```c
/* SERIAL.C: This code is designed to act as a low-level serial driver for
    higher-level programming.  Ideally, one could simply call init_serial()
    to initialize the serial port, then use serial_send("data", 4) to send
    an array of data (8-bit unsigned character strings).

    WRITTEN BY : Paul Miller <pamiller@uiuc.edu>
    $Id: serial.c,v 1.4 2003/08/08 16:08:56 paul Exp $
 */
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <math.h>
#include <limits.h>

#include "F28x_Project.h"     // Device Headerfile and Examples Include File
#include <buffer.h>
#include <F28379dSerial.h>
#include <F2837xD_sci.h>

serialSCIA_t SerialA;
serialSCIB_t SerialB;
serialSCIC_t SerialC;
serialSCID_t SerialD;

char RXAdata = 0;
char RXBdata = 0;
char RXCdata = 0;
char RXDdata = 0;
uint32_t numRXA = 0;
uint32_t numRXB = 0;
uint32_t numRXC = 0;
uint32_t numRXD = 0;


// for SerialA
uint16_t init_serialSCIA(serialSCIA_t *s, uint32_t baud)
{
    volatile struct SCI_REGS *sci;
    uint32_t clk;

    if (s == &SerialA) {
        sci = &SciaRegs;
        s->sci = sci;
        init_bufferSCIA(&s->TX);

        GPIO_SetupPinMux(43, GPIO_MUX_CPU1, 15);
        GPIO_SetupPinOptions(43, GPIO_INPUT, GPIO_PULLUP);
        GPIO_SetupPinMux(42, GPIO_MUX_CPU1, 15);
        GPIO_SetupPinOptions(42, GPIO_OUTPUT, GPIO_PUSHPULL);

    } else {
        return 1;
    }

    /* init for standard baud,8N1 comm */
    sci->SCICTL1.bit.SWRESET = 0;       // init SCI state machines and opt flags
    sci->SCICCR.all = 0x0;
    sci->SCICTL1.all = 0x0;
    sci->SCICTL2.all = 0x0;
    sci->SCIPRI.all = 0x0;
    clk = LSPCLK_HZ;                    // set baud rate
    clk /= baud*8;
    clk--;
    sci->SCILBAUD.all = clk & 0xFF;
    sci->SCIHBAUD.all = (clk >> 8) & 0xFF;

    sci->SCICCR.bit.SCICHAR = 0x7;      // (8) 8 bits per character
    sci->SCICCR.bit.PARITYENA = 0;      // (N) disable party calculation
    sci->SCICCR.bit.STOPBITS = 0;       // (1) transmit 1 stop bit
    sci->SCICCR.bit.LOOPBKENA = 0;      // disable loopback test
    sci->SCICCR.bit.ADDRIDLE_MODE = 0;  // idle-line mode (non-multiprocessor SCI comm)

    sci->SCIFFCT.bit.FFTXDLY = 0;       // TX: zero-delay

    sci->SCIFFTX.bit.SCIFFENA = 1;      // enable SCI fifo enhancements
    sci->SCIFFTX.bit.TXFIFORESET = 0;
    sci->SCIFFTX.bit.TXFFIL = 0x0;// TX: fifo interrupt at all levels   ???? is this correct
    sci->SCIFFTX.bit.TXFFINTCLR = 1;  // TX: clear interrupt flag
    sci->SCIFFTX.bit.TXFFIENA = 0;      // TX: disable fifo interrupt
    sci->SCIFFTX.bit.TXFIFORESET = 1;

    sci->SCIFFRX.bit.RXFIFORESET = 0;   // RX: fifo reset
    sci->SCIFFRX.bit.RXFFINTCLR = 1;    // RX: clear interrupt flag
    sci->SCIFFRX.bit.RXFFIENA = 1;      // RX: enable fifo interrupt
    sci->SCIFFRX.bit.RXFFIL = 0x1;      // RX: fifo interrupt
    sci->SCIFFRX.bit.RXFIFORESET = 1;   // RX: re-enable fifo

