                bSigMask,
                                bSigVals ) ) != 0 ) return e;

    return SER_SUCCESS;
}

/*****
/* ser_CLRIRQ : Disable serial interrupt requests to
/*
/*          IRQ controller.
**-----*/
/* Input : iSerPort - base port of interface that can no longer
/*          issue IRQs to IRQ controller.
*****/
VOID ser_CLRIRQ( INT iSerPort )
{
    outp( iSerPort + SER_MODEM_CONTROL,
          inp( iSerPort + SER_MODEM_CONTROL ) & ~SER_MCR_IRQENABLED );
}

/*****
/* ser_SETIRQ : Enable serial interrupt requests to
/*
/*          IRQ controller.
**-----*/
/* Input : iSerPort - base port of interface that must
/*          issue IRQs to IRQ controller.
*****/
VOID ser_SETIRQ( INT iSerPort )

```

```

{
    outp( iSerPort + SER_MODEM_CONTROL,
          inp( iSerPort + SER_MODEM_CONTROL ) | SER_MCR_IRQENABLED );
}

/*****
/* ser_SetIRQHandler : Set interrupt handler */
**-----**/
/* Input : iSerPort - base port of serial interface */
/*          for which interrupt handler */
/*          is being set. */
/*          iSerIRQ - IRQ(!) line reserved for port. */
/*          lpHandler - address of interrupt handler. */
/*          bEnablers - conditions initiating */
/*                   an IRQ (see SER_IER... bits) */
/* Output : Address of old IRQ handler */
*****/
VOID ( _interrupt _FP *ser_SetIRQHandler( INT iSerPort,
                                           INT iSerIRQ,
                                           VOID ( _interrupt _FP *lpHandler ) (),
                                           BYTE bEnablers ) ) ()
{
    /* Set IRQ enablers */
    outp( iSerPort + SER_IRQ_ENABLE, bEnablers );
    ser_SETIRQ( iSerPort ); /* Issue IRQs to IRQ controller */
    /* Set handler (IRQ is "enabled" there) */
    return irq_SetHandler( iSerIRQ, lpHandler );
}

/*****
/* ser_RestoreIRQHandler : Restore old IRQ handler */
**-----**/
/* Input : iSerPort - base port of serial interface whose */

```

```

/*                      old interrupt handler is being                      */
/*                      restored.                                           */
/*          iSerIRQ    -  IRQ line reserved for port.                      */
/*          lpHandler  -  address of old interrupt handler.                 */
/*****
VOID ser_RestoreIRQHandler( INT iSerPort,
                          INT iSerIRQ,
                          VOID ( _interrupt _FP *lpHandler ) ( ) )
{
    ser_CLRIRQ( iSerPort );/* No more IRQs to IRQ controller */
                          /* Set handler and clear all "enablers" */
    ser_SetIRQHandler( iSerPort, iSerIRQ, lpHandler, 0 );
    irq_Disable( iSerIRQ );    /* Also disable IRQs by the controller */
}

/*****
/* ser_PrintError : Output error message                                  */
/**-----**/
/* Input : pWin - address of window where output should appear          */
/*          or ZERO for output to screen.                                */
/*          e - error                                                      */
/*****
VOID ser_PrintError( PWINDOW pWin, INT e )
{
    switch( e )
    {
        case SER_LSR_DATA RECEIVED:
            win_printf( pWin, "Old data!\n" );
            break;
        case SER_ERRTIMEOUT:
            win_printf( pWin, "Timeout error! " );
            break;
    }
}

```

```

case SER_ERRSIGNALS:
    win_printf( pWin, "Signal lines!");
break;
default:
    if( e & SER_LSR_OVERRUNERROR )
        win_printf( pWin, "Overrun error!\n" );
    if( e & SER_LSR_PARITYERROR )
        win_printf( pWin, "Parity error!\n" );
    if( e & SER_LSR_FRAMINGERROR )
        win_printf( pWin, "Framing error!\n" );
    if( e & SER_LSR_BREAKDETECT )
        win_printf( pWin, "Break detect!\n" );
    }
}

/*****
/* ser_PrintModemStatus : Display status of signal lines */
/**-----*/
/* Input : pWin - Address of window where output should appear */
/*          or ZERO for output to screen. */
/*          iSerPort - base port of interface whose */
/*          line statuses are being displayed. */
/*****
VOID ser_PrintModemStatus( PWINDOW pWin, INT iSerPort )
{
    BYTE b;
    b = ( BYTE )inp( iSerPort + SER_MODEM_STATUS );
    win_printf(pWin,"%s", b & SER_MSR_DCTS ? "DCTS :" : " CTS : " );
    win_printf(pWin,"%s\n", b & SER_MSR_CTS ? "[X]" : "[ ]" );
    win_printf(pWin,"%s", b & SER_MSR_DDSR ? "DDSR :" : " DSR : " );
    win_printf(pWin,"%s\n", b & SER_MSR_DSR ? "[X]" : "[ ]" );
    win_printf(pWin,"%s", b & SER_MSR_DRI ? "DRI :" : " RI : " );
    win_printf(pWin,"%s\n", b & SER_MSR_RI ? "[X]" : "[ ]" );

```

