Asian Journal of Information Technology 5 (10): 1044-1049, 2006
© Medwell Online, 2006

Software Implementation of Wireless Modem Using RS232

Ankur Singh and Jasvir Singh
Department of Electronics Technology,
Guru Nanak Dev University, Amritsar, Punjab, India

Abstract: The word modem today's world is synonymous with communication. It is a signal converter that mediates the communication between a computer and the telephone network. It is an essential component of every network installation. The modem is entirely dependent upon telephone lines for communication. Busy telephone lines and line faults always pose a problem for a computer professional. Also the limited bandwidth of telephone lines limits the maximum rate of transmission of data that can be attained by modem. So in order to overcome the constraints of conventional modem, there comes a new concept called as wireless modem which transmits and receives data in the RF band thus eliminating the need of telephone lines totally with much improved bandwidth. The present study deals with software implementation of wireless modem using RS232 cable.

Key words: Software, wireless modem, bandwidth

INTRODUCTION

The wireless modem is a peripheral device that is connected to the computer for transmission/reception of data. A wireless modem is a modem that is connected to wireless network instead of telephone system. A wireless modem can use different modulation techniques for transferring the information signal (data). The various modulating techniques that can be used for transferring the data are[1-2] :

- Frequency Shift Keying (FSK)
- Phase Shift Keying (PSK)
- Amplitude Shift Keying (ASK)
- Quadrature Amplitude Modulation (QAM)

Each modulation technique has its own properties and advantages. Depending upon the modulation technique, a wireless modem can have different speed and frequency range. The FSK based wireless operates at speeds comparable to dial up modems, not anywhere near the speed of broadband internet connections. But a wireless modem is still a better option than a dial up modem[3-4].

This study will demonstrate how the wireless modem is interfaced with the computer so that data can be transmitted/received by computer at user end via wireless modem. In this study we will make a program to control the flow of data from computer to wireless modem. In this program we will modify the registers of UART to set properties baud rate, parameter count etc. to transmit data in a controlled manner.

The Fig.1 shows the overview of the interface of wireless modem with computer.

The hardware part of power supply, transmitter and receiver part of the wireless modem has been designed and developed[5].

The major components used for programming the interface are: RS232 and UART. In this we will initially discuss the RS232 cable, serial ports and registers of UART. After that it will be shown that how to program the interfacing of wireless modem with the computer.

RS232 cable: The RS232 is an asynchronous serial communications protocol, widely used on computers. Asynchronous means it doesn't have any separate synchronizing clock signal, so it has to synchronize itself to the incoming data-it does this by the use of 'START' and 'STOP' pulses. The signal itself is slightly unusual for computers, as rather than the normal 0V to 5V range, it uses+12V to -12V - this is done to improve reliability and greatly increases the available range it can work over. The EIA (Electronics Industry Association) standard specifies a maximum open-circuit voltage of 25 volts; signal levels of ±5 V, ±10 V, ±12 V and ±15 V are all commonly seen depending on the power supplies available within a device.

Serial Ports come in two sizes. There are the D-Type 25 pin connector and the D-Type 9 pin connector both of which are male on the back of the PC, thus you will require

Corresponding Author: Ankur Singh, Department of Electronics Technology, Guru Nanak Dev University, Amritsar, Punjab, India

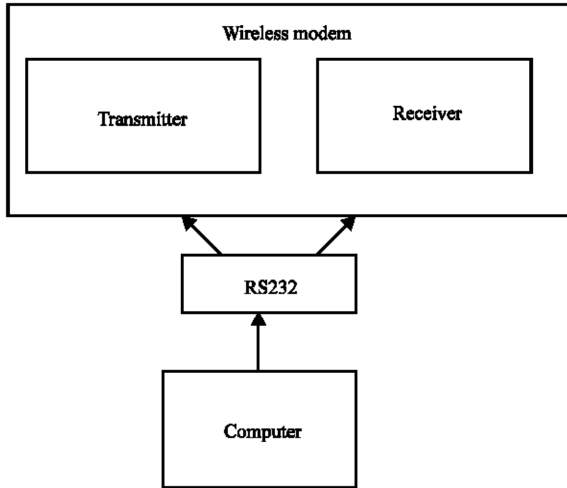
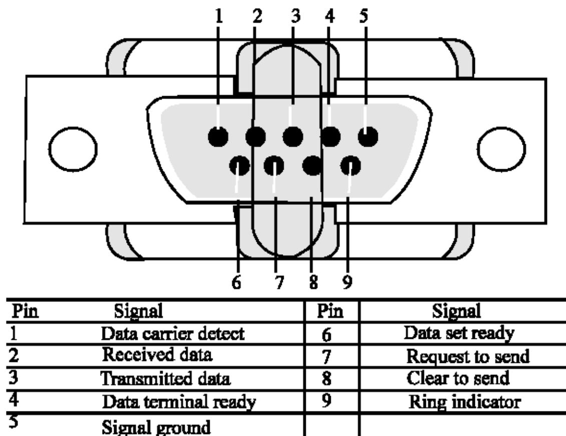


