CHAPTER 8

SERIAL COMMUNICATIONS INTERFACE (SCI)

In this chapter, we will discuss the Serial Communication Interface (SCI) of the TMS320F2407. These interfaces allow digital communication between the controller and external peripherals / other controller, in asynchronous and isosynchronous modes.

8.1 Introduction to Serial Communication

In this laboratory, the serial communication between a TMS320LF2407 EVM and a PC will be implemented and tested although the term “serial communication” itself can be generalized to PC-to-PC, DSP-to-DSP, and so on. Without losing generality, the introduction to the fundamentals of serial communication starts from what you may be more familiar with – a PC:

All IBM PC and compatible computers are typically equipped with two serial ports and one parallel port. Although these two types of ports are used for communicating with external devices, they work in different ways.

A parallel port sends and receives data eight bits at a time over 8 separate wires. This allows data to be transferred very quickly; however, the cable required is more bulky because of the number of individual wires it must contain. Parallel ports are typically used to connect a PC to a printer and are rarely used for much else. A serial port sends and receives data one bit at a time over one wire. While it takes eight times as long to transfer each byte of data this way, only a few wires are required. In fact, two-way (full duplex) communications is possible with only three separate wires - one to send, one to receive, and a common signal ground wire.

Bi-Directional Communications

The serial port on your PC is a full-duplex device meaning that it can send and receive data at the same time. In order to be able to do this, it uses separate lines for transmitting and receiving data. Some types of serial devices support only one-way

communications and therefore use only two wires in the cable - the transmit line and the signal ground.

Communicating by Bits

Once the start bit has been sent, the transmitter sends the actual data bits. There may either be 5, 6, 7, or 8 data bits, depending on the number you have selected. Both receiver and the transmitter must agree on the number of data bits, as well as the baud rate. Almost all devices transmit data using either 7 or 8 data bits.

Notice that when only 7 data bits are employed, you cannot send ASCII values greater than 127. Likewise, using 5 bits limits the highest possible value to 31. After the data has been transmitted, a stop bit is sent. A stop bit has a value of 1 - or a mark state - and it can be detected correctly even if the previous data bit also had a value of 1. This is accomplished by the stop bit's duration. Stop bits can be 1, 1.5, or 2 bit periods in length.

The Parity Bit

Besides the synchronization provided by the use of start and stop bits, an additional bit called a parity bit may optionally be transmitted along with the data. A parity bit affords a small amount of error checking, to help detect data corruption that might occur during transmission. You can choose even parity, odd parity, mark parity, space parity or none at all. When even or odd parity is being used, the number of marks (logical 1 bit) in each data byte are counted, and a single bit is transmitted following the data bits to indicate whether the number of 1 bits just sent is even or odd.

For example, when even parity is chosen, the parity bit is transmitted with a value of 0 if the number of preceding marks is an even number. For the binary value of 0110 0011 the parity bit would be 0. If even parity is in effect and the binary number 1101 0110 is sent, then the parity bit would be 1. Odd parity is just the opposite, and the parity bit is 0 when the number of mark bits in the preceding word is an odd number. Parity error checking is very rudimentary. While it will tell you if there is a single bit error in the character, it doesn't show which bit was received in error. Also, if even number of bits are in error then the parity bit would not reflect any error at all.

Mark parity means that the parity bit is always set to the mark signal condition and likewise space parity always sends the parity bit in the space signal condition. Since these two parity options serve no useful purpose whatsoever, they are almost never used.

RS-232C

RS-232 stands for Recommend Standard number 232 and C is the latest revision of the standard. The serial ports on most computers use a subset of the RS-232C standard. The full RS-232C standard specifies a 25-pin "D" connector of which 22 pins are used. Most of these pins are not needed for normal PC communications, and indeed, most new PCs are equipped with male D type connectors having only 9 pins.

