Stand-alone data logger

You can leave this small, cheap, autonomous and battery-powered data acquisition box almost anywhere, quietly gathering data ready for later retrieval and post processing on a pc. Pei An and Pinhua Xie explain its how it works.

In environmental monitoring applications, parameters such as temperature, humidity, or water level or pollution need to be monitored continuously over long periods.

A conventional pc-based data acquisition system can be used for such an application, but it may not be ideal. Firstly, such a system involves a computer and a data logger, making it expensive. Secondly, the physical size is large. Thirdly, power assumption will be high, and this implies that a powerful battery pack is required in applications where there is no mains supply.

A stand-alone data logger is a useful device for such an application. Firstly, it is dedicated. Its only task is to acquire data and save the data into its memory. It can be connected to a computer any time to allow its collected data to be transferred and analysed.

Such a data logger can be made small in size and with ultra-low power consumption. Such a stand-alone logger's small size allows it to be placed in any location. It can collect data continuously over a long period of time without having its battery changed.

This article describes a design of such a data logger. It has one analogue input channel with an input range of 0 to 2.5V and the a-to-d conversion accuracy is 12-bit. It has an on-board memory capable of holding 1000 a-to-d conversion results

When data is to be downloaded, the logger connects to a

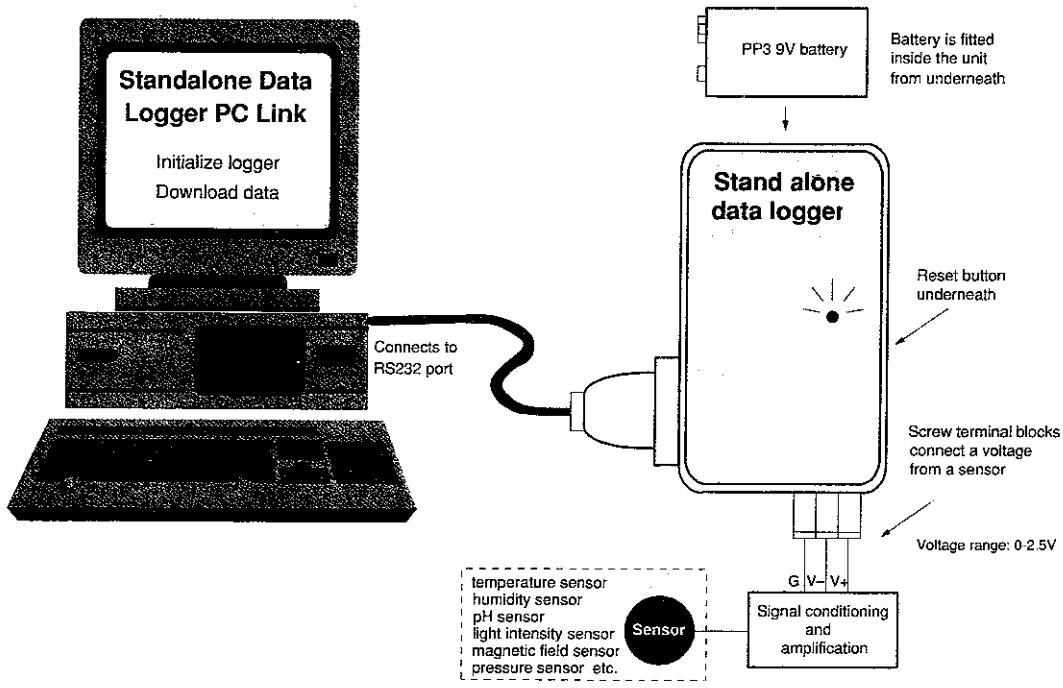


Fig. 1. A stand-alone data logger is used to acquire analogue data from the external world. The logger connects to a host computer via its RS232 port for initial configuration. After that it can be disconnected from the computer and placed in a designated location to acquire data. After the data acquisition session is completed, it is connected to the host computer again for data downloading. The logger only measures analogue voltage. A sensor, amplification and signal conditioning circuits are required to complete the system.

