Catching Analogue data into
PSION PALMTOPS AND PCs

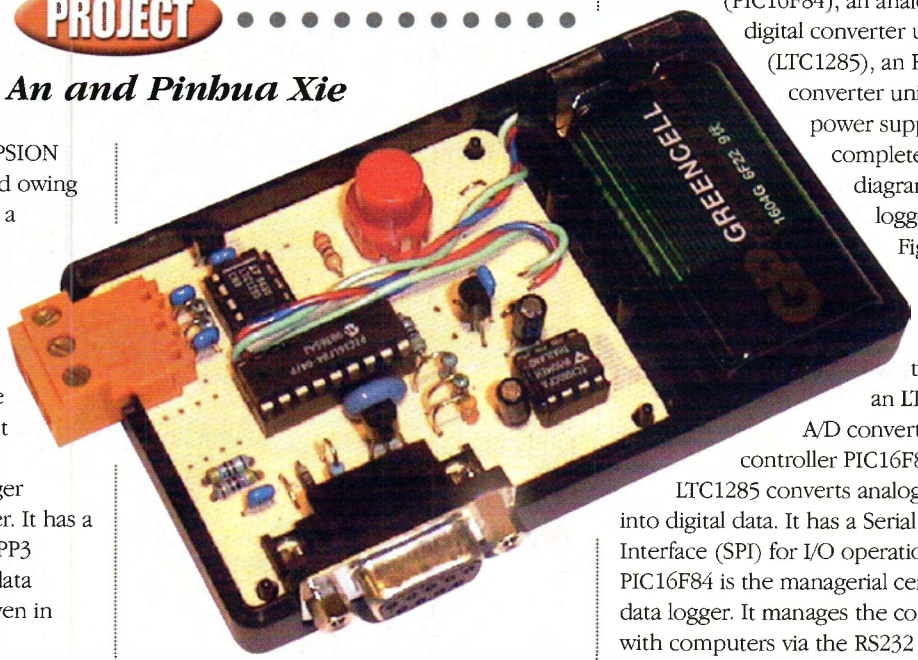
PROJECT

By Pei An and Pinhua Xie

It is very easy to interface the PSION palmtops to the external world owing to the fact that the PSION has a standard RS232 port and it has a powerful built-in programming language.

This article describes a serial port data logger that allows a PSION to catch analogue data. The data logger has one analogue input channel (0-2.5V with a 12-bit A/D conversion accuracy. The data logger utilizes a PIC16F84 micro-controller. It has a small size and is powered by a 9V PP3 battery. A schematic showing the data logger connected to a PSION is given in Figure 1.

The data logger can be also used with IBM PCs or other palm-top organisers having a standard RS232 port.



Operations of the data logger

The data logger is connected to the RS232 port of a host computer via a serial cable. After the data logger is powered on or reset, it enters a waiting state during which it waits for the computer to issue a command byte (=15h).

After the computer sends the byte to the data logger through the TX line of the RS232 port, the logger accepts the command and carries out an A/D conversion. After a conversion result is available, the logger transmits the conversion results back to the computer in two byte transmissions via RX line of the RS232 port. The first byte is the upper 4 bits of the conversion result and the second byte is the lower 8 bits. After the two bytes are transmitted, the data logger flashes its LED once to indicate that a conversion and data transfer cycle is completed. Next, the data logger goes back to the waiting state again.

The RS232 data format is as follows: 9600 Baud rate, 8 data bits, 1 stop bit and no parity check bit.

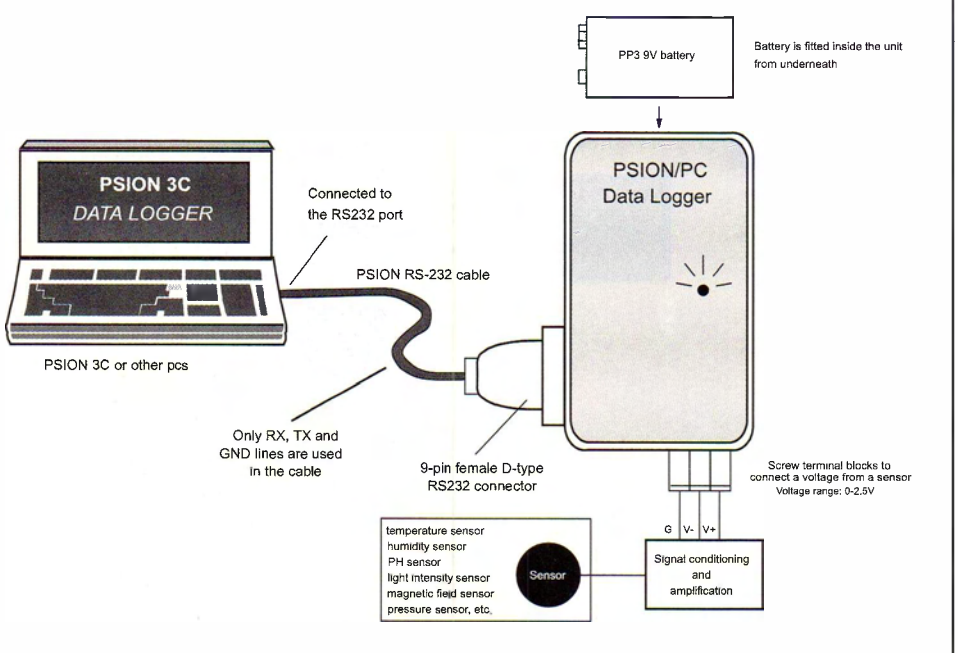


Figure 1. PIC based Psion/PC single channel data logger system

Hardware of the logger

The block diagram of the data logger is given in Figure 2. The system contains 4 units. They are a central control unit (PIC16F84), an analogue to digital converter unit (LTC1285), an RS232/TTL converter unit and a power supply unit. The complete circuit diagram of the data logger is given in Figure 3. The system utilises only two key ICs: an LTC1285CN8 A/D converter and a PIC controller PIC16F84. The LTC1285 converts analogue signals into digital data. It has a Serial Peripheral Interface (SPI) for I/O operations. The PIC16F84 is the managerial centre of the data logger. It manages the communication with computers via the RS232 port and it controls the operation of the A/D converter.

PIC16F84

The central control unit is based on a Microchip PIC16F84 peripheral interface controller. The 16F84 has an EEPROM memory (electrically erasable memory) to store the program. This makes it particularly useful for system development. This is the reason why it is adopted for this application.

