

```

/*****
/* Program      : SERIAL.C
/* Function     : Serial Communications Utility Procedures.
/* Author      : John F. Fitter B.E.
/* Platform    : HiTech PIC C v7.72 pl1
/* Target      : PIC16C67
/*
/* DTR/DSR handshaking is used because this is what the POS printer requires. Comms with
/* a pc can then be accomplished using the same cable as used for the printer.
/* Ensure that the pc program is written to use the same handshaking. The GreenLeaf Comm
/* library will allow this.
/*
/* Rev No.   Rev date   Test date   Test platform   Description
/* -----   -
/*      00    6jun98      -----      PIC16C77-ME     Original in HiTech C
/*
/* Copyright © 1998 Eagle Air Australia Pty. Ltd. All rights reserved
/*****

#define _SERIAL_C
#include <stdlib.h>
#include <stdio.h>
#include <commdefs.h>
#include "main.h"
#include "1306spi.h"
#include "25cxxspi.h"
#include "serial.h"
#include "delays.h"
#include "e2data.h"
#include "lcd44780.h"

/*****
// Serial port procedures.
/*****

// Serial port initialisation.

void init_serial() {

    dtr = B_LOW;                // setup the port directions
    dtr_dir = B_OUT;           // and handshake levels
    dsr_dir = B_IN;

    c_status.char_is_in = false; // no chars are waiting
    c_status.ser_rx_err = false; // and no errors

    sertimeout = DEF_SER_TIMEOUT; // set the default serial timeout
    setup_usart_async8_hi(DEF_BAUD); // set async comms
    enable_interrupts(INT_RDA); // and interrupt on receive
}

// Send one character to the serial port by polling for an opportunity. The transmit register
// is polled until it is clear then DSR is polled until the device is ready to receive.
// This procedure will clear TXIF automatically. The timeout timer is driven from the 1mS
// interrupts which drive the backlight. If a timeout occurs or is already in effect then the
// serial timeout flag is set, the character is not transmitted and the procedure returns
// false. Main line code must reset the timeout flag. If the character is successfully
// transmitted then the procedure returns true.

unsigned char putch_ser(unsigned char data) {

    ser_time = sertimeout;
    while(!TXIF && ser_time); // wait until transmit register clear
    ser_time = sertimeout;
    while(!DSR && ser_time); // wait until data set ready
    if(ser_time) TXREG = data; // send the data if not timed out
    return !!ser_time; // return true for success
}

// Send one integer in little endian format to the serial port by polling for an opportunity.
// Returns true on successfully transmitting the integer.

unsigned char putint_ser(unsigned int data) {

    if(putch_ser(LOBYTE(data))) // send the low byte
        return putch_ser(HIBYTE(data)); // send the high byte
    return false; // failed to send word
}

```

```

}

// Copies a received character from the serial receive buffer to the passed address. If a
// reception error has occurred the global serial receive error flag will be set by the ISR
// and the error will be cleared. This procedure will return the error status. The received
// character flag will be reset and DTR will be asserted to enable reception of more characters.
// If no character is waiting in the receive buffer then the procedure performs no function.
// The procedure returns true on successful reception of a character and false on error or if
// there is no character waiting. The procedure must copy the char before enabling reception
// of more chars otherwise a receive interrupt may occur and corrupt the received data.

unsigned char getch_ser(unsigned char *serial_char) {
    if(c_status.char_is_in) { // is there a character waiting ?
        c_status.char_is_in = false; // reset receive flag
        *serial_char = ser_char; // save the received char
        dtr = B_LOW; // enable reception of more characters
        if(!c_status.ser_rx_err) return true; // return true on success
    }
    return false; // return error or no char waiting
}

// Returns one byte of data from the serial port to the passed address. If there is no data
// in the input buffer then the procedure will wait for some up to the serial timeout period.
// The procedure returns true on successful reception and false on timeout or other errors.

unsigned char get_ser_byte(unsigned char* sbyte) {
    ser_time = sertimeout; // preset the serial timeout
    do if(!ser_time) return false; // wait for data or a timeout
    while(!getch_ser(sbyte)); // get the data if any
    return true; // return success
}

// Returns one integer from reception of two bytes sent in little endian format. The
// procedure returns true on successful reception and false on timeout or other errors.

unsigned char get_ser_int(unsigned int* sint) {
    unsigned char data;

    if(get_ser_byte(&data)) { // get the ls byte
        *sint = data; // and stash it away
        if(get_ser_byte(&data)) { // if no errors, then get the ms byte
            *sint |= (unsigned int)data << 8; // and do the same
            return true; // return success
        }
    }
    return false; // failed for some reason
}

// Procedure to reset the serial interface. 50 nulls without handshaking are sent. If a
// printer is attached this will wake it. If a pc is attached it will ignore the nulls.

void reset_ser() {
    unsigned char n = 50;
    while(n--) { // send the 50 nulls - ignore hshk
        ser_time = sertimeout; // set the serial timeout timer
        while(!TXIF && ser_time); // wait until transmit register clear
        TXREG = 0; // transmit the null
    }
}

/*****
/* Procedure to process serial data received from the PC */
/*
/* The instruction set from the PC which the controller responds to is as follows;
/*
/* ETX Controller will download contents of eeprom memory from PC
/* EM Controller will upload contents of eeprom memory to PC
/* ETB Controller will download contents of nvram from PC
/* SYN Controller will upload contents of nvram to PC
/* NAK Controller will reset
/* SUB Controller will return to factory defaults and then reset
/* 0xE1..0xE6 Simulate keypress from keys 1..6
/*
/* The communications protocol is as follows;