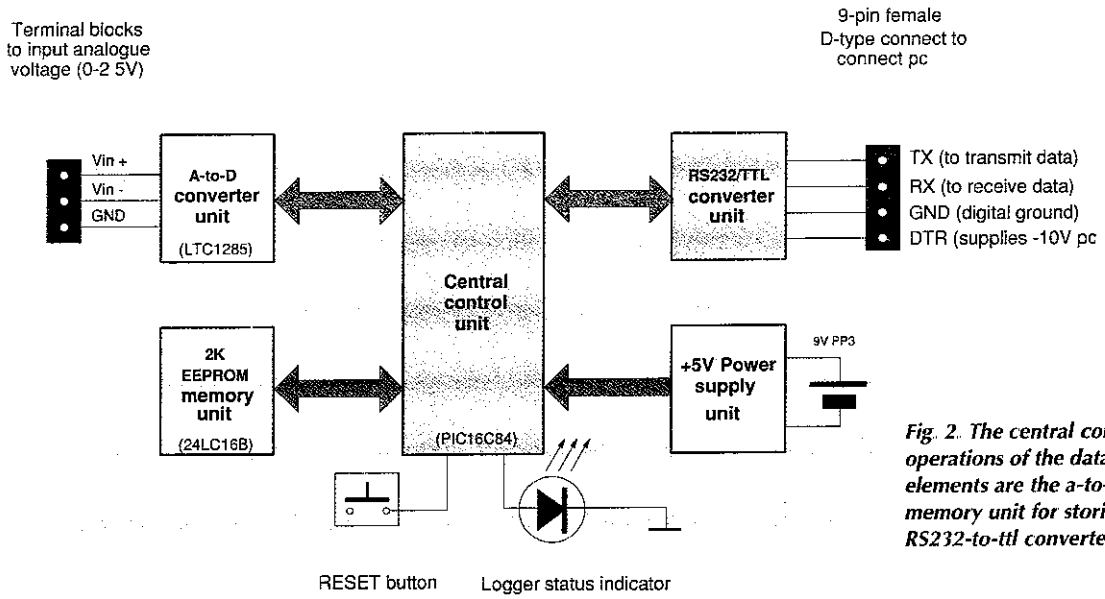


Fig. 2. The central control unit controls all operations of the data logger. Three main elements are the a-to-d converter unit, the memory unit for storing the data and the RS232-to-ttl converter.

.: via an RS232 serial port. When driven by a lithium PP3-sized battery it could capture data for a month or so unattended.

It is possible to build this logger into an enclosure the size of a small calculator. The complete system is illustrated in **Fig. 1**. Bear in mind that different sensors and signal conditioning and amplification circuits may be needed for different applications.

How data logger works

The data logger has three operating modes. These are the initialisation mode, the data logging mode and the data downloading mode.

In initialisation mode, the user specifies the start time of data logging and scanning interval – i.e. the period between two consecutive data loggings. This is done by plugging the data logger to the RS232 port of a host computer.

After the initialisation, the logger enters data logging mode. It can now be disconnected from the computer and placed to a designated location. The data logger converts analogue signal into digital data at a fixed interval and stores the data into its memory.

Data logging is terminated either by pressing the reset button on the logger or when the total number of stored data exceeds 1000. At this point, the logger is connected to the host computer once more for data downloading. During downloading, the data stored in the data logger are transferred into the computer.

Hardware details

Figure 2 shows the logger's block diagram. The system comprises five units. They are,

- Central controller based on the *PIC16C84*
- *LTC1285* analogue-to-digital converter
- *24LC16* memory unit ,
- RS232-to-ttl converter
- power supply

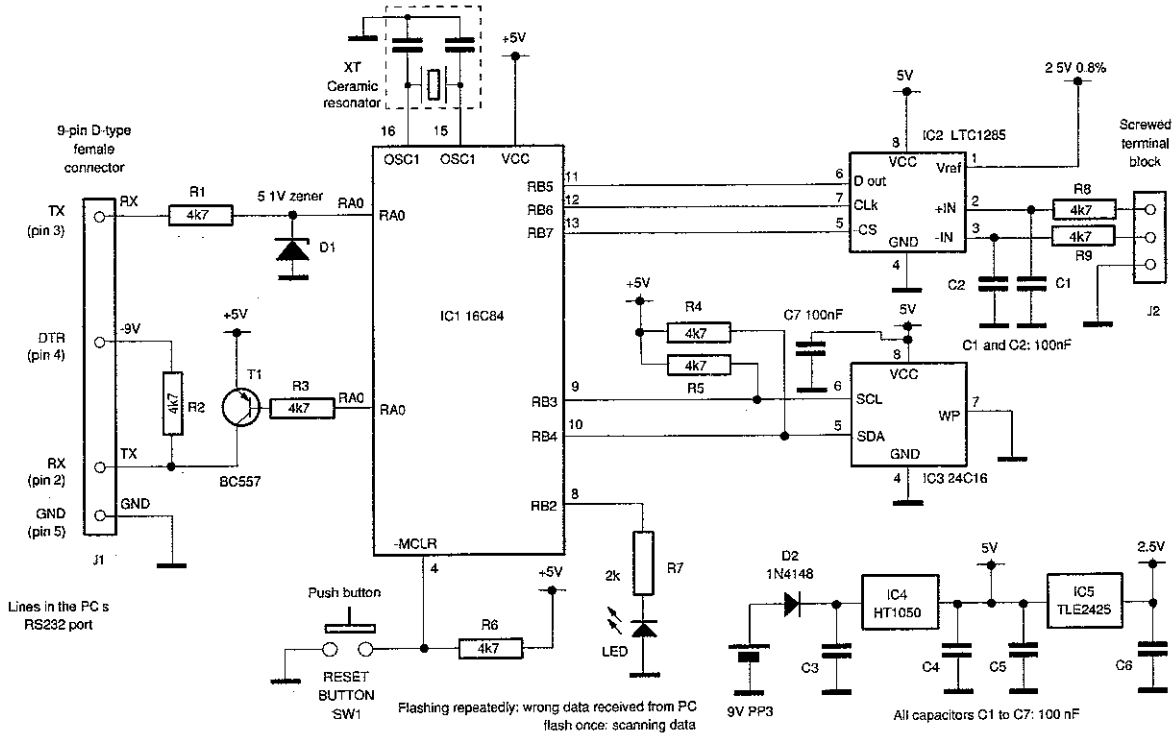


Fig. 3. Complete circuit diagram of the stand-alone data logger. A PIC16C84, an LTC1285 a-to-d converter and a 24LC16B 2Kbyte electrically-erasable prom are used in the circuit, which may be constructed on a single-sided board.