```

win_printf(pWin, "%s", b & SER_MSR_DCD ? "DCD   : " : "CD   : " );
win_printf(pWin, "%s\n", b & SER_MSR_CD ? "[X]" : "[ ]" );
}

/*****7*****/
/* ser_PrintLineStatus : Display port status */
/*-----*/
/* Input : pWin - Address of window where output should appear */
/*          or ZERO for output to screen. */
/*          iSerPort - base port of interface whose */
/*          internal statuses are being displayed. */
/*****/
VOID ser_PrintLineStatus( PWINDOW pWin, INT iSerPort )
{
    BYTE b;
    b = ( BYTE )inp( iSerPort + SER_MODEM_STATUS );
    win_printf(pWin, "Data received %s\n",
                b & SER_LSR_DATARECEIVED ? "[X]" : "[ ]" );
    win_printf(pWin, "Overrun error %s\n",
                b & SER_LSR_OVERRUNERROR ? "[X]" : "[ ]" );
    win_printf(pWin, "Parity error %s\n",
                b & SER_LSR_PARITYERROR ? "[X]" : "[ ]" );
    win_printf(pWin, "Framing error %s\n",
                b & SER_LSR_FRAMINGERROR ? "[X]" : "[ ]" );
    win_printf(pWin, "Break detect %s\n",
                b & SER_LSR_BREAKDETECT ? "[X]" : "[ ]" );
    win_printf(pWin, "THR empty %s\n",
                b & SER_LSR_THREMPY ? "[X]" : "[ ]" );
    win_printf(pWin, "TSR empty %s\n",
                b & SER_LSR_TSREMPY ? "[X]" : "[ ]" );
}

/*****/

```

```

/* ser_GetBaud : Get current baud rate for port */
/**-----**/
/* Input : iSerPort - Base address of port whose */
/*                baud rate is being determined. */
/* Output : baud rate */
/******/
LONG ser_GetBaud( INT iSerPort )
{
    UINT uDivisor;
    BYTE bSettings;

    _disable();
    bSettings = ( BYTE ) inp( iSerPort + SER_LINE_CONTROL );
    outp( iSerPort + SER_LINE_CONTROL, bSettings | SER_LCR_SETDIVISOR );
    /* Read baud rate divisor */
    uDivisor = MAKEWORD( inp( iSerPort + SER_DIVISOR_MSB ),
                        inp( iSerPort + SER_DIVISOR_LSB ) );

    outp( iSerPort + SER_LINE_CONTROL, bSettings );
    _enable();
    if( uDivisor ) return SER_MAXBAUD / uDivisor;
    return 0L;
}

/******/
/* ser_PrintCardSettings : Display transmission parameters */
/*                of port */
/**-----**/
/* Input : pWin - Address of window where output should appear */
/*                or ZERO for output to screen. */
/*                iSerPort - base port of interface whose */
/*                transmission parameters are being displayed. */
/******/

```



```

VOID ser_PrintCardSettings( PWINDOW pWin, INT iSerPort )
{
    LONG lBaudRate;
    BYTE bSettings;
    INT WordLen;

    switch( ser_UARTType( iSerPort ) )
    {
        case NOSER:
            win_printf(pWin, " No port detected!\n");
            return;
        case INS8250:
            win_printf(pWin, " INS8250 UART chip\n");
            break;
        case NS16450:
            win_printf(pWin, " NS16450 UART chip\n");
            break;
        case NS16550A:
            win_printf(pWin, " NS16550A UART chip\n");
            break;
        case NS16C552:
            win_printf(pWin, " NS16C552 UART chip\n");
            break;
    }
    lBaudRate = ser_GetBaud( iSerPort );
    win_printf(pWin, " Baud rate %ld\n", lBaudRate );

    bSettings = ( BYTE ) inp( iSerPort + SER_LINE_CONTROL );
    WordLen = 5 + ( bSettings & SER_LCR_WORDLEN );
    win_printf(pWin, " Data bits : %d\n", WordLen );
    win_printf(pWin, " Stop bits : %s\n",
                bSettings & SER_LCR_2STOPBITS ?

```

```

                ( WordLen == 5 ? "1.5" : "2" ) : "1");
win_printf(pWin, " Parity      : ");
switch( bSettings & SER_LCR_PARITYMSK )
{
    case SER_LCR_ODDPARITY:  win_printf(pWin, "odd");
    case SER_LCR_EVENPARITY: win_printf(pWin, "even"); break;
    case SER_LCR_PARITYSET:  win_printf(pWin, "always set"); break;
    case SER_LCR_PARITYCLR:  win_printf(pWin, "always cleared"); break;
    default: win_printf(pWin, "none");
}
/* PARITYON = 0 => No parity */

win_printf(pWin, "\n");
if( bSettings & SER_LCR_SENDBREAK )
    win_printf(pWin, "Send break signal\n");
}
#endif

```