```

```

/*                                                                                               */
/* byte 0          command          8 bits                                                       */
/* bytes 1..2      start address    16 bits, little endian                                       */
/* bytes 3..4      n                 16 bits, little endian                                       */
/* bytes 5..n+5    n data bytes     8 bits                                                       */
/* bytes n+6, n+7 16 bit CRC       16 bits, little endian                                       */
/*                                                                                               */
/* Rev No.   Rev date   Test date   Test platform   Description                                     */
/* -----   - - - - -   - - - - -   - - - - -   - - - - -                                     */
/*    00     10aug98    Rice16, PCB1.0   Original                                           */
/*    01     21jan99    Rice16, PCB1.0   Added response to STX                               */
/*    02     25jan99    Rice16, PCB1.0   Added CRC                                           */
/*****
// Generalized serial received data processing. Returns true on success.
unsigned char process_rx_data() {

    unsigned tx = false, e2 = false;
    unsigned int addr, nbytes, crc;

    if(getch_ser(&ser_data)) { // get a command, are there errors ?
        if(!c_status.awake) goto_sleep(false); // wakeup on serial activity
        switch(ser_data) { // select the command

            case A_SUB : reset_proc(true); // set factory defaults and reset proc
            case A_NAK : reset_proc(false); // simple reset of processor
            case A_EM  : e2 = true; // eeprom or nvram ?
            case A_SYN : tx = true; // transmitting or receiving ?
            case A_ETX : if(ser_data == A_ETX) e2 = true; // play with the logic a little bit !!
            case A_ETB :

                // Get destination/source address and number of bytes.
                if(get_ser_int(&addr) && get_ser_int(&nbytes)) {

                    // Initialize the CRC to zero.
                    crc = 0;

                    // Loop for all of the data - nbytes is the number of bytes.
                    while(nbytes--){

                        // If we are transmitting data - tx is the flag.
                        if(tx) {

                            // Get the data from the eeprom or nvram - e2 is the flag.
                            if(e2) ser_data = read_eeprom_deselect(addr);
                            else ser_data = read_1306(addr);

                            // Send the data to the serial port and bomb out on failure.
                            if(!putch_ser(ser_data)) return false;

                            // We must be receiving data.
                        }else {

                            // Get the data from the serial port and route it to the eeprom
                            // or nvram - e2 is the flag. Bomb out on failure.
                            if(get_ser_byte(&ser_data)) {
                                if(e2) write_eeprom(addr, ser_data);
                                else write_1306(addr, ser_data);
                            }else return false;
                        }

                        // Update the CRC for transmitted and received data using
                        // the CCITT 16bit algorithm (X^16 + X^12 + X^5 + 1).
                        crc = (unsigned char)(crc >> 8) | (crc << 8);
                        crc ^= ser_data;
                        crc ^= (unsigned char)(crc & 0xff) >> 4;
                        crc ^= (crc << 8) << 4;
                        crc ^= ((crc & 0xff) << 4) << 1;
                        ++addr;
                    }

                    // Send the CRC at the end of a reception or a transmission.
                    return putint_ser(crc);
                }
            }
        }
    }
    break;
    default : if(ser_data > 0xe0) c_status.ext_key = true;
              else return false;
}
return true;

```

```
    }  
    return false;  
}
```

```
// ***** EOF SERIAL.C *****
```