    sci->SCICTL2.bit.RXBKINTENA = 0;    // disable receiver/error interrupt
    sci->SCICTL2.bit.TXINTENA = 0;      // disable transmitter interrupt

    sci->SCICTL1.bit.TXWAKE = 0;
    sci->SCICTL1.bit.SLEEP = 0;         // disable sleep mode
    sci->SCICTL1.bit.RXENA = 1;         // enable SCI receiver
    sci->SCICTL1.bit.RXERRINTENA = 0;   // disable receive error interrupt
    sci->SCICTL1.bit.TXENA = 1;         // enable SCI transmitter
    sci->SCICTL1.bit.SWRESET = 1;       // re-enable SCI

    /* enable PIE interrupts */
    if (s == &SerialA) {
        PieCtrlRegs.PIEIER9.bit.INTx1 = 1;
        PieCtrlRegs.PIEIER9.bit.INTx2 = 1;
        IER |= (M_INT9);
        PieCtrlRegs.PIEACK.all = (PIEACK_GROUP9);

    }

    return 0;
}

void uninit_serialSCIA(serialSCIA_t *s)
{
    volatile struct SCI_REGS *sci = s->sci;

    /* disable PIE interrupts */
    if (s == &SerialA) {
        PieCtrlRegs.PIEIER9.bit.INTx1 = 0;
        PieCtrlRegs.PIEIER9.bit.INTx2 = 0;
        IER &= ~M_INT9;
    }
    sci->SCICTL1.bit.RXERRINTENA = 0;   // disable receive error interrupt
    sci->SCICTL2.bit.RXBKINTENA = 0;    // disable receiver/error interrupt
    sci->SCICTL2.bit.TXINTENA = 0;      // disable transmitter interrupt

    sci->SCICTL1.bit.RXENA = 0;         // disable SCI receiver
    sci->SCICTL1.bit.TXENA = 0;         // disable SCI transmitter
}



/***************************************************************************
 * SERIAL_SEND()
 *
 * "User level" function to send data via serial.  Return value is the
 * length of data successfully copied to the TX buffer.
 ***************************************************************************/
uint16_t serial_sendSCIA(serialSCIA_t *s, char *data, uint16_t len)
{
    uint16_t i = 0;
    if (len && s->TX.size < BUF_SIZESCIA) {
        for (i = 0; i < len; i++) {
            if (buf_writeSCIA_1(&s->TX, data[i] & 0x00FF) != 0) break;
        }
        s->sci->SCIFFTX.bit.TXFFINTCLR = 1;  // TX: clear interrupt flag
        s->sci->SCIFFTX.bit.TXFFIENA = 1;       // TX: enable fifo interrupt
    }
    return i;
}

// For SerialB
uint16_t init_serialSCIB(serialSCIB_t *s, uint32_t baud)
{
    volatile struct SCI_REGS *sci;
    uint32_t clk;

    if (s == &SerialB) {
        sci = &ScibRegs;
        s->sci = sci;
        init_bufferSCIB(&s->TX);

        GPIO_SetupPinMux(15, GPIO_MUX_CPU1, 2);
        GPIO_SetupPinOptions(15, GPIO_INPUT, GPIO_PULLUP);
        GPIO_SetupPinMux(14, GPIO_MUX_CPU1, 2);
        GPIO_SetupPinOptions(14, GPIO_OUTPUT, GPIO_PUSHPULL);


    } else {
        return 1;
    }

    /* init for standard baud,8N1 comm */
    sci->SCICTL1.bit.SWRESET = 0;       // init SCI state machines and opt flags
    sci->SCICCR.all = 0x0;
    sci->SCICTL1.all = 0x0;
    sci->SCICTL2.all = 0x0;
    sci->SCIPRI.all = 0x0;
