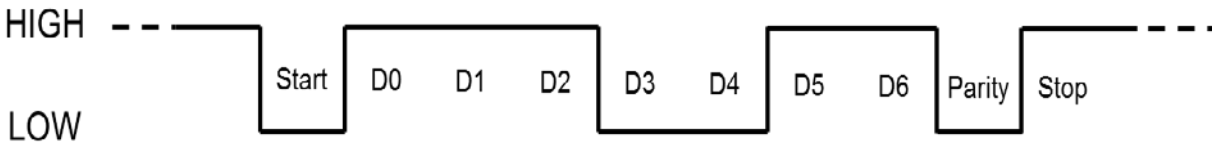
CpE 390 Microprocessor Systems

Lab 7: Asynchronous Serial I/O

1. Introduction

Serial communications is the transfer of data, one bit at a time, over a communications channel. Serial communications can either be synchronous or asynchronous. In synchronous communications, we send an explicit clock signal with the data. In asynchronous communications, we format the serial data in such a way that the clock in implicitly embedded in the serial data signal. The 68HC12 microcontroller contains on-chip support for both synchronous (e.g. SPI, I^2C) and asynchronous (RS-232) serial communications. In this lab, we will be using the asynchronous RS-232 communications link (a link that has already been established to allow the MiniIDE software on the PC to communicate with the EVB board.



The figure above illustrates the signals produced when data is transmitted across an RS-232 serial connection. When no data is being transmitted, the line is said to be idle. In this state, it is held 'high'. When data is transferred, a START bit (which is always 'low') precedes the data to signal to the receiver that a data byte is following. The data is sent LS Bit first. The last (MS Bit) may be a parity bit to allow error detection at the receiving end. After the parity bit, a STOP bit (which is always 'high') returns the line to its idle state, ready to transmit the next byte.

Serial I/O on the 68HC12 is performed by the Serial Communication Interface (SCI) module. The SCI is controlled, through software, by a number of registers. These registers are special memory locations that transfer data to and from hardware. There are three types of registers used in a peripheral interface: (i) control registers for determining the operation of the peripheral (ii) status registers which tell us the current state of the peripheral and (iii) data registers through which you can send and receive data.

There are two separate serial communication ports on the 68HC12: SCI0 and SCI1. We will be using SCI0. The registers associated with SCI0 are:

```
SCI0BDH     EQU   $00C8     ; high byte of SCI0 baud rate control register
SCI0BDL     EQU   $00C9     ; low byte of SCI0 baud rate control register
```

```
SCI0CR1     EQU   $00CA     ; SCI0 Control Register 1
SCI0CR2     EQU   $00CB     ; SCI0 Control Register 2
SCI0SR1     EQU   $00CC     ; SCI0 Status Register 1
SCI0SR2     EQU   $00CD     ; SCI0 Status Register 2
SCI0DRH     EQU   $00CE     ; high byte of SCI0 data register
SCI0DRL     EQU   $00CF     ; low byte of SCI0 data register
```

SCI0DRH is only needed when sending or receiving 9-bit data (9 data bits plus parity). When using 8-bit data (7 data bits plus parity), only the SCI0DRL data register is used.

The baud rate and control registers of the SCI0 port have already been set up by the DBUG12 monitor to allow communication with the MiniIDE software on the PC. You will not need to modify these registers in order to use the SCI0 to perform simple serial I/O.

2. Procedure

(a) Use the monitor program to examine the contents of the SCI0 baud rate and control registers. You can find a detailed description of the meaning of the various fields of these registers in your lecture notes and/or text-book. Explain the settings that have already been programmed into these registers by the DBUG12 monitor.

(b) Write a program to be run on your demo board. You may use the input-output routines that were given as part of the lecture notes. They are also reproduced in Appendix A.

The program and the user should interact as follows:

1. The program outputs the message “Please enter your name:” and waits for the user to enter his or her name followed by a CR (ENTER).
2. The program then outputs the message “Please enter your age:” and waits for the user to enter his or her age followed by a CR (ENTER).
3. The program outputs the message “Please enter your height in inches:” and waits for the user to enter his or her height followed by a CR (ENTER). The program then prints out the following message on the terminal screen and exits:

   Your name is www
   You are xxx years old
   You are yyy inches tall

4. In Appendix B, you will find two more routines:

The first *asctobin* converts a decimal ASCII string into an unsigned 8-bit integer. The address of the ASCII string is passed in register X. The routine returns the binary value of the integer in accumulator B.

The second *bintoasc* does the opposite – it converts an unsigned 8-bit integer into a 3 digit decimal ASCII string. You pass the binary value of the integer in accumulator B and the address of where you want the routine to place the ASCII string in register X.