Baud vs. Bits per Second

The baud unit is named after Jean Maurice Emile Baudot, who was an officer in the French Telegraph Service. He is credited with devising the first uniform-length 5-bit code for characters of the alphabet in the late 19th century. What baud really refers to is modulation rate or the number of times per second that a line changes state. This is not always the same as bits per second (BPS). If you connect two serial devices together using direct cables then baud and BPS are in fact the same. Thus, if you are running at 19200 BPS, then the line is also changing states 19200 times per second. But when considering modems, this isn't the case.

Because modems transfer signals over a telephone line, the baud rate is actually limited to a maximum of 2400 baud. This is a physical restriction of the lines provided by the phone company. The increased data throughput achieved with 9600 or higher baud modems is accomplished by using sophisticated phase modulation, and data compression techniques.

Synchronous and Asynchronous Communications

There are two basic types of serial communications, synchronous and asynchronous. With Synchronous communications, the two devices initially synchronize themselves to each other, and then continually send characters to stay in sync. Even when data is not really being sent, a constant flow of bits allows each device to know where the other is at any given time. That is, each character that is sent is either actual data or an

idle character. Synchronous communications allows faster data transfer rates than asynchronous methods, because additional bits to mark the beginning and end of each data byte are not required. The serial ports on IBM-style PCs are asynchronous devices and therefore only support asynchronous serial communications.

Asynchronous means "no synchronization", and thus does not require sending and receiving idle characters. However, the beginning and end of each byte of data must be identified by start and stop bits. The start bit indicate when the data byte is about to begin and the stop bit signals when it ends. The requirement to send these additional two bits cause asynchronous communications to be slightly slower than synchronous however it has the advantage that the processor does not have to deal with the additional idle characters.

An asynchronous line that is idle is identified with a value of 1, (also called a mark state). By using this value to indicate that no data is currently being sent, the devices are able to distinguish between an idle state and a disconnected line. When a character is about to be transmitted, a start bit is sent. A start bit has a value of 0, (also called a space state). Thus, when the line switches from a value of 1 to a value of 0, the receiver is alerted that a data character is about to come down the line.

8.2 Serial Communication Interface of 2407

(Read the TI document: TMS320LF/LC240x DSP Controllers Reference Guide - System and Peripherals, Chapter 8 – Serial Communication Interface)

8.3 DSP Program

Having studied the details of the SCI communication, let us now take a look at an example in order to understand how the SCI is programmed.

Example 1:

Write a program to initialize the SCI in idle-line mode with 8 character bits, 1 stop bit and odd parity. The baud rate of transmission and reception should be 19200 bps. The program should conduct asynchronous communication between 2407 and its host PC. The DSP receives characters from the PC keyboard input and transmits characters to HyperTerminal screen on the PC. User enters a password through HyperTerminal using

the keyboard. The DSP examines the value of the password and returns “Right” if it is correct or “Wrong” if it is incorrect. About the usage of HyperTerminal, please refer to the lab procedure below.

Solution:

Registers involved:

SCICCR = 0027h

Number of stop bits is 1. This is set by bit 7 = 0.

Odd parity is set by bit 6 = 0.

Parity is enabled by setting bit 5 = 1.

Loop back test mode is disabled by setting bit 4 =0.

The IDLE MODE is selected by setting bit 3 = 0.

8 bits per character are selected by setting bits 2-0 to 111.

SCICTL1 = 0003h to reset and 0023h to enable SCI

Bit 7 Reserved.

Bit 6 =0. Receive error interrupt is disabled.

Bit 5 = 0. SCI software reset initializes the SCI state machines and operating flags (SCICTL2 and SCIRXST) to the reset condition. Set it to 1 to enable SCI.

Bit 4 = 0. Reserved.

Bit 3 = 0. Transmit feature is not selected

Bit 2 = 0. SCI Sleep mode is disabled.

Bit 1 = 1. SCI transmitter enable.

Bit 0 = 1. SCI receiver enable.

SCICTL2 = 0002h

Bit 1 = 1. Receiver-buffer/break interrupt enabled.

Bit 0 = 0. Disable TXRDY interrupt.