The pin-out, the internal block diagram and the organization of file registers of the PIC16F84 is shown in Figure 4. Pin 14 and pin 5 are connected to the positive and negative rails of a power supply. The supply voltage range is 2 to 6 Volts. The power supply current is typically 2mA at 5V and 4 MHz clock frequency. This drops to several tens of mA when the IC is in standby mode. Pin 4 is the master clear. It must be high in normal operation. Pin 15 and 16 are connected to a crystal or ceramic resonator up to 4MHz.

The PIC16F84 contains the following main functional units:

- an 1kbyte 14-bit wide EEPROM to store instructions
- a 64 8-bit wide EEPROM to store data
- 15 (8-bit) special function hardware file registers (RTCC, OPTION, PCL, STATUS, FSR, PORTA, PORTB, TRISA, TRISB, EEDATA, EECON1, EEADR, EECON2, PCLATH, INTCON)
- 36 (8-bit) general purpose file registers (0C to 2F)
- an 8-bit accumulator, w

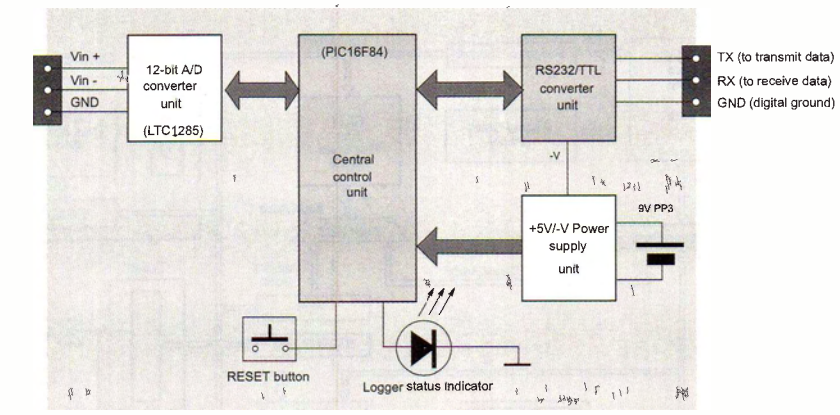


Figure 2. Block diagram of the data logger for Psion/PC computers

For PIC programming, there are only 35 single word instructions (assembly language), which makes the programming easy to learn. For more details of the hardware and the instruction set of the PIC16F84, please refer to the manufacture's data sheet (Reference 1).

For programmers who are very familiar with Basic or C language, Basic-like or C-like programming languages for the PIC can be used. With these high-level languages, many functions are built into the language. Also the compiler itself performs many low-level tasks like allocating memory spaces.

In the present circuit, the PIC works in the crystal oscillator mode. A 4MHz ceramic resonator is used (see Figure 3). I/O lines of the PIC16F84 are used as shown in Table 1.

- Port A (5 lines). Each line can be set as an input or output. RA4 is used also by timer/counter module. If configured as outputs, they sink 25mA and source 20mA
- Port B (8 lines). RB0 is also used as an interrupt input. If configured as outputs, they sink 25mA and source 20mA
- Four interrupt sources: External INT pin, RTCC timer overflow, PB4 to PB7 change status and data EEPROM write complete.
- 8-bit real time clock/counter with 8-bit programmable pre-scaler
- a watchdog timer

Table 1.

Port A

RA0 (pin 17)	serial data output from the PIC (connected to RX of the RS232 port)
RA1 (pin 18)	serial data input to the PIC (connected to TX of the RS232 port)
RA2 (pin 1)	RA3 (pin 2) and RA4 (pin 3): not used

Port B

RB0 (pin 6)	and RB1 (pin 7): not used
RB2 (pin 8)	control of the logger status LED (output)
RB5 (pin 11)	serial data output (Dout) from the LTC1285 (input)
RB6 (pin 12)	serial clock (CLK) of the LTC1285 (output)
RB7 (pin 13)	enable (-CS) of the LTC1285 (output)