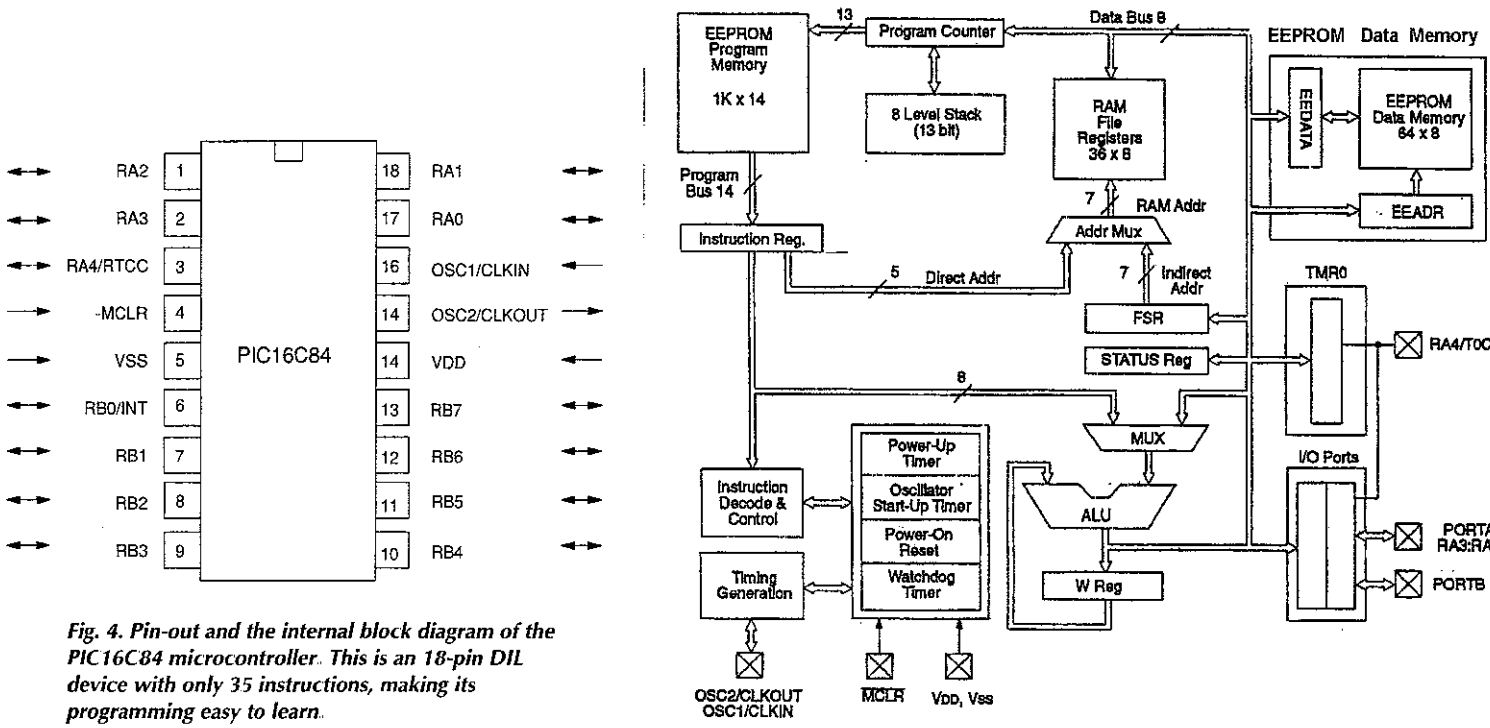


Fig. 4. Pin-out and the internal block diagram of the PIC16C84 microcontroller. This is an 18-pin DIL device with only 35 instructions, making its programming easy to learn.

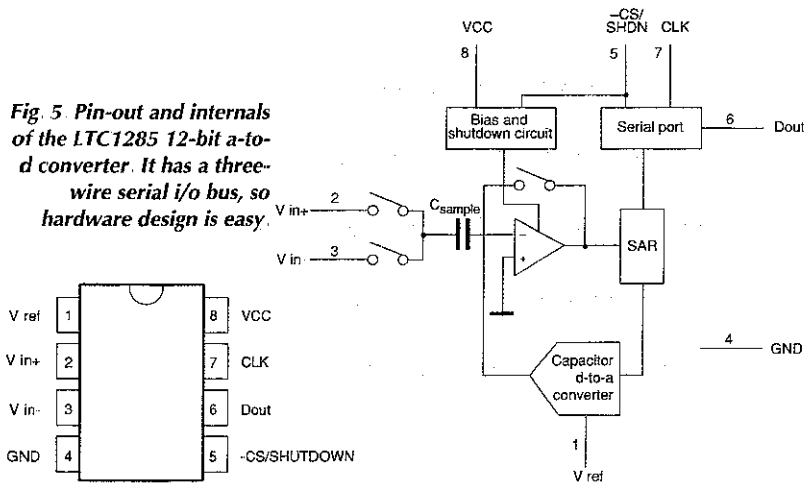


Fig. 5. Pin-out and internals of the LTC1285 12-bit a-to-d converter. It has a three-wire serial i/o bus, so hardware design is easy.

Table 1. In this application, lines of the PIC controller are designated as follows.