SCIHBAUD = 0000h

SCILBAUD = 194

SCI 16-bit baud selection. SCIHBAUD (MSbyte) and SCILBAUD (LSbyte) concatenate to form a 16-bit baud value. The internally generated serial clock is determined by the CPUCLK and the two baud select registers. The SCI uses the 16-bit value of these registers to select one of 64K serial clock rates for the communication modes. The SCI baud rate for the different communication modes is determined in the following ways:

- For BRR = 1 to 65 535

SCI asynchronous baud = {CPUCLK / [(BRR + 1) * 8]}

- For BRR = 0

SCI asynchronous baud = CPUCLK / 16

Thus, in our problem,

BRR = $\{30\times10^6/(19200*8)\}$ - 1 = 194

SCIPRI = 0000h

Set all interrupts to be high-priority requests (INT1)

The complete program listing is as follows – This program is about serial communication between a DSP and a PC. The software running on the PC side and the relative settings are described in Laboratory 7.

```
;*********************************************************************
; File Name:       ch8_e1.asm
; Target System: C240x Evaluation Board
; Description:
;     This program uses the SCI module to implement a simple
;     asynchronous communications routine. The SCI is initialized
;     to operate in idleline mode with 8 character bits, 1 stop
;     bit, and odd parity. The SCI Baud Rate Registers (BRR) are
;     set to transmit and receive data at 19200 baud. The SCI
;     generates an interrupt every time a character is loaded into
;     the receive buffer (SCIRXBUF). The interrupt service routine
;     (ISR)reads the receive buffer and determines if the carriage
;     return button, <CR>,has been pressed. If so, the character
;     string is compared with the value of password expected. If the
;     password is correct, the character string "Right" is transmitted
;     otherwise, the character string "Wrong" is transmitted.
;
;*********************************************************************

;*********************************************************************
;                              SYSTEM OPTIONS
;*********************************************************************
real_time   .set  0     ; 1 for real time mode, otherwise set 0
LENGTH      .set  16    ; length of the data stream to be transmitted
LENGTH2     .set  8
PASWD       .set  4321h ; 4 digit numeric password
```

```
B2_SADDR    .set  60h   ; On-chip DRAM B2 starting address (data memory)

;***********************************************************************

;-----------------------------------------------------------------------
; External references
;-----------------------------------------------------------------------
            .include    "f2407.h"
;           .global     MON_RT_CNFG

            .ref        SYS_INIT

;-----------------------------------------------------------------------
; Local Variable Declarations
;-----------------------------------------------------------------------
            .def  GPR0        ;General purpose register.

            .bss  GPR0,1      ;General purpose register.
            .bss  RX_PASWD,1  ;Storage for received password
            .bss  CHR_CNT,1   ;Counter for number of characters received

;=======================================================================
; V E C T O R    T A B L E    ( including RT monitor traps )
;=======================================================================
;        .include "c200mnrt.i" ; Include conditional assembly options.

            .global _c_int0,PHANTOM,GISR1,GISR2,GISR3,GISR4,GISR5,GISR6

;=======================================================================
; M A I N   C O D E  - starts here
;=======================================================================
            .text
_c_int0:
            CALL        SYS_INIT          ;DSP initialization

;-----------------------------------------------------------
; Initialise the Real time monitor
;-----------------------------------------------------------

            POINT_PG0

;---Real Time option---------------
      .if (real_time)
            CALL        MON_RT_CNFG       ;For Real-Time
      .endif
;--------------------------------

;-----------------------------------------------------------
; System Interrupt Init.
;-----------------------------------------------------------

;---Real Time option ------------------------------------------------