    clk = LSPCLK_HZ;                    // set baud rate
    clk /= baud*8;
    clk--;
    sci->SCILBAUD.all = clk & 0xFF;
    sci->SCIHBAUD.all = (clk >> 8) & 0xFF;

    sci->SCICCR.bit.SCICHAR = 0x7;      // (8) 8 bits per character
    sci->SCICCR.bit.PARITYENA = 0;      // (N) disable party calculation
    sci->SCICCR.bit.STOPBITS = 0;       // (1) transmit 1 stop bit
    sci->SCICCR.bit.LOOPBKENA = 0;      // disable loopback test
    sci->SCICCR.bit.ADDRIDLE_MODE = 0;  // idle-line mode (non-multiprocessor SCI comm)

    sci->SCIFFCT.bit.FFTXDLY = 0;       // TX: zero-delay

    sci->SCIFFTX.bit.SCIFFENA = 1;      // enable SCI fifo enhancements
    sci->SCIFFTX.bit.TXFIFORESET = 0;
    sci->SCIFFTX.bit.TXFFIL = 0x0;// TX: fifo interrupt at all levels   ???? is this correct
    sci->SCIFFTX.bit.TXFFINTCLR = 1;  // TX: clear interrupt flag
    sci->SCIFFTX.bit.TXFFIENA = 0;      // TX: disable fifo interrupt
    sci->SCIFFTX.bit.TXFIFORESET = 1;

    sci->SCIFFRX.bit.RXFIFORESET = 0;   // RX: fifo reset
    sci->SCIFFRX.bit.RXFFINTCLR = 1;    // RX: clear interrupt flag
    sci->SCIFFRX.bit.RXFFIENA = 1;      // RX: enable fifo interrupt
    sci->SCIFFRX.bit.RXFFIL = 0x1;      // RX: fifo interrupt
    sci->SCIFFRX.bit.RXFIFORESET = 1;   // RX: re-enable fifo

    sci->SCICTL2.bit.RXBKINTENA = 0;    // disable receiver/error interrupt
    sci->SCICTL2.bit.TXINTENA = 0;      // disable transmitter interrupt

    sci->SCICTL1.bit.TXWAKE = 0;
    sci->SCICTL1.bit.SLEEP = 0;         // disable sleep mode
    sci->SCICTL1.bit.RXENA = 1;         // enable SCI receiver
    sci->SCICTL1.bit.RXERRINTENA = 0;   // disable receive error interrupt
    sci->SCICTL1.bit.TXENA = 1;         // enable SCI transmitter
    sci->SCICTL1.bit.SWRESET = 1;       // re-enable SCI

    /* enable PIE interrupts */
    if (s == &SerialB) {
        PieCtrlRegs.PIEIER9.bit.INTx3 = 1;
        PieCtrlRegs.PIEIER9.bit.INTx4 = 1;
        IER |= (M_INT9);
        PieCtrlRegs.PIEACK.all = (PIEACK_GROUP9);
    }

    return 0;
}

void uninit_serialSCIB(serialSCIB_t *s)
{
    volatile struct SCI_REGS *sci = s->sci;

    /* disable PIE interrupts */
    if (s == &SerialB) {
        PieCtrlRegs.PIEIER9.bit.INTx3 = 0;
        PieCtrlRegs.PIEIER9.bit.INTx4 = 0;
        IER &= ~M_INT9;
    }

    sci->SCICTL1.bit.RXERRINTENA = 0;   // disable receive error interrupt
    sci->SCICTL2.bit.RXBKINTENA = 0;    // disable receiver/error interrupt
    sci->SCICTL2.bit.TXINTENA = 0;      // disable transmitter interrupt

    sci->SCICTL1.bit.RXENA = 0;         // disable SCI receiver
    sci->SCICTL1.bit.TXENA = 0;         // disable SCI transmitter
}



/***************************************************************************
 * SERIAL_SEND()
 *
 * "User level" function to send data via serial.  Return value is the
 * length of data successfully copied to the TX buffer.