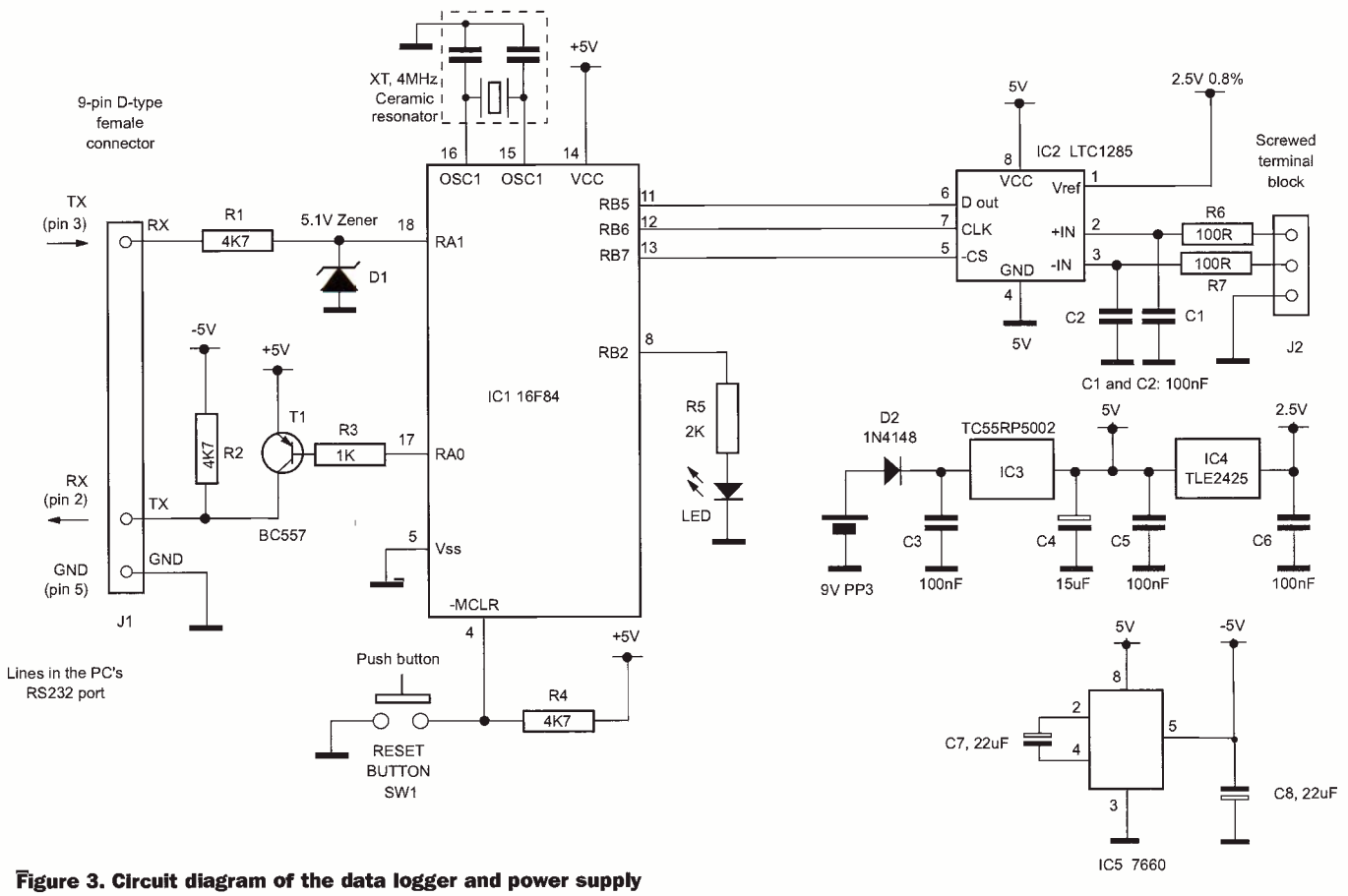


Figure 3. Circuit diagram of the data logger and power supply

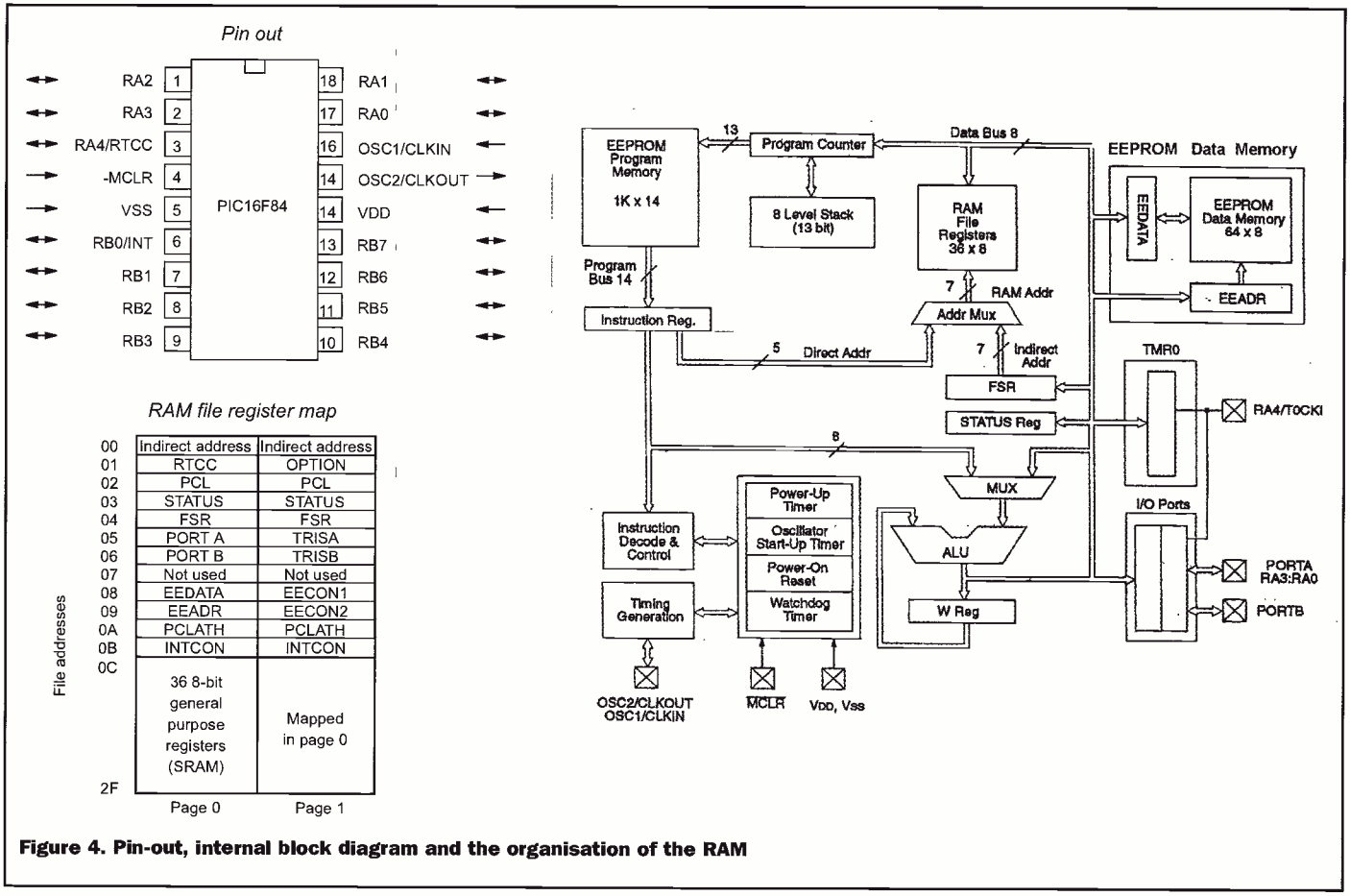


Figure 4. Pin-out, internal block diagram and the organisation of the RAM

Analogue to digital converter unit (LTC1285CN8)

The analogue to digital converter unit utilises an LTC1285CN8 12-bit successive approximation A/D converter. The pin-out and internal block diagram of the IC is shown in Figure 5. It requires a power supply 2.7V to 6V. Pin 8 and pin 4 are connected to the positive and negative rail of the power supply. Pin 4 is the reference voltage input. The typical supply current to the chip is 260mA at a sampling rate 6.6kHz with a power supply of 2.7V. When it is in standby mode, the supply current drops to several nanoamps. The LTC1285 has a differential analogue input (pin 2 and pin 3) and the analogue input leakage current is typically 1mA. For more details of the chip, please refer to the manufacture's data sheet (Reference 2).