      .if (real_time)
            SPLK  #0000000001000001b,IMR  ;En Int lvl 1 & 7
                  ;|||||||||!||||||       for SCI and RT
                  ;5432109876543210
      .endif

      .if (real_time != 1)
```

```
            SPLK  #0000000000000001b,IMR  ;Enable Int 1 only for SCI
                  ;||||!!!!!!!!!!!!!!!
                  ;5432109876543210
      .endif

            SPLK  #0FFFFh, IFR            ;Clear any pending Ints
;-----------------------------------------------------------------------

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup shared I/O pins
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #DP_PF2                 ;set data page

        SPLK    #0000000000000011b,MCRA ;set SCI TX & RX pins
*                 ||||||||||||||||
*                 FEDCBA9876543210
* bit 15        0:      0=IOPB7,     1=TCLKINA
* bit 14        0:      0=IOPB6,     1=TDIRA
* bit 13        0:      0=IOPB5,     1=T2PWM/T2CMP
* bit 12        0:      0=IOPB4,     1=T1PWM/T1CMP
* bit 11        0:      0=IOPB3,     1=PWM6
* bit 10        0:      0=IOPB2,     1=PWM5
* bit 9         0:      0=IOPB1,     1=PWM4
* bit 8         0:      0=IOPB0,     1=PWM3
* bit 7         0:      0=IOPA7,     1=PWM2
* bit 6         0:      0=IOPA6,     1=PWM1
* bit 5         0:      0=IOPA5,     1=CAP3
* bit 4         0:      0=IOPA4,     1=CAP2/QEP2
* bit 3         0:      0=IOPA3,     1=CAP1/QEP1
* bit 2         0:      0=IOPA2,     1=XINT1
* bit 1         1:      0=IOPA1,     1=SCIRXD
* bit 0         1:      0=IOPA0,     1=SCITXD

;=====================================================================
; Initialize user variables
;=====================================================================
            LDP   #RX_PASWD
            SPLK  #0, RX_PASWD
            SPLK  #0, CHR_CNT
;=====================================================================
; Initialize B2 RAM to zero's.
;=====================================================================
            LAR   AR2,#B2_SADDR     ; AR2 <- B2 start address
            MAR   *,AR2             ; Set ARP=AR2
            LACL  #0                ; Set ACC = 0
            RPT   #1Fh              ; Set repeat cntr for 31+1 loops
            SACL  *+                ; Write zeros to B2 RAM
;=====================================================================
; Initialize DATAOUT with data to be transmitted.
;=====================================================================
            LAR   AR2,#B2_SADDR     ;Reset AR2 B2 start address
            RPT   #(LENGTH - 1)     ;set repeat counter for LENGTH loops
            BLPD  #TXDATA,*+        ;loads B2 with TXDATA

;=====================================================================
; INITIALIZATION OF INTERRUPT DRIVEN SCI ROUTINE
;=====================================================================
SCI_INIT:
            LDP   #DP_PF1
```

```
            SPLK  #0027h, SCICCR    ;1 stop bit,odd parity,8 char bits,
                                     ;async mode, idleline protocol
            SPLK  #0003h, SCICTL1   ;Enable TX, RX, internal SCICLK,
                                     ;Disable RX ERR, SLEEP, TXWAKE
                                     ;Reset SCI
            SPLK  #0002h, SCICTL2   ;Enable RX INT,disable TX INT
            SPLK  #0000h, SCIHBAUD
            SPLK  #194, SCILBAUD    ;Baud Rate=19200 b/s (30 MHz CLKOUT)
                                     ; 30000000/19200/8-1 = 194
            SPLK  #0000h, SCIPRI    ;Set TXD & RXD INT high priority
            SPLK  #0023h, SCICTL1   ;Enable TX, RX, internal SCICLK,
                                     ;Disable RX ERR, SLEEP, TXWAKE
                                     ;Enable SCI
            LAR   AR0, #SCITXBUF    ;Load AR0 with SCI_TX_BUF address
            LAR   AR1, #SCIRXBUF    ;Load AR1 with SCI_RX_BUF address
            LAR   AR2, #B2_SADDR    ;Load AR2 with TX data start address

            CLRC  INTM              ;Enable interrupts

WAIT: B     WAIT

;=====================================================================
; I S R           INT1 ISR
; Description:
;           The GISR1 first determines if the SCI RXINT caused
;           the interrupt. If so, the SCI Specific ISR reads the
;           character in the RX buffer. If the character received