 ***************************************************************************/
uint16_t serial_sendSCIB(serialSCIB_t *s, char *data, uint16_t len)
{
    uint16_t i = 0;
    if (len && s->TX.size < BUF_SIZESCIB) {
        for (i = 0; i < len; i++) {
            if (buf_writeSCIB_1(&s->TX, data[i] & 0x00FF) != 0) break;
        }
        s->sci->SCIFFTX.bit.TXFFINTCLR = 1;  // TX: clear interrupt flag
        s->sci->SCIFFTX.bit.TXFFIENA = 1;       // TX: enable fifo interrupt
    }
    return i;
}

// for SerialC
uint16_t init_serialSCIC(serialSCIC_t *s, uint32_t baud)
{
    volatile struct SCI_REGS *sci;
    uint32_t clk;

    if (s == &SerialC) {
        sci = &ScicRegs;
        s->sci = sci;
        init_bufferSCIC(&s->TX);
        GPIO_SetupPinMux(139, GPIO_MUX_CPU1, 6);
        GPIO_SetupPinOptions(139, GPIO_INPUT, GPIO_PULLUP);
        GPIO_SetupPinMux(56, GPIO_MUX_CPU1, 6);
        GPIO_SetupPinOptions(56, GPIO_OUTPUT, GPIO_PUSHPULL);

    } else {
        return 1;
    }

    /* init for standard baud,8N1 comm */
    sci->SCICTL1.bit.SWRESET = 0;       // init SCI state machines and opt flags
```

```c
    sci->SCICCR.all = 0x0;
    sci->SCICTL1.all = 0x0;
    sci->SCICTL2.all = 0x0;
    sci->SCIPRI.all = 0x0;
    clk = LSPCLK_HZ;                    // set baud rate
    clk /= baud*8;
    clk--;
    sci->SCILBAUD.all = clk & 0xFF;
    sci->SCIHBAUD.all = (clk >> 8) & 0xFF;

    sci->SCICCR.bit.SCICHAR = 0x7;      // (8) 8 bits per character
    sci->SCICCR.bit.PARITYENA = 0;      // (N) disable party calculation
    sci->SCICCR.bit.STOPBITS = 0;       // (1) transmit 1 stop bit
    sci->SCICCR.bit.LOOPBKENA = 0;      // disable loopback test
    sci->SCICCR.bit.ADDRIDLE_MODE = 0;  // idle-line mode (non-multiprocessor SCI comm)

    sci->SCIFFCT.bit.FFTXDLY = 0;       // TX: zero-delay

    sci->SCIFFTX.bit.SCIFFENA = 1;      // enable SCI fifo enhancements
    sci->SCIFFTX.bit.TXFIFORESET = 0;
    sci->SCIFFTX.bit.TXFFIL = 0x0;// TX: fifo interrupt at all levels   ???? is this correct
    sci->SCIFFTX.bit.TXFFINTCLR = 1;  // TX: clear interrupt flag
    sci->SCIFFTX.bit.TXFFIENA = 0;      // TX: disable fifo interrupt
    sci->SCIFFTX.bit.TXFIFORESET = 1;

    sci->SCIFFRX.bit.RXFIFORESET = 0;   // RX: fifo reset
    sci->SCIFFRX.bit.RXFFINTCLR = 1;    // RX: clear interrupt flag
    sci->SCIFFRX.bit.RXFFIENA = 1;      // RX: enable fifo interrupt
    sci->SCIFFRX.bit.RXFFIL = 0x1;      // RX: fifo interrupt
    sci->SCIFFRX.bit.RXFIFORESET = 1;   // RX: re-enable fifo

    sci->SCICTL2.bit.RXBKINTENA = 0;    // disable receiver/error interrupt
    sci->SCICTL2.bit.TXINTENA = 0;      // disable transmitter interrupt

    sci->SCICTL1.bit.TXWAKE = 0;
    sci->SCICTL1.bit.SLEEP = 0;         // disable sleep mode
    sci->SCICTL1.bit.RXENA = 1;         // enable SCI receiver
    sci->SCICTL1.bit.RXERRINTENA = 0;   // disable receive error interrupt
    sci->SCICTL1.bit.TXENA = 1;         // enable SCI transmitter
    sci->SCICTL1.bit.SWRESET = 1;       // re-enable SCI

    /* enable PIE interrupts */
    if (s == &SerialC) {
        PieCtrlRegs.PIEIER8.bit.INTx5 = 1;
        PieCtrlRegs.PIEIER8.bit.INTx6 = 1;
        PieCtrlRegs.PIEACK.all = (PIEACK_GROUP8);
        IER |= (M_INT8);

    }

    return 0;
}

void uninit_serialSCIC(serialSCIC_t *s)
{
    volatile struct SCI_REGS *sci = s->sci;

    /* disable PIE interrupts */
    if (s == &SerialC) {
        PieCtrlRegs.PIEIER8.bit.INTx5 = 0;
        PieCtrlRegs.PIEIER8.bit.INTx6 = 0;
        IER &= ~M_INT8;
    }

    sci->SCICTL1.bit.RXERRINTENA = 0;   // disable receive error interrupt
    sci->SCICTL2.bit.RXBKINTENA = 0;    // disable receiver/error interrupt
    sci->SCICTL2.bit.TXINTENA = 0;      // disable transmitter interrupt

    sci->SCICTL1.bit.RXENA = 0;         // disable SCI receiver
    sci->SCICTL1.bit.TXENA = 0;         // disable SCI transmitter
}



/***************************************************************************
 * SERIAL_SEND()
 *
 * "User level" function to send data via serial.  Return value is the
 * length of data successfully copied to the TX buffer.