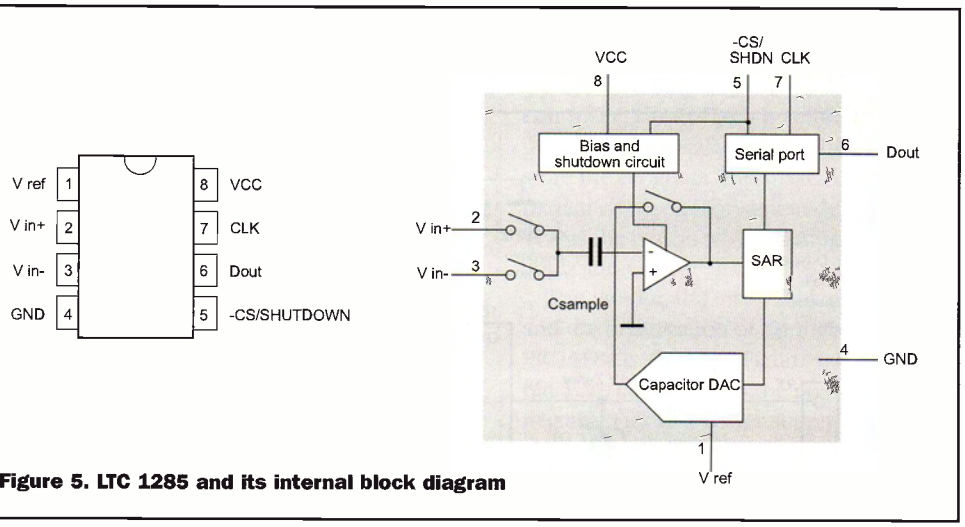


Figure 5. LTC 1285 and its internal block diagram

The LTC1285 communicates with other circuitry through a 3-wire SPI serial interface. These three wires are -CS/SHDN, CLK and Dout. -CS/SHDN (pin 1) low selects the chip and initiates data transfer. If the pin is at a high state, the converter is in the standby mode. CLK (pin 7) is the clock input. It synchronises the serial data transfer and determines conversion speed. At the falling edge of CLK, each bit of an A/D conversion result (12 bits) is sent out from Dout pin (pin 6).

The operating sequence of the LTC1285 is shown in Figure 6. Data transfer is initiated at the falling edge of the chip select -CS/SHDN (pin 1). After -CS falls the second CLK pulse enables Dout. A null bit (logic 0) appears on the Dout (pin 6) firstly. At the next 12 falling edge of the clock, the 12 bits of the A/D conversion result appear on Dout one by one.

In the present circuit, -CS, Dout and CLK are connected to the RB7, RB5 and RB6 of the PIC. The PIC sets RB7 (-CS) and RB6 (CLK) as output lines. RB5 is set as an input.

RS232/TTL translator unit

The function of this unit is to perform voltage conversions between RS232 and TTL levels. From the circuit diagram, we see that the RX line (the line from which the logger receives data, RS232 voltage level) is converted into a TTL voltage level using a simple voltage clamp circuit based on R1 and a Zener diode D1. This converter does not have an inverting action. A TX signal (output from the logger, RS232 voltage level) is generated by a circuit consisting of R2, R3 and T1. The circuit requires a positive and a negative power supply. The former is from the +5V power supply of the data logger board. An on-board voltage inverter, TC7660, generates the latter.

The pin-out the RS232 port on the PSION 3C is given in Figure 7. If the data logger is to be connected to the PSION computer, a cable as shown in Figure 8 is needed. You could make one yourself.

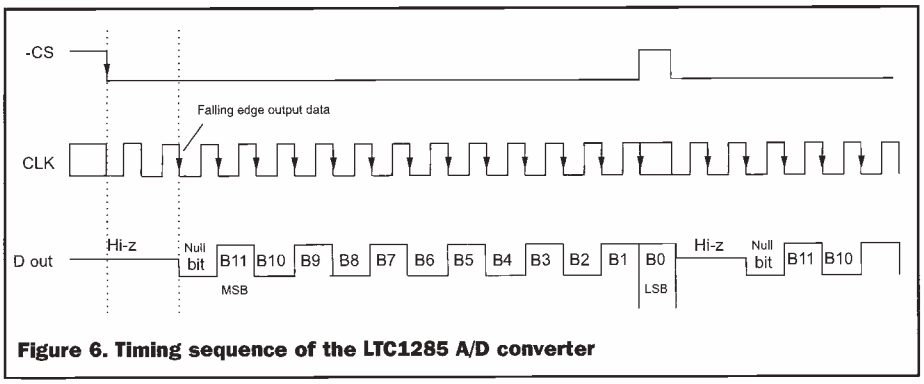


Figure 6. Timing sequence of the LTC1285 A/D converter

There are only three wires to connect. It is noted that the standard PSION 3C serial cable has a 9-pin D-type female connector at the end. The connector on the data logger is also a 9-pin D-type female connector. A gender converter should be used in this case to allow the cable to be connected to the data logger. Figure 8 also shows how this converter can be constructed. Care should be taken when making the cable and the gender converter to ensure that the transmit line from the PSION goes into the receive line of the data logger. The transmit line from the data logger should go into the receive line of the PSION computer.

The pin-out of the RS232 port on a PC is given in Figure 9. If the logger is to be connected to an IBM PC, a standard RS232 cable is used. The details of the RS232 port and how to use it can be found in Reference 3.

Power supply unit

The circuit of the power supply unit is given in Figure 3. The power supply is a PP3 9V battery. It is regulated into a +5V power supply using a TC55RP5002EZB regulator. The TC55RP is a 5V fixed voltage regulator with a maximum supply current 30mA. It offers a very low dropout voltage of 100mV and a quiescent current of 3.5mA. The +5V supply is converted into +2.5V by the TLE2425 2.5V voltage reference IC. The 2.5V reference voltage is used by the A/D converter. The TC7660 converts the +5V voltage into a -5V voltage.

Construction of the logger

The data logger is constructed on a single-sided PCB board and is housed in a slim size box. Figure 10 gives the component layout and Figure 11 shows the assembly of the logger inside the box.