Modify your program so that before it prints out your name, age and height, it first uses the *asctobin* routine to convert your height into a binary value and stores it in a named 8-bit memory location. You should then use the bintoasc routine to convert this binary value back to ASCII and use the converted string when you output the person's height. This will, of course, give the same answer. This is just to check that you are using these routines correctly.

5. Now that we have your height in inches as a binary value, you can perform arithmetic on this value. Modify your program to convert your height in inches to centimeters and your print-out sequence to look like:

   Your name is www
   You are xxx years old
   You are zzz cms tall

6. Show your working program to the TA

Appendix A

```
SCI0SR1:    EQU     $00CC               ;serial communications status register
SCI0DRL:    EQU     $00CF               ;serial communications data register
CR:         EQU     $0D                 ; carriage return (ENTER) code
LF:         EQU     $0A                 ; line-feed code
BS:         EQU     $08                 ; back-space code
WS:         EQU     $20                 ; (white) space code

; Output the character passed in Accumulator A to the SCI0
putcSCI0:   brclr   SCI0SR1, $80, *     ; wait for TDRE to be set
            staa    SCI0DRL             ; output the character
            rts

; Read a character from the SCI0 and return it in Accumulator A
getcSCI0:   brclr   SCI0SR1, $20, *     ; wait for RDRF to be set
            ldaa    SCI0DRL             ; output the character
            rts

; Output a null terminated string to the SCI0. A pointer to the string is passed in register X
putsSCI0:   ldaa    1, x+               ; get a character and increment pointer
            beq     done                ; end of string?
            jsr     putcSCI0
            bra     putsSCI0            ; go to next character
done:       rts

; Read a CR terminated string from the SCIO and save it as a null terminated string at the
; location pointed to by register X. Echo all input characters and use the backspace character
; to erase previously input characters
getsSCI0:   jsr     getcSCI0            ; get a character from SCIO
            cmpa    #CR                 ; is it carriage return?
            beq     stend
            staa    0,x                 ; save the character
            jsr     putcSCI0            ; echo the character
            cmpa    #BS                 ; is it a backspace character?
            bne     nc                  ; no, continue
            dex                         ; decrement buffer pointer
            ldaa    #WS                 ; output space character (to erase previous)
            jsr     putcSCI0
            ldaa    #BS                 ; backspace over space character
            jsr     putcSCI0
            bra     getsSCI0            ; go get next character
nc:         inx                         ; increment pointer
            bra     getsSCI0
stend:      ldaa    #LF                 ; echo LF for CR
            jsr     putcSCI0
```

```
        clr     0,x             ;terminate string with NULL
        rts
```

Appendix B

```
; This routine converts a decimal ASCII string into an unsigned 8-bit integer.
; A pointer to the ASCC string is passed in X. The result in returned in B

asctobin:   psha                    ; save A
            pshy                    ; save Y
            ldy     #0              ; initialize value to zero
abloop:     ldaa    1,X+            ; get next ascii char
            cmpa    #$30            ; check for numeric character
            blo     abdone          ; if char<'0', then go to next char
            cmpa    #$39
            bhi     abdone          ; if char>'9', then go to next char
            suba    #$30            ; convert from ascii to binary
            psha                    ; temp save value on stack
            tfr     Y, B            ; mov accum value to B
            ldaa    #10
            mul                     ; mult accum value in B by 10
            tfr     D, Y            ; and store back in Y
            pulb                    ; restore value of new char
            aby                     ; and add to accum value in Y
            bra     abloop          ; go to next character
abdone:     tfr     Y, B            ; move result into B
            puly                    ; restore registers
            pula
            rts                     ; and return


; This routine converts an 8-bit unsigned integer into a decimal ASCII string
; The 8-bit value is passed in B.
; A pointer to where the routine should form the string should be passed in X.

bintoasc:   psha                    ; save A
            pshy                    ; save Y
            tfr     X, Y            ; use Y as pointer register
            ldaa    #0              ; clear MSByte of D
            ldx     #10
            idiv                    ; divide by 10
            addb    #$30            ; convert remainder to ascii
            stab    2, Y            ; store ones Digit
            tfr     X, D            ; move quotient to D
```

```
        ldx     #10
        idiv                    ; divide by 10
        addb    #$30            ; convert remainder to ascii
        stab    1,Y             ; store tens digit
        tfr     X, D            ; move quotient to D
        addb    #$30            ; convert to ascii
        stab    0, Y            ; store hundreds digit
        clr     3, Y            ; place NULL at end of string
        cmpb    #$30            ; suppress leading zeros
        bne     baexit
        movb    #WS, 0,Y        ; if hundreds digit='0', replace by space
        ldab    1,Y
        cmpb    #$30
        bne     baexit
        movb    #WS, 1,Y        ; if tens digit='0', replace by space
baexit: puly                    ; restore Y
        pula                    ; restore A
        rts
```