 ***************************************************************************/
uint16_t serial_sendSCIC(serialSCIC_t *s, char *data, uint16_t len)
{
    uint16_t i = 0;
    if (len && s->TX.size < BUF_SIZESCIC) {
        for (i = 0; i < len; i++) {
            if (buf_writeSCIC_1(&s->TX, data[i] & 0x00FF) != 0) break;
        }
        s->sci->SCIFFTX.bit.TXFFINTCLR = 1;  // TX: clear interrupt flag
        s->sci->SCIFFTX.bit.TXFFIENA = 1;       // TX: enable fifo interrupt
    }
    return i;
}

// For Serial D
uint16_t init_serialSCID(serialSCID_t *s, uint32_t baud)
{
    volatile struct SCI_REGS *sci;
    uint32_t clk;

    if (s == &SerialD) {
        sci = &ScidRegs;
        s->sci = sci;
        init_bufferSCID(&s->TX);
        GPIO_SetupPinMux(105, GPIO_MUX_CPU1, 6);
        GPIO_SetupPinOptions(105, GPIO_INPUT, GPIO_PULLUP);
        GPIO_SetupPinMux(104, GPIO_MUX_CPU1, 6);
        GPIO_SetupPinOptions(104, GPIO_OUTPUT, GPIO_PUSHPULL);
    }
    else {
        return 1;
    }

    /* init for standard baud,8N1 comm */
    sci->SCICTL1.bit.SWRESET = 0;       // init SCI state machines and opt flags
    sci->SCICCR.all = 0x0;
    sci->SCICTL1.all = 0x0;
    sci->SCICTL2.all = 0x0;
    sci->SCIPRI.all = 0x0;
    clk = LSPCLK_HZ;                    // set baud rate
    clk /= baud*8;
    clk--;
    sci->SCILBAUD.all = clk & 0xFF;
    sci->SCIHBAUD.all = (clk >> 8) & 0xFF;

    sci->SCICCR.bit.SCICHAR = 0x7;      // (8) 8 bits per character
    sci->SCICCR.bit.PARITYENA = 0;      // (N) disable party calculation
    sci->SCICCR.bit.STOPBITS = 0;       // (1) transmit 1 stop bit
    sci->SCICCR.bit.LOOPBKENA = 0;      // disable loopback test
    sci->SCICCR.bit.ADDRIDLE_MODE = 0;  // idle-line mode (non-multiprocessor SCI comm)

    sci->SCIFFCT.bit.FFTXDLY = 0;       // TX: zero-delay

    sci->SCIFFTX.bit.SCIFFENA = 1;      // enable SCI fifo enhancements
    sci->SCIFFTX.bit.TXFIFORESET = 0;
    sci->SCIFFTX.bit.TXFFIL = 0x0;// TX: fifo interrupt at all levels   ???? is this correct
    sci->SCIFFTX.bit.TXFFINTCLR = 1;  // TX: clear interrupt flag
    sci->SCIFFTX.bit.TXFFIENA = 0;      // TX: disable fifo interrupt
    sci->SCIFFTX.bit.TXFIFORESET = 1;

    sci->SCIFFRX.bit.RXFIFORESET = 0;   // RX: fifo reset
    sci->SCIFFRX.bit.RXFFINTCLR = 1;    // RX: clear interrupt flag
    sci->SCIFFRX.bit.RXFFIENA = 1;      // RX: enable fifo interrupt
    sci->SCIFFRX.bit.RXFFIL = 0x1;      // RX: fifo interrupt