Fig. 1: Block diagram of interfacing of wireless modem with computer via RS232 cable



Pin	Signal	Pin	Signal
1	Data carrier detect	6	Data set ready
2	Received data	7	Request to send
3	Transmitted data	8	Clear to send
4	Data terminal ready	9	Ring indicator
5	Signal ground		

Fig. .2: Pin out diagram of RS232

a female connector on your device. In this study we will use 9 pin RS232 connector. Fig.2 shows the pin out diagram.

RS232 waveform: RS-232 communication is asynchronous. That is a clock signal is not sent with the data. Each word is synchronized using it's start bit and an internal clock on each side, keeps tabs on the timing.

The diagram above shows the expected waveform from the UART when using the common 8N1 format. 8N1 signifies eight Data bits, No Parity and 1 Stop Bit. The RS-232 line, when idle is in the Mark State (Logic 1). A transmission starts with a start bit which is (Logic 0). Then each bit is sent down the line, one at a time. The LSB (Least Significant Bit) is sent first. A Stop Bit (Logic 1) is then appended to the signal to make up the transmission.

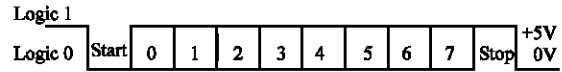


Fig. 3: TTL/CMOS serial logic waveform

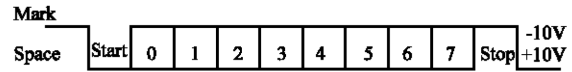


Fig. 4: RS-232 logic waveform

Fig. 3 shows the next bit after the Stop Bit to be Logic 0. This must mean another word is following and this is its Start Bit. If there is no more data coming then the receive line will stay in its idle state (logic 1). We have encountered something called a Break Signal. This is when the data line is held in a Logic 0 state for a time long enough to send an entire word. Therefore if you don't put the line back into an idle state, then the receiving end will interpret this as a break signal.

The data sent using this method, is said to be framed. That is the data is framed between a Start and Stop Bit. Should the Stop Bit be received as Logic 0, then a framing error will occur. This is common, when both sides are communicating at different speeds.

Fig. 4 is only relevant for the signal immediately at the UART. RS-232 logic levels uses +3 to +25 volts to signify a Space (Logic 0) and -3 to -25 volts for a Mark (logic 1). Any voltage in between these regions (ie between +3 and -3 Volts) is undefined. Therefore, this signal is put through a RS-232 Level Converter. This is the signal present on the RS-232 Port of your computer, shown below.

The above waveform applies to the Transmit and Receive lines on the RS-232 port. These lines carry serial data, hence the name Serial Port. There are other lines on the RS-232 port which, in essence are Parallel lines. These lines (RTS, CTS, DCD, DSR, DTR, RTS and RI) are also at RS-232 Logic Levels. Almost all digital devices which we use require either TTL or CMOS logic levels. Therefore the first step to connecting a device to the RS-232 port is to transform the RS-232 levels back into 0 and 5 Volts. This is done by RS-232 Level Converters. Two common RS-232 Level Converters are the 1488 RS-232 Driver and the 1489 RS-232 Receiver.