Line	Pin	Description
Port A		
RA_0	18	Serial data input to the PIC (connected to Tx of the PC's RS232 port)
RA_1	17	Serial data output from the PIC (connected to Rx of the PC's RS232 port)
RA_2	1	Not used
RA_3	2	Not used
RA_4	3	Not used
Port B		
RB_0	6	Not used
RB_1	7	Not used
RB_2	8	Control of the logger status led (output)
RB_3	9	Serial clock of the I^2C bus (SCL) for the 24LC16B (output)
RB_4	10	Serial data of the I^2C bus (SDA) for the 24LC16B (input and output)
RB_5	11	Serial data output (D_{out}) from the LTC1285 (input)
RB_6	12	Serial clock (CLK) of the LTC1285 (output)
RB_7	13	enable (–CS) of the LTC1285 (output)

The complete circuit is given in **Fig. 3**. The system utilises only three key ICs, namely the controller, the a-to-d converter and the memory. The *LTC1285* a-to-d converter has a serial peripheral interface, or SPI, for all i/o operations. The 2K eeprom has a 2Kbyte capacity. It communicates with external devices using an I^2C bus. The *PIC16C84* manage the gathering of data from the a-to-d converter and the storage of it in the eeprom. It also looks after communication with the host computer via the RS232 port.

Central control unit. The central control unit is based on a Microchip *PIC16C84* peripheral interface controller. This device is a relatively recent addition to Microchip's range. It has has an electrically erasable memory to store program, making it particular useful for product development. This is why we adopted it for this application.

The pin-out and internal block diagram of the *16C84* is shown in **Fig. 4**. Pins 14 and 5 connect to the positive and negative rails of a power supply. The supply voltage range is 2 to 6V. Power supply current is typically 2mA at 5V and 4MHz clock frequency. This drops to several tens of microamps when the IC is in standby mode.

Pin 4 is the master clear. It must be held high in normal operation. Pin 15 and 16 connect to a crystal or ceramic resonator up to 4MHz. The *16C84* has a 1024 word 14-bit wide electrically erasable prom to store instructions and a 64 byte eeprom to store data. There are 15 special function registers and 36 byte-wide general purpose registers.

There are two i/o ports. Port A is brought out on RA_{0-4} while port B is on RB_{0-7}. Each individual line can be configured as an input or output. As an output, any line is able to source 20mA and sink 25mA. Line RA_4 has a secondary function. It is used for the timer/counter modules. Also, RB_0 doubles as an external interrupt input.

The PIC has only 35 single-word instructions, which makes programming it easy to learn. In this application, the PIC works in the crystal-oscillator mode. A 4MHz ceramic resonantor – a three pin device – is used, Fig. 3. Input/output lines of the PIC are committed in as in **Table 1**.

Analogue to digital converter unit. The analogue-to-digital converter core is a Linear Technology *LTC1285CN8* 12-bit a-to-d converter using successive approximation conversion, **Fig. 5**. It requires a power supply of 2.7V to 6V on pins 4 and 8 and a reference voltage on pin 1.

Typical supply current is 260µA at a 6.6kHz sampling rate and with a 2.7V power rail. When in standby mode, the supply current drops to several nanoamps. The *1285* has a differential analogue input on pin 2 and 3 and the analogue input leakage current is typically 1µA.

The converter communicates with other circuitry through a three-wire SPI serial interface. These three wires are –CS/SHDN, CLK and D_{out}. On going low, pin 5 selects the chip and initiates data transfer. If the pin goes high, the converter enters standby mode.

Pin 7 is the clock input. It synchronises the serial data transfer and determines conversion speed. At the falling edge of the clock signal, each bit of the 12 bits of an a-to-d conversion result is sent out from D_{out} pin.

The operating sequence of the *1285* is shown in **Fig. 6**. Data transfer is initiated at the falling edge of the chip select, pin 1. Following chip select's falling edge, the second clock pulse enables data output D_{out}. A null bit (logic 0) appears first on D_{out}, pin 6. At the next 12 falling edge of the clock, the 12 bits of the a-to-d conversion result appear on D_{out} one by one.

In the present circuit, –CS, D_{out} and CLK connect to RB_7, RB_5 and RB_6 of the PIC. The PIC sets RB_7 (–CS) and RB_6 (CLK) as output lines. Line RB_5 is set as an input.

Memory unit. The memory unit uses a *24LC16B* 2Kbyte eeprom from Microchip. The memory is organised is 2Kbyte memory locations. It is possible to erase and write to the memory up to a million times.

The chip requires a 2.5V to 5V power supply with a typical current consumption of 1mA in active mode and 10µA in standby mode. It has an I^2C bus for data transfer operations and it operates as a slave device on the bus, **Fig. 7**.

Lines A_{0-2} have no function and can be left open. Pin WP is for write protection and is normally tied to the ground to enable write operation. Lines SCL and SDA are the clock and data lines of the I^2C bus.

Data can be written to and read from the rom via the I^2C bus. The write operation has two modes – byte-write mode and page-write mode. The former writes a single byte to a memory location. The latter writes 256 bytes to a block in one go. The read operation has a current-address-read mode and a random-read mode. Byte-write mode and the random-read mode are used in this application. Their timing sequence is described below, **Fig. 8**.

Following a start condition on the I^2C bus, an eight-bit slave address byte is clocked into the memory from the controller. The slave address from bits 7 to 0 is:

1, 0, 1, 0, B_2, B_1, B_0 and R/–W.

Bits 7 to bit 4 are the permanent address of the *24LC16* memory. Bits B_{2-0} specify one of the four memory blocks. When R/–W is high, the operation is a read operation, otherwise it is a write operation.