    sci->SCIFFRX.bit.RXFIFORESET = 1;   // RX: re-enable fifo

    sci->SCICTL2.bit.RXBKINTENA = 0;    // disable receiver/error interrupt
    sci->SCICTL2.bit.TXINTENA = 0;      // disable transmitter interrupt

    sci->SCICTL1.bit.TXWAKE = 0;
    sci->SCICTL1.bit.SLEEP = 0;         // disable sleep mode
    sci->SCICTL1.bit.RXENA = 1;         // enable SCI receiver
    sci->SCICTL1.bit.RXERRINTENA = 0;   // disable receive error interrupt
    sci->SCICTL1.bit.TXENA = 1;         // enable SCI transmitter
    sci->SCICTL1.bit.SWRESET = 1;       // re-enable SCI

    /* enable PIE interrupts */
    if (s == &SerialD) {
        PieCtrlRegs.PIEIER8.bit.INTx7 = 1;
        PieCtrlRegs.PIEIER8.bit.INTx8 = 1;
        PieCtrlRegs.PIEACK.all = (PIEACK_GROUP8);
        IER |= (M_INT8);
    }

    return 0;
}

void uninit_serialSCID(serialSCID_t *s)
{
    volatile struct SCI_REGS *sci = s->sci;

    /* disable PIE interrupts */
    if (s == &SerialD) {
        PieCtrlRegs.PIEIER8.bit.INTx7 = 0;
        PieCtrlRegs.PIEIER8.bit.INTx8 = 0;
        IER &= ~M_INT8;
    }

    sci->SCICTL1.bit.RXERRINTENA = 0;   // disable receive error interrupt
    sci->SCICTL2.bit.RXBKINTENA = 0;    // disable receiver/error interrupt
    sci->SCICTL2.bit.TXINTENA = 0;      // disable transmitter interrupt

    sci->SCICTL1.bit.RXENA = 0;         // disable SCI receiver
    sci->SCICTL1.bit.TXENA = 0;         // disable SCI transmitter
}



/***************************************************************************
 * SERIAL_SEND()
 *
 * "User level" function to send data via serial.  Return value is the
 * length of data successfully copied to the TX buffer.
 ***************************************************************************/
uint16_t serial_sendSCID(serialSCID_t *s, char *data, uint16_t len)
{
    uint16_t i = 0;
    if (len && s->TX.size < BUF_SIZESCID) {
        for (i = 0; i < len; i++) {
            if (buf_writeSCID_1(&s->TX, data[i] & 0x00FF) != 0) break;
        }
        s->sci->SCIFFTX.bit.TXFFINTCLR = 1;  // TX: clear interrupt flag
        s->sci->SCIFFTX.bit.TXFFIENA = 1;       // TX: enable fifo interrupt
    }
    return i;
}



/***************************************************************************
 * TXxINT_DATA_SENT()
 *
 * Executed when transmission is ready for additional data.  These functions
 * read the next char of data and put it in the TXBUF register for transfer.