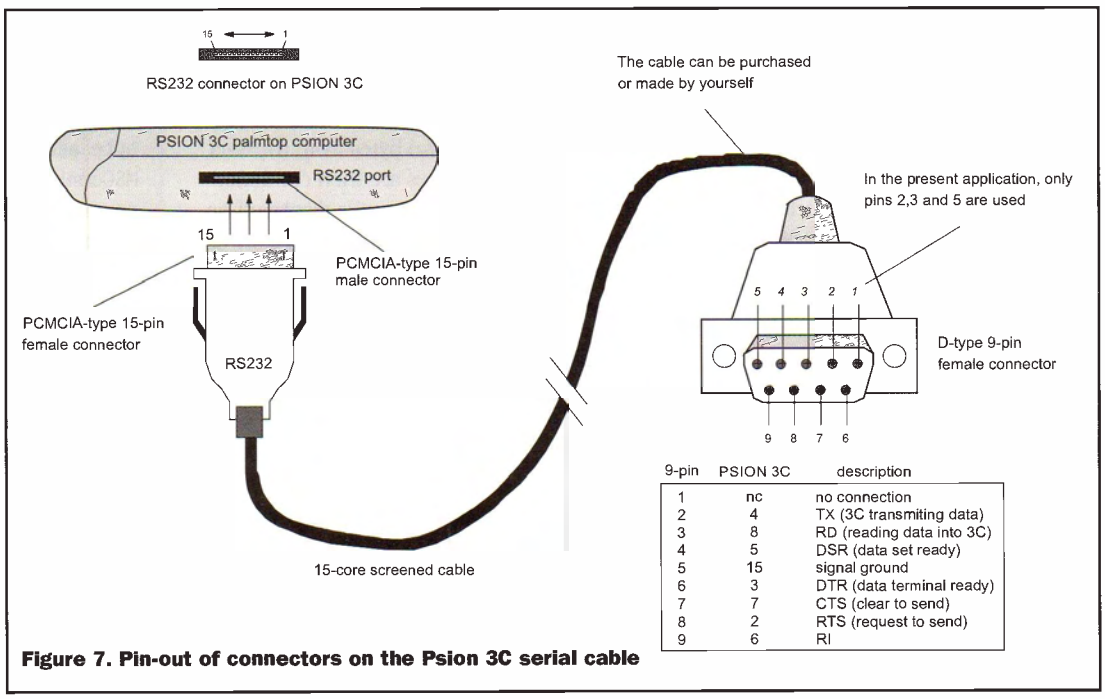
PIC software development

After pressing the reset button or the power to the data logger being turned on, the data logger enters the waiting procedure, during which it constantly monitors the serial data input line of the RS232 port through PA1 (pin 18).

Once a command byte 15h is received, the data logger begins the A/D conversion procedure. The A/D converter is activated and the A/D conversion result is loaded into the PIC. Next, the PIC transmits the A/D conversion result back to the host computer in two separate byte transmissions. The two bytes represent a 12-bit A/D conversion data. The upper 4 bits are sent first and the lower 8 bits are sent next. The PIC sends out the bytes serially via RA0 (pin 17) and communication between the host computer and the PIC has no handshakes.

The PIC software can be written in assembly language and in Basic-like or C-like languages. The present program is written in BASIC PRO language from Micro Engineering Lab (References 4 and 5). The complete PIC software is written using PIC BASIC Pro and is given in Program List 1.

The PIC BASIC compiler is a programming language that makes it very easy and quicker



9-pin	PSION 3C	description
1	nc	no connection
2	4	TX (3C transmiting data)
3	8	RD (reading data into 3C)
4	5	DSR (data set ready)
5	15	signal ground
6	3	DTR (data terminal ready)
7	7	CTS (clear to send)
8	2	RTS (request to send)
9	6	RI

Figure 7. Pin-out of connectors on the Psion 3C serial cable

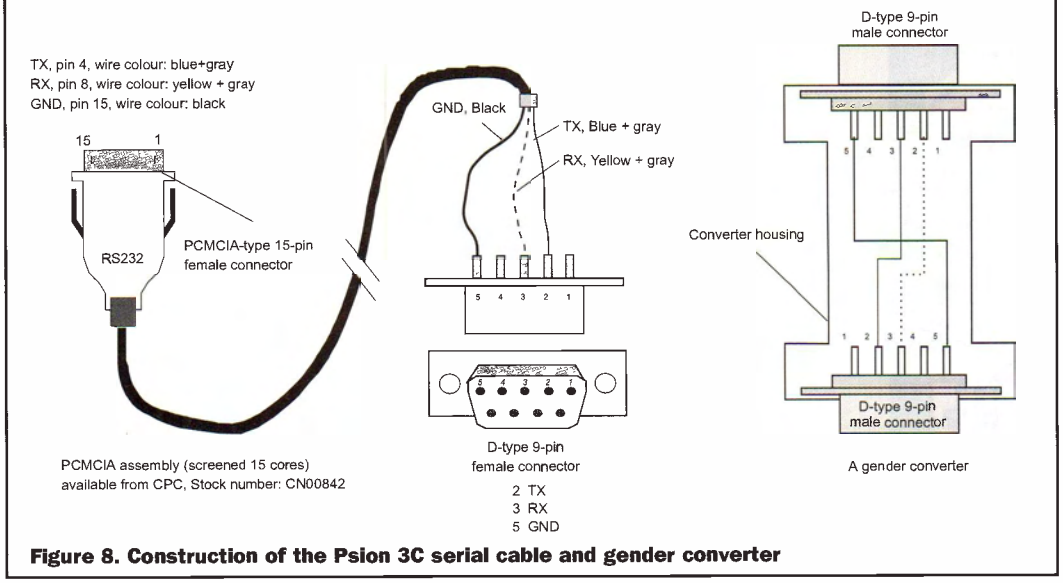
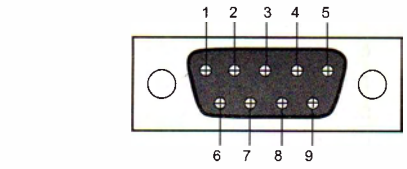
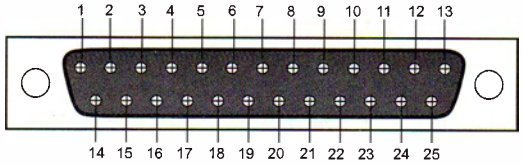


Figure 8. Construction of the Psion 3C serial cable and gender converter



(a) 9-pin male socket viewed from the back of the computer



(b) 25-pin male socket viewed from the back of the computer

Pin functions of the RS232 connectors

25 PIN	9 PIN	NAME	DIRECTION (FOR PCS)	DESCRIPTION
1		*Prot*	-	Protective ground
2	3	*TD*	OUTPUT	Transmit data
3	2	*RD*	INPUT	Receive data
4	7	*RTS*	OUTPUT	Request to send
5	8	*CTS*	INPUT	Clear to send
6	6	*DSR*	INPUT	Data set ready
7	5	*GND*	-	Signal ground (common)
8	1	*DCD*	INPUT	Data carrier detedt
20	4	*DTR*	OUTPUT	Data terminal ready
22	9	*RI*	INPUT	Ring indicator
23		*DSRD*	I/O	Data signal rate detector

Figure 9. Pin-out of the RS232 port on PCs

for you to program a wide range of powerful Microchip's PIC micro-controllers. The Basic-like structure is much easier to read and write than the assembly language. It is a viable alternative to assembly language. The PIC BASIC compiler gives you direct access to all the PIC micro-controllers' registers (I/O ports, A/D converters, Hardware serial ports etc.). It automatically takes care of memory allocation and page boundaries and memory banks. It also provides many built-in commands to do various things that will save a programmer many hours to develop in the assembly language.