;           corresponds to a carriage return, <CR>, the character
;           string received is compared with the expected value of
;           password. If the value matches, "Right" is transmitted,
;           else "Wrong" is transmitted. If the character received
;           does NOT correspond to a carriage return, <CR>, then
;           the ISR returns to the main program without transmitting
;           a character string. If the SCI RXINT did not cause an
;           interrupt, then the value ’0x0bad’is stored in the
;           accumulator and program gets caught in the BAD_INT
;           endless loop.
;=====================================================================

GISR1       LDP   #DP_PF1
            LACL  PIVR              ;Load peripheral INT vector address
            SUB   #0006h            ;Subtract RXINT offset from above
            BCND  SCI_ISR,EQ        ;verify RXINT initiated interrupt
            B     BAD_INT           ;Else, bad interrupt occurred
SCI_ISR     LDP   #CHR_CNT
            LARP  1                 ;Set ARP=AR1
            LACC  *                 ;Load ACC w/RX buffer character
            SUB   #000Dh            ;Check if <CR> has been pressed:
            BCND  XMIT_CHAR,EQ      ;YES? Transmit data.
            LACC  *
            SUB   #30h              ;Extract the actual value from the
                                     ;ASCII equivalent of the number.
            RPT   CHR_CNT
            SFL
            SFR
            OR    RX_PASWD
            SACL  RX_PASWD
            LACC  CHR_CNT
            ADD   #4
            SACL  CHR_CNT
```

```
            B     ISR_END           ;N0? Return from GISR1.

XMIT_CHAR
            SPLK  #0, CHR_CNT
            LAR   AR2, #B2_SADDR
            LACC  RX_PASWD
            SUB   #PASWD
            SPLK  #0, RX_PASWD      ;Check if received no. = password
            BCND  XMIT_CHAR1, EQ
            LAR   AR2, #(B2_SADDR + LENGTH2) ; if not point to starting
                                    ; address of the string "Wrong"

XMIT_CHAR1
            LARP  2
            LACC  *+,AR0            ;Load char to be xmitted into ACC
            BCND  ISR_END1,EQ       ;Check for Null Character
                                    ;YES? Return from GISR1.
            SACL  *,AR2             ;NO? Load char into xmit buffer.
XMIT_RDY
            LDP   #DP_PF1
            BIT   SCICTL2,BIT7      ;Test TXRDY bit
            BCND  XMIT_RDY,NTC      ;If TXRDY=0,then repeat loop
            B     XMIT_CHAR1        ;else xmit next character
ISR_END1
            B     ISR_END
ISR_END     LAR   AR2, #B2_SADDR    ;Reload AR2 w/ TX data start address

            CLRC  INTM              ;Clear INT Mask flag
            RET                     ;Return from GISR1

BAD_INT     LACC  #0BADh            ;Load ACC with "bad"
            B     BAD_INT           ;Repeat loop

;=========================================================================
; I S R  -  PHANTOM
;
; Description:    Dummy ISR, used to trap spurious interrupts.
;=========================================================================
PHANTOM B   PHANTOM
;GISR1      RET
GISR2       RET
GISR3       RET
GISR4       RET
GISR5       RET
GISR6       RET

; Initialize Transmit Data for Interrupt Service Routine
            .data
TXDATA      .word 0052h       ;Hex equivalent of ASCII character 'R'
            .word 0069h       ;Hex equivalent of ASCII character 'i'
            .word 0067h       ;Hex equivalent of ASCII character 'g'
            .word 0068h       ;Hex equivalent of ASCII character 'h'
            .word 0074h       ;Hex equivalent of ASCII character 't'
            .word 000Ah       ;Hex equivalent of line feed
            .word 000Dh       ;Hex equivalent of carriage return
            .word 0000h       ;Hex equivalent of NULL

WRONG       .word 0057h       ;Hex equivalent of ASCII character 'W'
            .word 0072h       ;Hex equivalent of ASCII character 'r'
            .word 006fh       ;Hex equivalent of ASCII character 'o'
```

```
            .word 006eh       ;Hex equivalent of ASCII character 'n'
            .word 0067h       ;Hex equivalent of ASCII character 'g'
            .word 000Ah       ;Hex equivalent of line feed
            .word 000Dh       ;Hex equivalent of carriage return
            .word 0000h       ;Hex equivalent of NULL
```