 ***************************************************************************/
#ifdef _FLASH
#pragma CODE_SECTION(TXAINT_data_sent, ".TI.ramfunc");
#endif
__interrupt void TXAINT_data_sent(void)
{
    char data;
    if (buf_readSCIA_1(&SerialA.TX,0,&data) == 0) {
        while ( (buf_readSCIA_1(&SerialA.TX,0,&data) == 0)
                && (SerialA.sci->SCIFFTX.bit.TXFFST != 0x10) ) {
            buf_removeSCIA(&SerialA.TX, 1);
            SerialA.sci->SCITXBUF.all = data;
        }
    } else {
        SerialA.sci->SCIFFTX.bit.TXFFIENA = 0;      // TX: disable fifo interrupt
    }
    SerialA.sci->SCIFFTX.bit.TXFFINTCLR = 1;  // TX: clear interrupt flag
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;
}


//for serialB
#ifdef _FLASH
#pragma CODE_SECTION(TXBINT_data_sent, ".TI.ramfunc");
#endif
__interrupt void TXBINT_data_sent(void)
{
    char data;
    if (buf_readSCIB_1(&SerialB.TX,0,&data) == 0) {
        while ( (buf_readSCIB_1(&SerialB.TX,0,&data) == 0)
                && (SerialB.sci->SCIFFTX.bit.TXFFST != 0x10) ) {
            buf_removeSCIB(&SerialB.TX, 1);
            SerialB.sci->SCITXBUF.all = data;
        }
    } else {
        SerialB.sci->SCIFFTX.bit.TXFFIENA = 0;      // TX: disable fifo interrupt
    }
    SerialB.sci->SCIFFTX.bit.TXFFINTCLR = 1;  // TX: clear interrupt flag
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;
}


//for serialC
#ifdef _FLASH
#pragma CODE_SECTION(TXCINT_data_sent, ".TI.ramfunc");
#endif
__interrupt void TXCINT_data_sent(void)
{
    char data;
    if (buf_readSCIC_1(&SerialC.TX,0,&data) == 0) {
        while ( (buf_readSCIC_1(&SerialC.TX,0,&data) == 0)
                && (SerialC.sci->SCIFFTX.bit.TXFFST != 0x10) ) {
            buf_removeSCIC(&SerialC.TX, 1);
            SerialC.sci->SCITXBUF.all = data;
        }
    } else {
        SerialC.sci->SCIFFTX.bit.TXFFIENA = 0;      // TX: disable fifo interrupt
    }
    SerialC.sci->SCIFFTX.bit.TXFFINTCLR = 1;  // TX: clear interrupt flag
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP8;
}

//for serialD
#ifdef _FLASH
#pragma CODE_SECTION(TXDINT_data_sent, ".TI.ramfunc");
#endif
__interrupt void TXDINT_data_sent(void)
{
    char data;
    if (buf_readSCID_1(&SerialD.TX,0,&data) == 0) {
        while ( (buf_readSCID_1(&SerialD.TX,0,&data) == 0)
                && (SerialD.sci->SCIFFTX.bit.TXFFST != 0x10) ) {
```

```
            buf_removeSCID(&SerialD.TX, 1);
            SerialD.sci->SCITXBUF.all = data;
        }
    } else {
        SerialD.sci->SCIFFTX.bit.TXFFIENA = 0;      // TX: disable fifo interrupt
    }
    SerialD.sci->SCIFFTX.bit.TXFFINTCLR = 1;  // TX: clear interrupt flag
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP8;
}

//for SerialA
#ifdef _FLASH
#pragma CODE_SECTION(RXAINT_recv_ready, ".TI.ramfunc");
#endif
__interrupt void RXAINT_recv_ready(void)
{
    RXAdata = SciaRegs.SCIRXBUF.all;

    /* SCI PE or FE error */
    if (RXAdata & 0xC000) {
        SciaRegs.SCICTL1.bit.SWRESET = 0;
        SciaRegs.SCICTL1.bit.SWRESET = 1;
        SciaRegs.SCIFFRX.bit.RXFIFORESET = 0;
        SciaRegs.SCIFFRX.bit.RXFIFORESET = 1;
    } else {
        RXAdata = RXAdata & 0x00FF;
        if(RXAdata == 'a') {
            GpioDataRegs.GPBCLEAR.bit.GPIO34 = 1;
        } else if(RXAdata == 's') {
            GpioDataRegs.GPBSET.bit.GPIO34 = 1;
        }
        if(RXAdata == 'z') {
            GpioDataRegs.GPASET.bit.GPIO22 = 1;
        } else if(RXAdata == 'x') {
            GpioDataRegs.GPACLEAR.bit.GPIO22 = 1;
        }
        if(RXAdata == 'c') {