UART (Universal Asynchronize Receiver/Transmitter) REGISTERS: The UART is an RS232 I/O chip. The UART converts the parallel data coming from CPU into serial form to make it compatible to RS232 cable. Similarly, it converts the serial data coming from RS232 cable into parallel form to make it compatible to the computer. In this study, we are to set the registers of UART to set the

Table 1: The registers used in the present study is given below:

Base address	DLAB	Read/Write	Abr.	Register name
+ 0	=0	Write	-	Transmitter holding buffer
	=0	Read	-	Receiver buffer
	=1	Read/Write	-	Divisor latch low byte
+ 1	=0	Read/Write	IER	Interrupt enable register
	=1	Read/Write	-	Divisor latch high byte
+ 2	-	Read	IIR	Interrupt Identification register
	-	Write	FCR	FIFO control register
+ 3	-	Read/Write	LCR	Line control register
+ 4	-	Read/Write	MCR	Modem control register
+ 5	-	Read	LSR	Line status register
+ 6	-	Read	MSR	Modem status register
+ 7	-	Read/Write	-	Scratch register

properties like baud rate, parity etc. A Table 1 of UART registers.

When DLAB is set to '0' or '1' some of the registers change. This is how the UART is able to have 12 registers (including the scratch register) through only 8 port addresses. DLAB stands for Divisor Latch Access Bit. When DLAB is set to '1' via the line control register, two registers become available from which you can set your speed of communications measured in bits per second. The UART is fitted with a Programmable Baud Rate Generator which is controlled by these two registers. Lets say for example we only wanted to communicate at 2400 BPS. We worked out that we would have to divide 115,200 by 48 to get a workable 2400 Hertz Clock. The Divisor, in this study 48, is stored in the two registers controlled by the Divisor Latch Access Bit. This divisor can be any number which can be stored in 16 bits (ie 0 to 65535). The UART only has a 8 bit data bus, thus this is where the two registers are used. The first register (Base +0) when DLAB = 1 stores the Divisor latch low byte where as the second register (base +1 when DLAB =1) stores the Divisor latch high byte.

Table 2 is some more common speeds and their divisor latch high bytes and low bytes. Note that all the divisors are shown in Hexadecimal.

The UART has a 8-bit data bus. An 8-bit data when stored in a UART register will make various changes corresponding value of each bit which is either ‘1’ or ‘0’. The various functions of the UART registers, used in the interfacing, depending upon state of bits are:

Table 2:The commonly used baudrate divisors

Speed (BPS)	Divisor (Dec)	Divisor latch high byte	Divisor latch low byte
50	2304	09h	00h
300	384	01h	80h
600	192	00h	C0h
2400	48	00h	30h
4800	24	00h	18h
9600	12	00h	0Ch
19200	6	00h	06h
38400	3	00h	03h
57600	2	00h	02h
115200	1	00h	01h

Interrupt Identification Register (IIR):

Bit			Notes
Bits 6 and 7	Bit 6	Bit 7	
	0	0	No FIFO
	0	1	FIFO Enabled but unusable
	1	1	FIFO Enabled
Bit 5			64 Byte Fifo Enabled (16750 only)
Bit 4			Reserved
Bit 3	0		Reserved on 8250, 16450
	1		16550 Time-out Interrupt Pending
Bits 1 and 2	Bit 2	Bit 1	
	0	0	Modem status interrupt
	0	1	Transmitter holding register empty
Interrupt			
	1	0	Received data available interrupt
	1	1	Receiver line status interrupt
Bit 0	0		Interrupt pending
	1		No interrupt pending

First In / First Out Control Register (FCR):

Bit			Notes
Bits 6 and 7	Bit 7	Bit 6	Interrupt trigger level
	0	0	1 Byte
	0	1	4 Bytes
	1	0	8 Bytes
	1	1	14 Bytes
Bit 5			Enable 64 Byte FIFO (16750 only)
Bit 4			Reserved
Bit 3			DMA mode select. Change status
of			RXRDY and TXRDY
pins from			mode 1 to mode 2.
Bit 2			Clear transmit FIFO
Bit 1			Clear receive FIFO
Bit 0			Enable FIFO's

Modem Control Register (MCR):

Bit	Notes
Bit 7	Reserved
Bit 6	Reserved
Bit 5	Autoflow control enabled (16750
only)	
Bit 4	Loop back mode
Bit 3	Aux output 2
Bit 2	Aux output 1
Bit 1	Force request to send
Bit 0	Force data terminal ready

Line Control Register (LCR):

Bit 7	1	Divisor latch access bit		
	0			Access to receiver buffer, transmitter buffer and Interrupt enable register
Bit 6				Set break enable
Bits 3, 4 and 5	Bit 5	Bit 4	Bit 3	Parity select
	X	X	0	No parity
	0	0	1	Odd parity
	0	1	1	Even parity
	1	0	1	High parity (Sticky)
	1	1	1	Low parity (Sticky)
Bit 2				Length of stop Bit
	0			One stop bit
	1			2 Stop bits for words of length 6, 7 or 8 bits or 1.5 stop bits for word lengths of 5 bits.
Bits 0 and 1	Bit 1	Bit 0		Word length
	0	0		5 Bits
	0	1		6 Bits
	1	0		7 Bits
	1	1		8 Bits

PROGRAM DEVELOPMENT

In the process of program development, first of all we will locate the base "address" of the Port I/O so that we can communicate with the UART chip directly. For a typical PC system the following are standard port addresses (Table 3).