After the slave address bits are transferred into memory, an address byte is transmitted to it which specifies a particular memory location in the selected memory block. This address is written to the address pointer of the *24LC16* and its value ranges from 0 to 255.

If the operation is a write operation, the eight bits of data are sent to it next. In random-read mode, after writing to the address pointer, a start condition is generated again and it is followed by sending slave address bits with the R/–W bit set to 1, to signify reading. Now, the data stored in the memory is sent out bit by bit.

List 1. These nine bytes are sent to the PIC controller in the logger from the pc immediately after initialisation.

```
byte 1: data logging launch time  year (0-99 decimal)
byte 2: data logging launch time, month (1-12 decimal)
byte 3: data logging launch time, day (1-31 decimal)
byte 4: data logging launch time, hour (1-24 decimal)
byte 5: data logging launch time, minute (1-59 decimal)
byte 6: scanning rate (1=1 second, 2=1minute, 3=1 hour)
byte 7: delay number high byte, Dh
byte 8: delay number mid byte, Dm
byte 9: delay number low byte, Dl
```

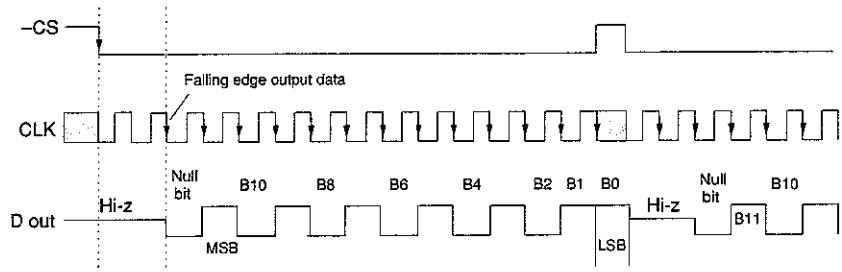


Fig. 6. Timing sequence of the LTC1285. After –CS falls, the converter enters the data conversion stage. The falling edge of the third clock pulse causes D_{out} to output bit 11 of the conversion result. The following 11 clock pulses causes D_{out} to output bit 10 to bit 0.

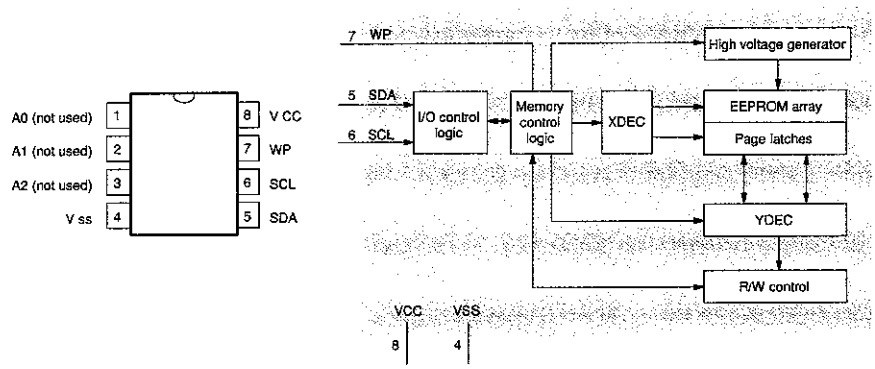
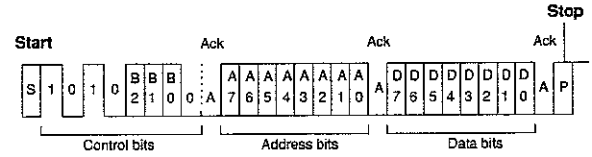
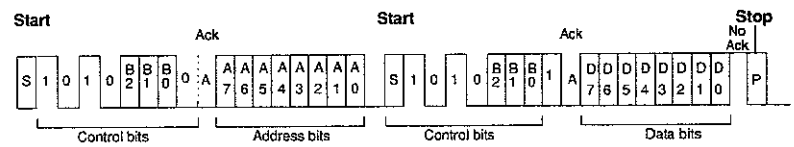


Fig. 7. Details of the 24LC16 2kbyte eeprom. It has an I^2C bus comprising a clock line, SCL, and a data line, SDA.



(a) Byte write operation



(b) Random read operation

Fig. 8. Timing sequence of the 24LC16 eeprom. See details of the I^2C bus operation in the panel.

List 2. When the logger is to off-load its data, it send this sequence to the pc first, followed by the stored data.

```
byte 1: data logging start time, year (0-99 decimal)
byte 2: data logging start time, month (1-12 decimal)
byte 3: data logging start time, day (1-31 decimal)
byte 4: data logging start time, hour (1-24 decimal)
byte 5: data logging start time, minute (1-59 decimal)
byte 6: scanning rate (1=1 second, 2=1minute, 3=1 hour)
byte 7: total number of data logged. lower 8 bits, Dl
byte 8: total number of data logged. upper 8 bits, Dh
```

In the present circuit, SCL and SDA are controlled by the PIC via $RB_{3,4}$. Both lines are pulled high by $R_{4,5}$ to form an I^2C bus. The PIC permanently sets RB_3, the SCL line, as an output line. Depending on the I^2C operation in progress, SDA on RB_4 is set as an input or an output.

RS232/ttl translator unit. The function of this unit is to perform voltage conversions between RS232 and ttl logic levels. From the circuit diagram, you will see that the Rx line – i.e. the line from which the logger receives data, RS232 voltage level – is converted into a ttl voltage level using a simple clamp based on R_1 and zener diode D_1. This converter does not have an inverting action.