            GpioDataRegs.GPCSET.bit.GPIO94 = 1;
        } else if(RXAdata == 'v') {
            GpioDataRegs.GPCCLEAR.bit.GPIO94 = 1;
        }
        if(RXAdata == 'b') {
            GpioDataRegs.GPCSET.bit.GPIO95 = 1;
        } else if(RXAdata == 'n') {
            GpioDataRegs.GPCCLEAR.bit.GPIO95 = 1;
        }
        if(RXAdata == 'm') {
            GpioDataRegs.GPDSET.bit.GPIO97 = 1;
        } else if(RXAdata == ',') {
            GpioDataRegs.GPDCLEAR.bit.GPIO97 = 1;
        }

        numRXA ++;
    }

    SciaRegs.SCIFFRX.bit.RXFFINTCLR = 1;
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;
```

```c

}

//for SerialB
#ifdef _FLASH
#pragma CODE_SECTION(RXBINT_recv_ready, ".TI.ramfunc");
#endif
__interrupt void RXBINT_recv_ready(void)
{
    RXBdata = ScibRegs.SCIRXBUF.all;

    /* SCI PE or FE error */
    if (RXBdata & 0xC000) {
        ScibRegs.SCICTL1.bit.SWRESET = 0;
        ScibRegs.SCICTL1.bit.SWRESET = 1;
        ScibRegs.SCIFFRX.bit.RXFIFORESET = 0;
        ScibRegs.SCIFFRX.bit.RXFIFORESET = 1;
    } else {
        RXBdata = RXBdata & 0x00FF;
        // Do something with recieved character
        numRXB ++;
    }

    ScibRegs.SCIFFRX.bit.RXFFINTCLR = 1;
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;
}


// for SerialC
#ifdef _FLASH
#pragma CODE_SECTION(RXCINT_recv_ready, ".TI.ramfunc");
#endif
__interrupt void RXCINT_recv_ready(void)
{
    RXCdata = ScicRegs.SCIRXBUF.all;

    /* SCI PE or FE error */
    if (RXCdata & 0xC000) {
        ScicRegs.SCICTL1.bit.SWRESET = 0;
        ScicRegs.SCICTL1.bit.SWRESET = 1;
        ScicRegs.SCIFFRX.bit.RXFIFORESET = 0;
        ScicRegs.SCIFFRX.bit.RXFIFORESET = 1;
    } else {
        RXCdata = RXCdata & 0x00FF;
        numRXC ++;

    }

    ScicRegs.SCIFFRX.bit.RXFFINTCLR = 1;
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP8;
}


//for SerialD
#ifdef _FLASH
#pragma CODE_SECTION(RXDINT_recv_ready, ".TI.ramfunc");
#endif
```

```c
__interrupt void RXDINT_recv_ready(void)
{
    RXDdata = ScidRegs.SCIRXBUF.all;

    /* SCI PE or FE error */
    if (RXDdata & 0xC000) {
        ScidRegs.SCICTL1.bit.SWRESET = 0;
        ScidRegs.SCICTL1.bit.SWRESET = 1;
        ScidRegs.SCIFFRX.bit.RXFIFORESET = 0;
        ScidRegs.SCIFFRX.bit.RXFIFORESET = 1;
    } else {
        RXDdata = RXDdata & 0x00FF;
        numRXD ++;

    }

    ScidRegs.SCIFFRX.bit.RXFFINTCLR = 1;
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP8;
}


// SerialA only setup for Tera Term connection
char serial_printf_bufSCIA[BUF_SIZESCIA];

uint16_t serial_printf(serialSCIA_t *s, char *fmt, ...)
{
    va_list ap;

    va_start(ap,fmt);
    vsprintf(serial_printf_bufSCIA,fmt,ap);
    va_end(ap);

    return serial_sendSCIA(s,serial_printf_bufSCIA,strlen(serial_printf_bufSCIA));
}

//For Text LCD
char UART_printf_buffer[BUF_SIZESCIB];
void UART_vprintfLine(unsigned char line, char *format, va_list ap)
{
   char sendmsg[24];

   int i;

   vsprintf(UART_printf_buffer,format,ap);

   // Add header information and pad end of transfer with spaces to clear display
   sendmsg[0] = 0xFE;
   sendmsg[1] = 'G';
   sendmsg[2] = 1;
   sendmsg[3] = line;
   for (i=4;i<24;i++) {
       if (i >= strlen(UART_printf_buffer)+4) {
           sendmsg[i] = ' ';
       } else {
           sendmsg[i] = UART_printf_buffer[i-4];
       }
   }
```

```c
    serial_sendSCIB(&SerialB,sendmsg,24);
}

void UART_printfLine(unsigned char line, char *format, ...)
{
    va_list ap;
    va_start(ap, format);
    UART_vprintfLine(line,format,ap);
}


```