Laboratory Experiment 7

Objectives

To use serial communication using the SCI module of the DSP and the RS232 interface of the EVM to establish communication with the PC COM port.

Discussion

RS232 Interface on 2407 EVM

The 'C240x evaluation board has an RS-232 compatible DB-9 serial port for asynchronous communication. The DB-9 serial port (P6) interfaces to the SCI peripheral on the 'LF2407 device through an RS-232 transceiver. This RS232 connector allows the user to connect an external instrument or computer to the EVM320LF2407. This means data can be logged or commands given to the control algorithm. Five RS-232 signals can be used to implement various communications protocols with software and hardware handshaking on the 'C240x EVM. These signals are:

_ Receive data (RX)

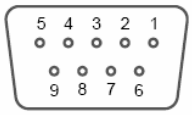
_ Transmit data (TX)

_ Clear to send (CTS)

_ Request to send (RTS)

_ Data terminal ready (DTR)

The pin positions for the P6 connector as viewed from the edge of the EVM320LF2407.



The pin numbers and their corresponding signals are shown in the table below:

Table 7: P6 RS232 Pinout

Pin #	PC (female)	SD EVM
2	Rx, input	Tx, output
3	Tx, output	Rx, input
4	DTR, output	Reset/CTS, input
5	GND	GND
8	CTS, input	RTS, output

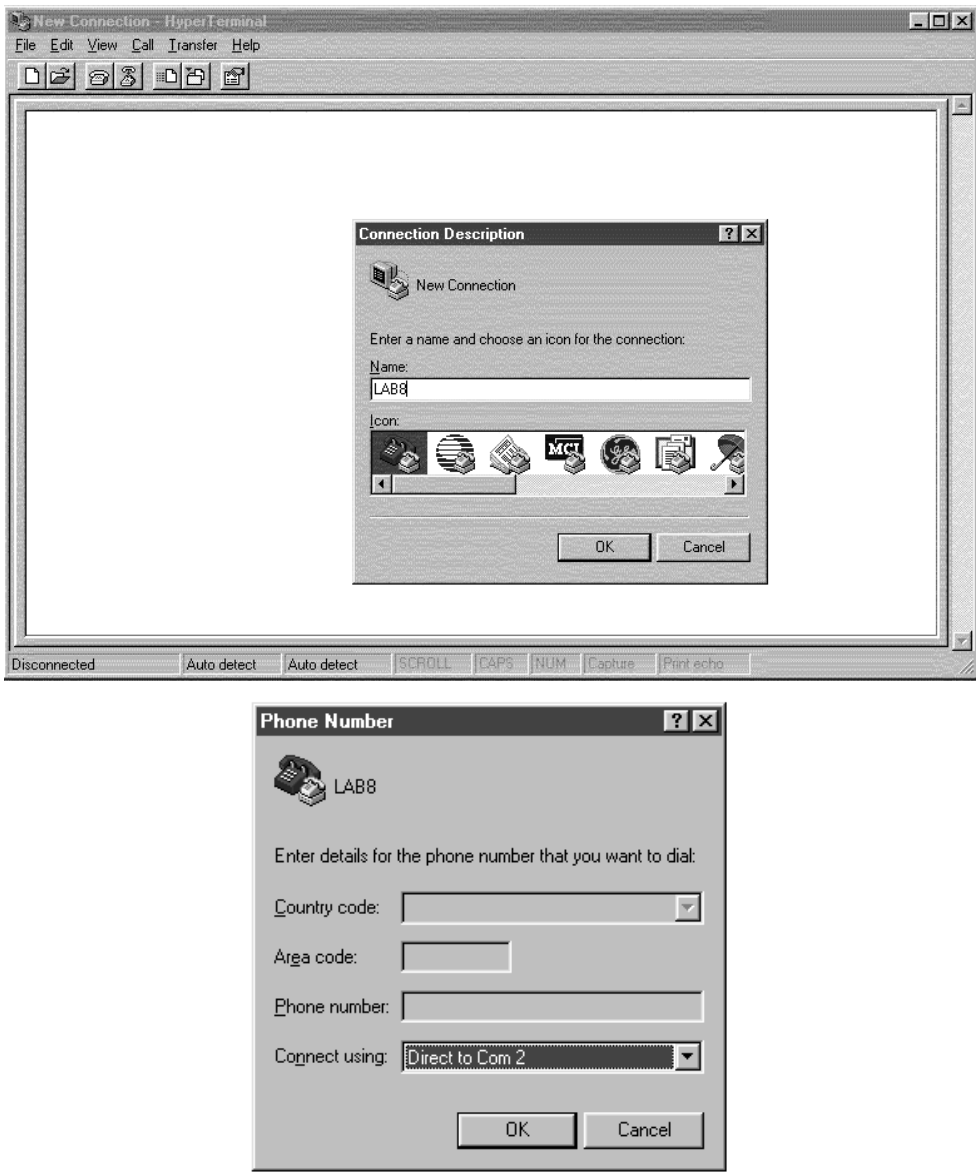
An RS-232 cable is required to connect the serial port on the evaluation board to a host. The pin-out for the evaluation board and host serial ports is provided in Table below. The cable labeled RS232 Cable conforms to these specifications.