The Tx signal – the signal output from the logger, RS232 voltage level – is generated by a circuit consisting R_2, R_3 and Tr_1. The circuit requires a positive and a negative power supplies. The former is from the +5V power supply of the data logger board. The latter is 'stolen' from the RS232 port of the computer. The DTR, or data-terminal ready, line in the pc's RS232 port is set low which outputs a –10V level. The pin-out of the pc's RS232 port connector and its functions are given in **Fig. 9**.

Supplying power

As Fig. 3 shows, the power supply is a PP3 9V battery, regulated to +5V using an *HT1050* regulator. This is a 5V fixed voltage regulator with a maximum supply current 30mA. It offers a very low dropout voltage of 100 mV and a quiescent current of 3.5μA.

The 5V supply is converted into 2.5V by the *TLE2425* 2.5V voltage reference for use by the a-to-d converter.

Our stand-alone data logger implementation is constructed on a single-sided pcb and is housed in a slim ABS box.

Software for the PIC

The PIC software divides into three main procedures. The first is the initialisation procedure, the second is the data logging procedure and the third is the data downloading procedure, **Fig. 10**. Their functions are described briefly below:

After pressing the reset button, the initialisation procedure is activated once the PIC detects a serial byte AA_{16} sent by the host computer at its RA_1 pin. After this, the procedure receives nine initialisation bytes, as described in **List 1**.

Delay period in second is calculated using the following,

$$2.6\times(256\times256\times D_h+256\times D_m+D_l)$$

To carry out the initialisation, the host computer must send AA_{16} and the nine bytes through its RS232 port.

After the PIC receives the ninth byte, it automatically enters the data logging mode. Firstly, the data logger is in the sleep mode until the launch time of data logging is reached. While the PIC is not logging data, the PIC, a-to-d converter and the memory are all in sleep mode.

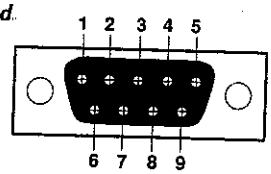
The PIC wakes up and makes the *TLC1285* converter perform an a-to-d conversion. The resulting bits are read into the PIC serially. After reading 12 data bits, the PIC writes the value into the *24LC16B* eeprom.

Next, the PIC goes back to sleep. It waits for a time period as specified by the scanning interval and then starts another data logging cycle. There are two ways to terminate the data logging. One is to press the reset button anytime. The other is that the number of data stored in memory exceeds 1000.

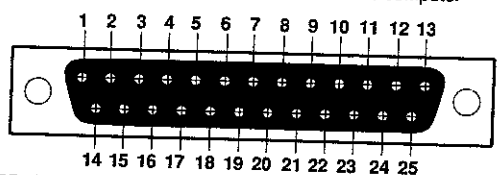
Data downloading to the pc is activated once the PIC detects an RS232 serial byte 55_{16} sent by the host computer from its RB_1 line, after the reset button is pressed. Following this, the data logger begins to output data. The first eight bytes are data headers and the logged data follows, **List 2**. The total number of data stored by the data logged is calculated using $D_h\times256+D_l$.

The bytes succeeding those are data bytes. Two bytes rep-

Continued over page



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



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

Fig. 9. Pin-out of the RS232 port on IBM compatibles. In this application, only the Tx transmit output from the PC, the Rx input and the DTR output are used.

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

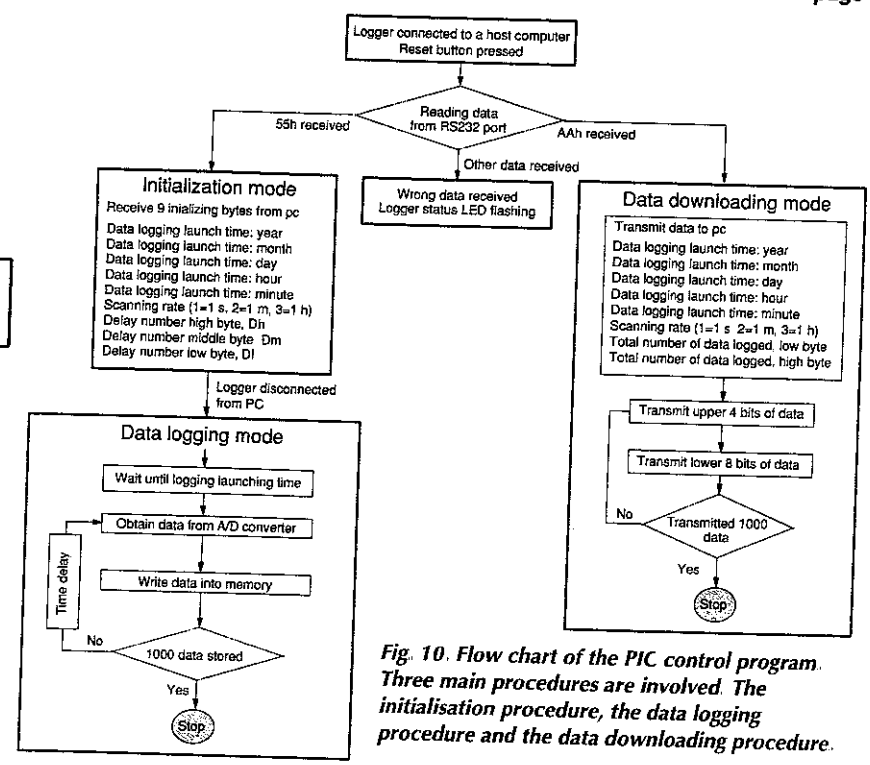


Fig. 10. Flow chart of the PIC control program. Three main procedures are involved. The initialisation procedure, the data logging procedure and the data downloading procedure.