PSION link software

The demonstration OPL program is very simple. Firstly, it sends a command byte 15h to the data logger. After this it reads two bytes from the data logger. The two bytes are then combined into a voltage. The voltage is then displayed on the screen. There are plenty of explanations given in the Program List 2. The use of the OPL programming language can be found in Reference 6.

TP6 PC link software

The program for PCs is written in Turbo Pascal 6 for DOS (Program List 3). Firstly, it sends a command byte 15h to the data logger. After this it reads two bytes from the data logger. The two bytes are then combined into a voltage. The voltage is displayed on the screen. The details of how PC software controls the RS232 port can be found in Reference 7.

VB5 PC link software (Reference 8)

Details of programming the RS232 port on PCs using Visual Basic programming language can be found in Reference 8. The program source code in VB5 is given in Program List 4. Some of the commands used in the present project are explained below:

A PC may have a number of COM ports. Each port has a logic name COM1, COM2 or COM3. To select a port, use the following command:

```
MSComm1.CommPort = 4 'here COM4 is
selected
```

Then the port is configured to treat the input data byte as binary and the port is opened:

```
MSComm1.InputMode =
comInputModeBinary
MSComm1.PortOpen = True
```

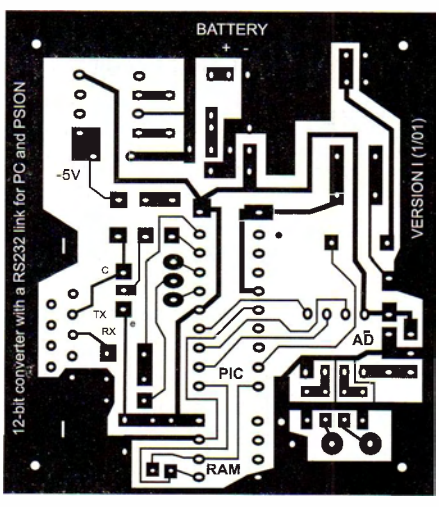
To output data from the COM port, use the following command:

```
MSComm1.Output = Chr$(1 * 16 + 5)
'15H is output from the COM port
```

Finally to read data from the COM port, the following commands are used:

```
MSComm1.InputLen = 1        'When
reading data from the COM, read one
byte each time
MSComm1.InBufferCount = 0   'The
count of byte in the input buffer is
cleared = 0

Do
    DoEvents
Loop Until MSComm1.InBufferCount = 2
'Check number of bytes in the input
buffer
'If it is 2, it means two bytes are
received
volt = (AscB(MSComm1.Input) * 256 +
```



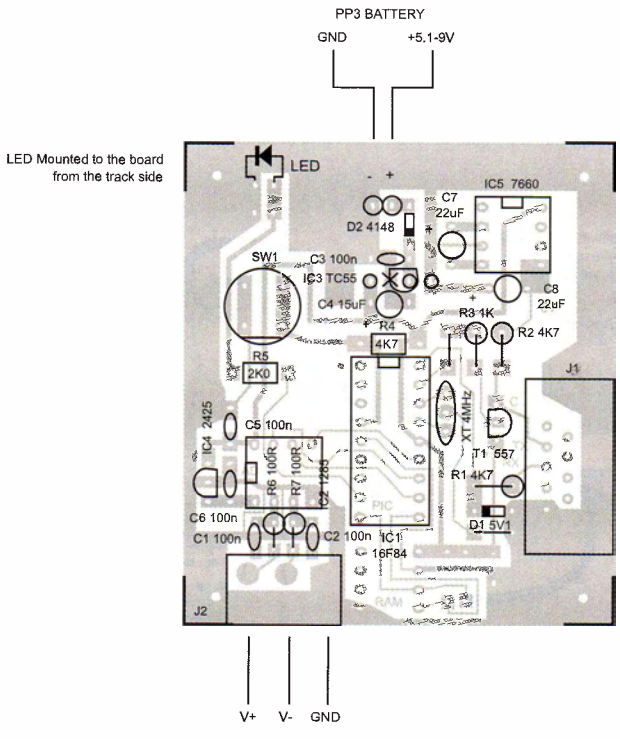


Figure 10. Pin-out of the RS232 port on PCs

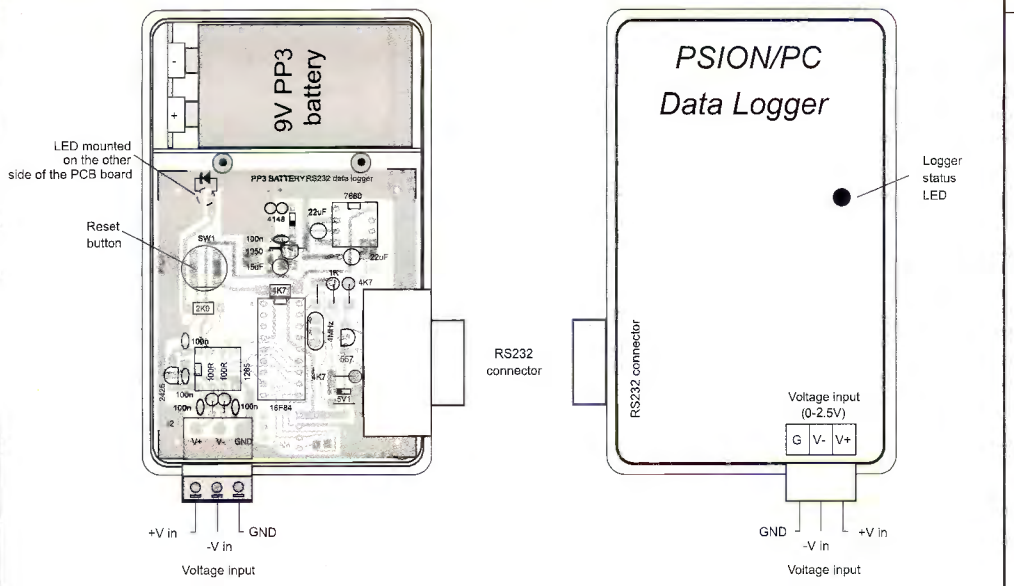


Figure 11. Assembly of the data logger

```
AscB(MSComm1.Input)) / 4096 * 2.5
'The first MSComm1.input reads the
'1st byte and the second
'MSComm1.input reads the 2nd byte.
```

Technical support

Kits including all necessary components (pre-programmed PIC) to construct a complete data logger are available from the authors. Please make your enquiry to Dr. Pei An Tel/Fax/Answer: +44-(0)161-477-9583, e-mail: pan@intec-group.co.uk