While programming there are two methods available to us. In first method, the UART can be polled to see if any new data is available. In second method, we can set up an interrupt handler to remove data from UART when it generates an interrupt. Polling the UART is a lot slower method, which is very CPU intensive thus can only have a maximum speed of around 34.8 KBPS before you start losing data. The other option is interrupt handler that will easily support 115.2 KBPS, even on low end computers. In this study we will discuss only interrupt handler method. In polling method modifications only to the registers of the UART are required to achieve the desirable settings for communication. In interrupt driven method, in addition to modifications to UART's registers we are also required to change the settings of PIC (Programmable Interrupt Controller). For using interrupts we must know the IRQ of the communication port. Once we know the IRQ the next step is to find it's interrupt vector or software interrupt as some people may call it. Basically any 8086 processor has a set of 256 interrupt vectors numbered 0 to 255. Each of these vectors contains a 4 byte code which is an address of the Interrupt Service Routine (ISR). Fortunately C being a high level language, takes care of the addresses for us. All we have to know is the actual interrupt vector. The interrupt vectors for serial ports are given in the Table 4 only the interrupts which are associated with IRQ's.

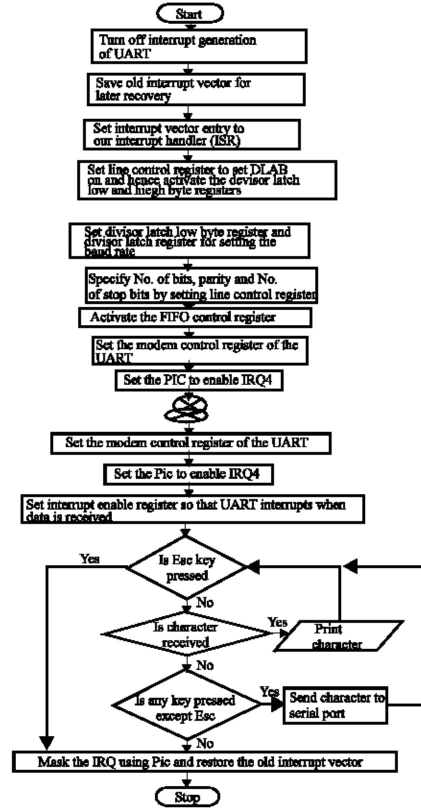
The other 240 are of no interest to us when programming RS-232 type communications. For example if we were using COM1 which has a IRQ of 4, then the interrupt vector would be 0C in hex. Using C we would set up the vector using the predefined function in C, that is setvect(0x0C,void interrupt(*isr)()); where second parameter of function would lead us to a set of instructions which would service the interrupt. Hence the second parameter is our interrupt handler called an Interrupt Service Routine. Anything can be put in this service routine.

Table 3: The standard port addresses

Name	Address	IRQ
COM 1	3F8	4
COM 2	2F8	3
COM 3	3E8	4
COM 4	2E8	3

Table 4: The interrupt vectors (Hardware only)

INT (Hex)	IRQ	Common uses
0B	3	Serial Comms. COM2/COM4
0C	4	Serial Comms. COM1/COM3



Flowchart for interrupt driven communication program for the wireless modem: In the above flowchart, we have modified the UART registers and also set the PIC. Data from the computer is send using RS232 cable is sent bits string to the IC 75189 (Level converter of wireless modem) using the program developed. The output of RS232 cable is checked using CRO. A shifting voltage level was observed as per the requirement. The listing of the computer program is shown in Appendix A.

CONCLUSION

A flowchart for interfacing pc with the transmitter section of wireless modem using rs232 cable was developed. The software program was developed by modifying the UART's registers. The software developed during the present was tested by sending data from the pc and receiving at the end of rs232 cable. A shift in voltage level was obtained.