What is I²C bus?

Devised by Phillips, I²C stands for inter-IC-communication. It is a data bus that allows integrated circuits or modules to communicate with each other.

The bus allows data and instructions to be exchanged between devices via only *two* wires. This greatly simplifies the design of a complex electronic circuits. There is a family of I²C compatible devices available for various applications. They include i/o expansion, a-to-d and d-to-a conversion, time keeping, memory and frequency synthesis, etc.

Principle of the I²C bus. The I²C bus consists of two lines: a bi-directional data line called SDA and a clock line called SCL. Both are pulled up to the positive power supply via resistors. An I²C bus system is shown in **Fig. A**.

A device generating a message is a 'transmitter' while a device receiving a message is the 'receiver'. The device controlling the bus operation is the 'master' and devices controlled by the master are 'slaves'.

The following communication protocol is defined:

- a data transfer may be initiated only when the bus is not busy
- during the data transfer, the data line must remain stable whenever the clock line is high.

Changes in the data line while the clock line is high is interpreted as control signals. The following bus conditions are defined, **Fig. B**

- **Bus not busy**: both data and clock lines remain *high*
- **Start data transfer**: a change in the state of the data line *from high to low* while the clock is *high*, defines the start condition
- **Stop data transfer**: A change in the state of the data line *from low to high* while the clock is *high* defines the stop condition.
- **Data valid**: The state of the data line represents valid data after a start condition. The data line is *stable* for the duration of the *high period of the clock signal*. The data on the line may be changed during the low period of the clock signal. There is one clock pulse per bit data. Each data transfer is initiated with a start condition and terminated with a stop condition. The number of data bytes transferred between the start and stop conditions is not limited. The information is transmitted byte-wise and the receiver acknowledges with a ninth bit.
- **Acknowledge bit**: Each byte is followed by an acknowledge bit. The acknowledge bit is a high level put on the bus by the transmitter whereas the master generates an extra acknowledge related clock pulse. The acknowledge bit is a low level put on the bus by the receiver. A slave receiver which is addressed is obliged to generate an acknowledge bit after the reception of each byte.

The device that acknowledges has to pull down the SDA line during the acknowledge clock pulse in such a way that the SDA line is at a stable low state during the high period of the acknowledge related clock pulse. A master receiver must signal an end to the slave transmitter by not generating an acknowledge on the last byte that has been clocked out of the slave.

How the bus operates. Before any data is transmitted on the bus, the device which should respond is addressed first. This is carried out with the seven-bit address byte plus R/–W bit transmitted after a start condition. A typical address byte has the following format:

```
Fixed Address bits + Programmable address
bits + R/-W bit (in total 8 bits)
```

The fixed address depends on the IC and it can not be changed.* The programmable address bits can be set using the address pins on the chip. The last bit is the read/write bit which indicates the direction of data flow. The byte following the address byte is the control byte which depends on the IC used. Following the control byte are the data bytes. The serial data has the format shown in Fig. 8.

*Although some I²C devices have inputs that can modify the address depending on their logic state, allowing more than one of the same ic to be used on the same bus – Ed

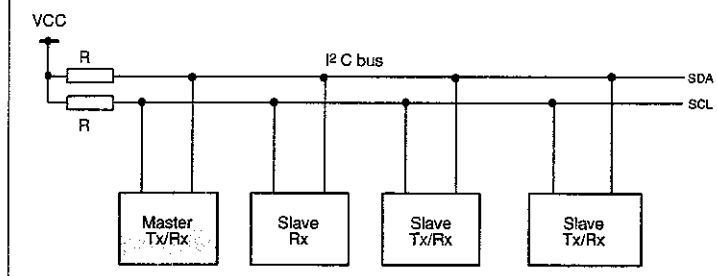


Fig. A. An I²C bus consists of only two data lines: serial clock, SCL, and serial data, SDA. I²C compatible devices connect to the bus using these two wires, making hardware design simple.