References

1. PIC16F84 data sheet, Microchip Technology Incorporated **<www.microchip.com>**
2. LTC1285 data sheet, Linear Technology Incorporated **<www.linear-tech.com>**
3. PC Interfacing - Using Centronic, RS232 and game ports, Pei An, Newnes, Butterworth-Heinemann, 1998, ISBN0240514483 **<www.intec-group.co.uk>**
4. User manual of PicBasic Pro compiler **<www.picbasic.co.uk>**
5. Experimenting with the PICBASIC PRO compiler, Les Johnson, A Crownhill Publication **<www.crownhill.co.uk>**
6. Programming manual for PSION series 3 **<www.psion.com>**
7. Real-world Programming with Visual Basic, Anthony T. Mann, SAMS publishing ISBN 0-672-30619-0
8. PC interfacing, communications and windows programming, William Buchanan, Addison-Wesley, ISBN 0-201-17818-4

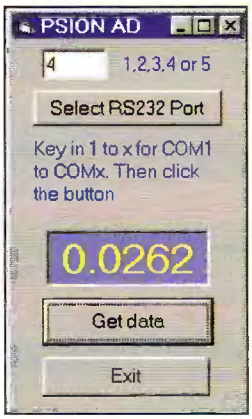


Figure 12. Screen dump of the VB5 driver

PARTS LIST

Resistors

0.25W, 1% metal film resistors

R1	4K7
R2	4K7
R3	1K0
R4	4K7
R5	2K
R6,R7	100R

Capacitors

C1,2,3, 5,6	100nF ceramic disc
C4	15uF electrolytic capacitor
C7,8	22uF electrolytic capacitor

Semiconductors

IC1	PIC16F84 micro controller
IC2	LTC1285 A/D converter
IC3	TC55RP5002EZB +5V low power voltage regulator
IC4	TLE2425 2.5 voltage reference
IC5	7660 voltage inverter
D1	5V1 zener diode
D2	1N4148
LED	3mm diameter, 1mA low current LED
T1	BC557 pnp transistor

Others

J1	9 pin D-type female connector
J2	3-way detachable screwed terminals
SW1	push button switch
XT	4MHz ceramic resonator (3 pin device)

PCB board

9V PP3 battery

Program List-1 (PIC program in PicBasic)

```
TX        VAR PORTA.0        'define line for transmit
RX        VAR PORTA.1        'define line for receiver
LED       VAR PORTB.2        'define line for LED

CS        VAR PORTB.7        'define lines for SPI bus of the A/D converter
CLK       VAR PORTB.6
DOUT      VAR PORTB.5

datafpc   VAR BYTE  'define variables to be used
datatpc   VAR BYTE
j         VAR BYTE
datal     VAR BYTE
datah     VAR BYTE

'*************START OF PROGRAM*************
DEFINE OSC 4
LOW LED
LOW CLK
HIGH CS
INPUT DOUT
GOTO START

'****FLASH LED SUB
FLASH:    HIGH LED
          PAUSE 100
          LOW LED
          RETURN

'****RECEIVE DATA FROM PC
RX_PC:
          SERIN2 RX,16468,[datafpc]     '9600, no RS232 driver
          RETURN

'****TRANSMIT DATA TO PC
TX_PC:
          SEROUT2 TX,84,[datatpc]                '9600, no RS232 driver
          RETURN

CLKPULSE:
          @ BSF PORTB.6
          PAUSEUS 10
          @ BCF PORTB.6
          PAUSEUS 10
          RETURN

'****A/D CONVERTER
AD:       @ BCF PORTB.7       'LOW CS
          PAUSEUS 10
          GOSUB CLKPULSE : GOSUB CLKPULSE
          GOSUB CLKPULSE : datah.3=DOUT
          GOSUB CLKPULSE : datah.2=DOUT
          GOSUB CLKPULSE : datah.1=DOUT
          GOSUB CLKPULSE : datah.0=DOUT
          GOSUB CLKPULSE : datal.7=DOUT
          GOSUB CLKPULSE : datal.6=DOUT
          GOSUB CLKPULSE : datal.5=DOUT
          GOSUB CLKPULSE : datal.4=DOUT
          GOSUB CLKPULSE : datal.3=DOUT
          GOSUB CLKPULSE : datal.2=DOUT
          GOSUB CLKPULSE : datal.1=DOUT
          GOSUB CLKPULSE : datal.0=DOUT
          @ BSF PORTB.7       'HIGH CS
          datah.7=0
          datah.6=0
          datah.5=0
          datah.4=0
          RETURN

'****TRANSMIT A/D RESULT BACK TO PC
TXAD:
          GOSUB AD
          datatpc=datah
          GOSUB TX_PC
          GOSUB FLASH
          datatpc=datal
          GOSUB TX_PC
          RETURN

'**************MAIN PROGRAM****************
START:    PAUSE 100
          GOSUB FLASH
ST1:
          GOSUB RX_PC
          IF datafpc=$15 THEN GOSUB TXAD
          GOTO ST1
```

Program List-2 (PSION 3C data logging program)

```
PROC RS232logger:
REM PSION 3C OPL test program for RS232 data logger
REM COPYRIGHT to Pei AN, 20/4/98
REM handshake setting: None

REM define local variables to be used in the program

        LOCAL term&,baud%, parity%, data%
        LOCAL stop%, hand%, frame%,srchar%(6)
        LOCAL d1%,d2%, dummy%,err%,ret%,len%
        LOCAL voltage

REM open the RS232 channel
        LOPEN "TTY:a"

REM dedine RS232 configuration parameters
REM RS232 setting: 9600, 8 bit, no parity, 1 stop, no handshake
        baud%=15 : parity%=0 : data%=8
        stop%=1 : hand%=4   : term&=&04002400
        frame%=data%-5
        IF stop%=2 : frame%=frame% OR 16 : ENDIF
        IF parity% : frame%=frame% OR 32 : ENDIF
        Srchar%(1)=baud% OR (baud% * 256)
        Srchar%(2)=frame% OR (parity% * 256)
        Srchar%(3)=(hand% AND 255) OR $1100
        Srchar%(4)=$13

REM config the RS232 port
        err%=IOW(-1,7,srchar%(1), dummy%)

REM PSION output a command to data logger
REM PSION then inputs two data bytes

        CLS
        FONT 8,8
        STYLE 16

        AT 15,1
        PRINT"PSION   DATA   LOGGER"

DO
        LPRINT CHR$(21);          REM output a command byte
        len%=1                    REM number of data read =1
        ret%=IOW(-1,1,d1%,len%)  REM input upper 4 bits from logger
        ret%=IOW(-1,1,d2%,len%)  REM input lower 8 bits from logger
        voltage=((d1% AND 15)*256+d2%)*2.5/4096  REM construct conversion data
        AT 13,3
        PRINT "INPUT  VOLTAGE  [V]: ";FIX$(voltage,3,6)
        pause 2                   REM a short pause
        UNTIL KEY$=CHR$(13)
ENDP
```