Appendix A
Listing of computer program:

```
#include <dos.h>
#include <stdio.h>
#include <conio.h>

#define PORT1 0x3F8  /* Port Address Goes Here */
#define INTVECT 0x0C /* Com Port's IRQ here (Must also change PIC setting) */

/* Defines Serial Ports Base Address */
  /* COM1 0x3F8                         */
  /* COM2 0x2F8                         */
  /* COM3 0x3E8                         */
  /* COM4 0x2E8                         */
int bufferin = 0;
int bufferout = 0;
char ch;
char buffer[1025];
void interrupt (*oldport1isr)();
void interrupt PORT1INT()  /* Interrupt Service Routine (ISR) for PORT1 */
{
 int c;
 do { c = inportb(PORT1 + 5);
        if (c and 1) {buffer[bufferin] = inportb(PORT1);
                                          bufferin++;
                                          if (bufferin == 1024) {bufferin = 0;}}
   }while (c and 1);
 outportb(0x20,0x20);
}
void main(void)
{
 int c;
outportb(PORT1 + 1, 0);        /* Turn off interrupts - Port1 */

oldport1isr = getvect(INTVECT); /* Save old Interrupt Vector of later
                                                        recovery */
setvect(INTVECT, PORT1INT);     /* Set Interrupt Vector Entry */
                                /* COM1 - 0x0C */
                                /* COM2 - 0x0B */
                                /* COM3 - 0x0C */
                                /* COM4 - 0x0B */
/*      PORT 1  Settings        */

outportb(PORT1  +  3,  0x80);   /*  SET  DLAB  ON  */
outportb(PORT1 + 0, 0x0C);  /* Set Baud  rate  -  Divisor  Latch  Low  Byte  */
                                /* Default 0x03 =  38,400 BPS */
                                /*      0x01 = 115,200 BPS */
                                /*      0x02 =  57,600 BPS */
                                /*      0x06 =  19,200 BPS */
                                /*      0x0C =   9,600 BPS */
                                /*      0x18 =   4,800 BPS */
                                /*      0x30 =   2,400 BPS */
outportb  (PORT1  + 1,  0x00);   /* Set Baud rate - Divisor Latch High Byte */
```

```
outportb  (PORT1  + 3,  0x03);  /*  8 Bits, No Parity, 1 Stop Bit */
outportb  (PORT1  + 2,  0xC7); /*   FIFO   Control   Register  */
outportb(PORT1 + 4, 0x0B);  /* Turn on DTR, RTS and OUT2 */

outportb(0x21,(inportb(0x21) and 0xEF));  /* Set Programmable Interrupt Controller */
                                          /* COM1 (IRQ4) - 0xEF  */
                                          /* COM2 (IRQ3) - 0xF7  */
                                          /* COM3 (IRQ4) - 0xEF  */
                                          /* COM4 (IRQ3) - 0xF7  */
outportb(PORT1 + 1 , 0x01);  /* Interrupt when data received */
printf("\nSample Comm's Program. Press ESC to quit \n");
 do {
    if (bufferin != bufferout){ch = buffer[bufferout];
                                      bufferout++;
                                      if (bufferout == 1024) {bufferout = 0;}
                                      printf("%c",ch);}
    if (kbhit()){c = getch();
                                       outportb(PORT1, c);}
   } while (c !=27);
 outportb(PORT1 + 1, 0);       /* Turn off interrupts - Port1 */
 outportb(0x21,(inportb(0x21) | 0x10));  /* MASK IRQ using PIC */
                                          /* COM1 (IRQ4) - 0x10  */
                                          /* COM2 (IRQ3) - 0x08  */
                                          /* COM3 (IRQ4) - 0x10  */
                                          /* COM4 (IRQ3) - 0x08  */
 setvect(INTVECT, oldport1isr); /* Restore old interrupt vector */
}
```

REFRENCES

1. Peter Davis, 1990. Wireless local area network technologies, mcgrawhill Publication.
2. Andrew S. Tanenbaum, 2001. Computer network, PHI India LTD.
3. http://standertsieee.org
4. Rappaport, T.S., 1996. Wireless communication principle and practices, Upper saddle river, NJ, USA.
5. Ankur Singh, 2006. Data transmission wireless modem for information connectivity (presented) at national level technical symposium, DAV institute of engineering and technology, Jalandher.