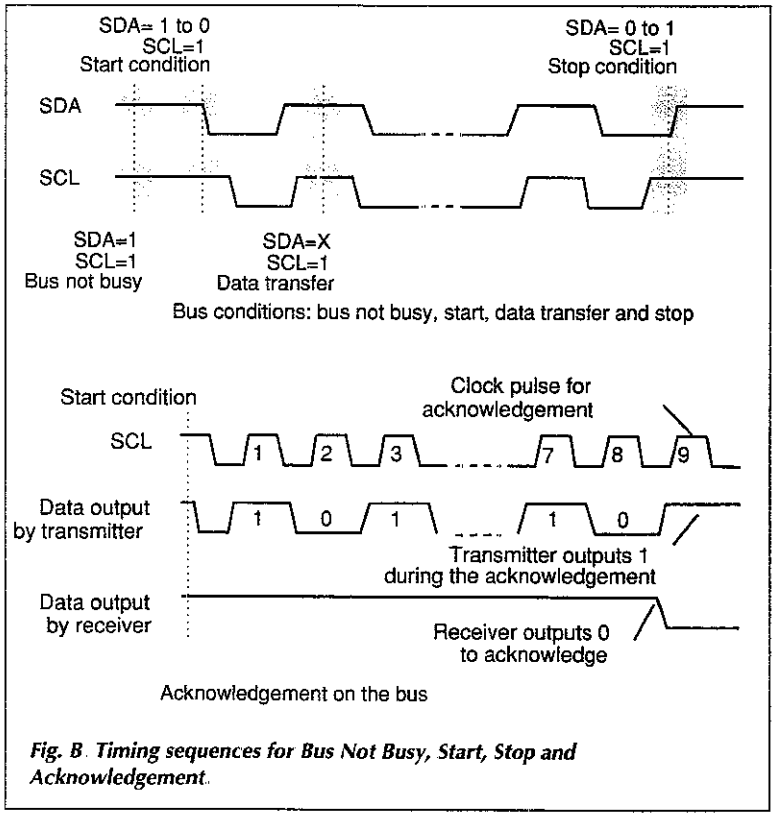


Fig. B. Timing sequences for Bus Not Busy, Start, Stop and Acknowledgement.

List 3. Outline of how to send initialisation data to the logger using Turbo Pascal 6.

```
Procedure init_logger
{initialize data logger}
begin
    ; {send AA=10*16+10 byte to start initialization};
    Port[RS232_address]:=10*16+10
    delay(1000); {a short delay}
    Port[RS232_address]:=start_year;     delay(1000);
    Port[RS232_address]:=start_month;    delay(1000);
    Port[RS232_address]:=start_day;      delay(1000);
    Port[RS232_address]:=start_hour;     delay(1000);
    Port[RS232_address]:=start_minute;   delay(1000);
    Port[RS232_address]:=scanning_inteval;   delay(1000);
    Port[RS232_address]:=delay_h;        delay(1000);
    Port[RS232_address]:=delay_m;        delay(1000);
    Port[RS232_address]:=delay_l;        delay(1000);

End'
```

List 4. Routine for downloading data from the data logger into the pc.

```
Procedure readdata;

Function data:byte;
{to read data from COM port with valid-data-received detection}
begin
    ; {check if a new valid data is received}
    repeat until (Port[RS232_address+2] and 1) =0
    {check if a valid serial data is received by the COM port}
    data:=port[RS232_address]; {read the received data}
end;
begin
    port[RS232_address]:= 5*16+5; {to start data downloading pro-
cedure}
    yearx:=data;
    monthx:=data;
    dayx:=data;
    hourx:=data;
    minutex:=data;
    scan_intervalx:=data;
    number_lowbyte:=data;
    number_highbyte:=data;
    Total_number:=number_lowbyte+number_highbyte*256;
    for i:=1 to total_number do
    begin
        d1:=data;
        d2:=data;
        data_from_logger[i]:=(d1*256 + d2)* 2.50/4096;
    end;
```

List 5.

```
Procedure Write_interrupt_enable(RS232_address,
Output_byte: integer);
{to enable interrupt identification 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;
```

resent a 12-bit a-to-d conversion. The upper four bits are sent first, then the lower eight bits.

The procedure causes the PIC to output 2000 bytes – which is in effect 1000 data words. While the PIC is doing so, it does not generate any handshake signals. The host computer must be able to detect each valid received byte and to read it. This is easily achieved on modern computers.

The program list of the PIC control software is lengthy and it is not possible to include it here, but it is available from the authors, as described in the Technical Support panel.

PC link software

This section describes how a personal computer controls the data logger and presents some hands-on programming examples. Turbo Pascal 6 programming language is used.

The following Pascal procedure shows how to send initialisation data to the data logger via its RS232 port. A Pascal command

```
"Port[RS232_address]:=DATA"
```

is used to output the variable DATA from the RS232 port. The RS232 port address (RS232_address) should be supplied to this command.

There are various ways of finding the RS232 address. You will notice that initialisation starts by sending a AA_{16} byte to the stand-alone data logger. A short delay is need between each data transmission, **List 3**.

The following procedure downloads data from the data logger. The download procedure starts with a Pascal command ,

```
"Port[RS232_address]:=55h"
```

Next the pc reads data from the data logger, **List 4**.

When the computer reads serial data from the RS232 port, it must be able to detect when a valid serial data is transmitted from the data logger to the computer. This is achieved by enabling the selected COM port to generate a valid-data-received identification.

After a valid data transmission is completed, bit 0 of the interrupt-identification register of the selected COM port goes low. The register has an address of: RS232_address+2. The way to do the checking is shown in 'Function data:byte' in List 4.

To enable the COM port to generate a valid-data-received identification, the procedure in **List 5** is used before calling the above procedure. The variable output_byte should be 1.

After the program reads all the data from the data logger, It saves the data into a dos text file. The data can be analysed by spreadsheet packages such as the Microsoft excel.

The complete program list of the pc link software is lengthy. It is available from the authors if required. ■

Technical support

Designers' kits containing all the necessary components to construct a complete stand-alone data logger are available from the authors. This includes a pre-programmed PIC. Source code for the PIC and the computer linker program are also available. Please make your enquiry to Dr Pei An, 11 Sandpiper Driver, Stockport, Manchester SK3 8UL, UK. Tel/Fax/Answer:+44-(0)161-477-9583. Alternatively, e-mail to pan@fs1.eng.man.ac.uk