Program List 3 (TP6 data logger program)

```
Program RS232_data_logger;
{Software driver for PSION/PC RS232 data logger}
{The COM port is configured as Baud rate: 9600/4800/2400/1200
 Data bit length:8;  Parity Check: None; Stop Bit: 1  }
{Copyright to Pei An and Pinhua Xie, 20/4/98}

uses
    dos,crt,graph;
var
    RS232_address,com_number,number_of_COM,code:integer;
    dummy:byte;
    loggerdata: real;


Procedure detect_RS232;
{Universal auto detection of COM base address. User section of RS232 port}
{ $0000:$0400 holds the printer base address for COM1
  $0000:$0402 holds the printer base address for COM2
  $0000:$0404 holds the printer base address for COM3
  $0000:$0406 holds the printer base address for COM4
  $0000:$0411 number of parallel interfaces in binary format}
var
   COM:array[1..4] of integer;
   kbchar:char;
begin
      clrscr;
      COM_number:=1; {defaut printer}
      number_of_COM:=mem[$0000:$0411]; {read number of parallel ports}
      number_of_COM:=(number_of_COM and (8+4+2)) shr 1;
      COM[1]:=memw[$0000:$0400];       {Memory read procedure}
      COM[2]:=memw[$0000:$0402];
      COM[3]:=memw[$0000:$0404];
      COM[4]:=memw[$0000:$0406];
      textbackground(blue); clrscr;
      textcolor(yellow); textbackground(red); window(10,22,70,24); clrscr;
      writeln('Number of COM installed    :  ',number_of_COM:2);
      writeln('Addresses for COM1 to COM4:  ',COM[1]:3,'   ', COM[2]:3,'   ', COM[3]:3,'   ',
COM[4]:3);
      write('Select COM to be used (1,2,3,4)    :  ');
      delay(1000);
      if number_of_COM>1 then begin   {select COM1 through COM4 if more than 1 LPT installed}
        repeat
            kbchar:=readkey;                {read input key}
            val(kbchar, COM_number, code);   {change character to value}
        until (COM_number>=1) and (COM_number<=4) and (COM[COM_number]<>0);
                              end;
      clrscr;
      RS232_address:=COM[COM_number];
      writeln('Your selected RS232 interface:  COM',COM_number:1);
      write('RS232  Address                :  ',RS232_address:4);
      delay(5000);
      textbackground(black); window(1,1,80,25); clrscr;
end;


Procedure Write_interrupt_enable(RS232_address, Output_byte: integer);
{to enable interrupt indentification register on certain conditions
 output_byte=1, to generate an interrupt flag when a valid serial data is received}
begin
     Port[RS232_address+1]:=Output_byte;
end;


Function Read_interrupt_identification(RS232_address:integer):integer;
{to read interrupt identification register to check if an interrupt is pending}
begin
      Read_interrupt_identification:=Port[RS232_address+2]
end;


Procedure initialize;
{initialize COM to be 9600, no-parity check, 8 bit data and 1 stop bit}
{initialization is made by INTR($14)  dx=COM_number-1 for COM_number
 Bit functions of al register:
     Bit 7,6,5: set Baud rate       111=9600
     Bit 4,3  : set parity          00 =none parity
     Bit 2    : set stop bit        0=1 stop bit
     Bit 1, 0 : set data bit number 11=8 bit}
var
   register:registers;
begin
     with register do begin
          ah:=0;    al:=128+64+32+0+0+0+2+1;   dx:=COM_number-1;
          intr($14, register);
                      end;
end;


Procedure write_port(dummy_address, databyte:byte);
{Output the databyte:byte to Port[RS232_address]}
begin
     port[RS232_address]:=databyte;
end;


Function Input:byte;
{Input data from Port[RS232_address]}
begin
     Input:=port[RS232_address];
end;


Function data:byte;
{to read data from COM port with valid-data-received detection}
var
   d1x,d2x:array [1..1005] of byte;
   datax:byte;
begin
      repeat until (Read_interrupt_identification(RS232_address) and 1) =0;
      {check if a valid serial data is received by the COM port}
      data:=input; {read the received data}
end;


Function voltage:real;
{read voltage from the logger}
var
   dummy,i,d1,d2:integer;
begin

     write_port(0,1*16+5);                    {send 15=1*16+5 byte to command the logger to
convert};
     d1:=data; delay(1); d2:=data; delay(1);    {receive two bytes from the logger}
     loggerdata:=(d1 and (8+4+2+1) *256 + d2)* 2.50/4096; {combine the two byte in a voltage}
     voltage:=loggerdata;
end;


Procedure Diagram;
{A diagram showing the layout of the data logger}
{showing the analogue conversion results}
begin
     window(1,1,80,25);
     Textbackground(blue);
     textcolor(lightblue);
     clrscr;
     repeat
          gotoxy(15,18); write('  PC sends 15h to logger ');
          delay(20000);
          gotoxy(15,18); write('                         ');
          textcolor(green);
          gotoxy(15,19); write('  Logger starts loggering');
          delay(20000);
          gotoxy(15,19); write('                         ');
          textcolor(red);
          gotoxy(15,20); write('  Logger sends data to PC');
          delay(20000);
          gotoxy(15,20); write('                         ');
          textcolor(red);
          gotoxy(24,15); write(voltage:6:3);
          textcolor(yellow);
     until keypressed;
end;

{*****************MAIN PROGRAM*********************}
begin
      detect_rs232; {check the number of RS232 ports installed on your pc}
      initialize; {initialize the selected RS232 port}
      write_interrupt_enable(RS232_address, 1);
      diagram;
end.
```

Program List 4 (VB5 data logger program)

```
Private Sub Command1_Click()
Dim volt As Single
    Sleep (10)
    MSComm1.Output = Chr$(1 * 16 + 5)
    MSComm1.InputLen = 1
    MSComm1.InBufferCount = 0
    Do
        DoEvents
    Loop Until MSComm1.InBufferCount = 2
    volt = (AscB(MSComm1.Input) * 256 + AscB(MSComm1.Input)) / 4096 * 2.5
    Label1.Caption = format(volt,"0.0000")
    End Sub

Private Sub Command2_Click()
    MSComm1.CommPort = Text1.text
    MSComm1.InputMode = comInputModeBinary
    MSComm1.PortOpen = True
    Sleep (100)
End Sub

Private Sub Command3_Click()
    End
End Sub

Private Sub Form_Load()
End Sub

Module1.bas
Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
```