The HyperTerminal:

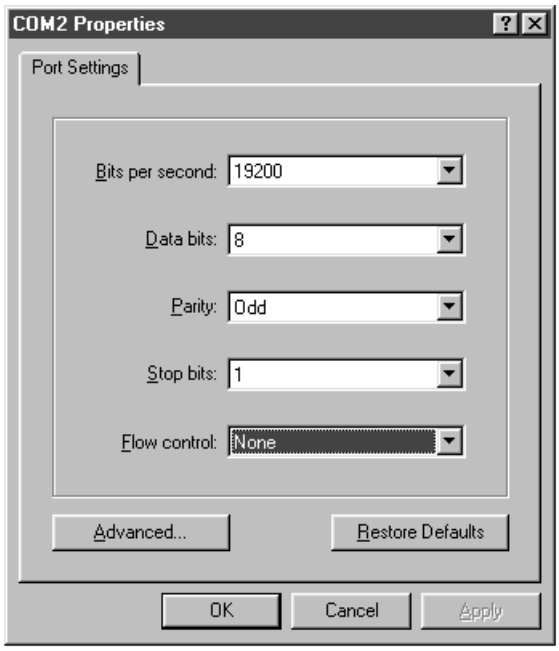
In order to test the program example discussed in the previous section, we employ the Windows HyperTerminal. This is a program that allows asynchronous communication with the PC. The protocol can be specified in this software.

Procedure:

1. Turn off PC and EVM. Connect the cable labeled "RS232 Cable" such that the 9-pin connector end is connected to DB-9 on EVM, and the 9-pin connector end is connected to the COM1 port on the PC. Now power on the PC and EVM.
2. Launch HyperTerminal from “Accessories” of Windows. The following menu pops up. Enter the name of the session: e.g. LAB7. Select the port COM1, since this is the port we are going to use for communication.



3. Another menu pops up asking for the configuration details. Set up the port as shown in the figure below:



4. Once the Hyperterm settings are done, start the C-source debugger and run the program. Now enter a password 1234, the terminal responds with a "Right". Try another password e.g. 5476, the terminal responds with a "Wrong". This is clearly shown in figure below:

Laboratory Assignments

1. Add comments for all lines in the interrupt service routine of the example program `Ch8_e1.asm` where there are no comment lines existing.
2. Write a program to enhance the password checking mechanism shown in the example – add features to simulate a real password checking interface:

- Display a password input prompt on the Hyperterm screen at the beginning;
- Display an asterisk “*” for each digit of password input;

- If wrong, give prompt to reenter;
- If wrong for 3 times, display some fail information;
- If correct, display some information indicating the success.

Hint: use short words for prompts since the memory space in B2 is only 1Fh, so make your prompts short.

The ASCII code table is attached below:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL