12 Applications

The PC is a potential prime mover in a huge variety of process control and instrumentation applications ranging from simple stand-alone machine controllers to fully integrated production control systems. This chapter aims to provide readers with an introduction to the procedure for selecting and specifying hardware and software for a PC-based instrumentation or process control system. In addition, several representative applications of PC-based systems are discussed.

Expansion cards

The range of PC expansion boards currently available from a large number of manufacturers includes:

- Analogue I/O cards with up to 16 analogue inputs and up to four buffered analogue outputs.
- Digital I/O cards with direct TTL-compatible inputs and outputs.
- Digital I/O cards with opto-isolated inputs and outputs.
- Digital I/O cards with buffered I/O lines.
- Digital output cards fitted with relays or solid-state devices for AC or DC power control.
- EPROM programmers.
- IEEE-488/GPIB interface cards.
- Network adapter cards.
- Modem cards.
- Prototyping cards (these may include the necessary PC ISA/PCI bus interface logic and provide the user with an area for soldering components fitted into a 0.1” matrix of plated through holes).
- Serial communications cards for RS-232, RS-422, RS423 or RS-485 serial ports.
- Stepper motor controllers.
- Multi-function I/O cards (offering mixed analogue and digital I/O facilities).
- Thermocouple interface cards.
- High-speed data acquisition cards.
- Bus expansion cards (which interface with external card frames or mother boards).
- PC instrument cards (e.g. function generators, counters/digital frequency meters, spectrum analysers, etc.).

In addition, the system builder is able to select from a large range of signal conditioning cards which provide the necessary interfacing circuitry for a wide range of popular sensors and output devices. It is thus eminently possible to construct a PC-based process control system simply by selecting ‘off-the-shelf’ modules. Only when dealing with very specialized applications is it necessary to manufacture ones own dedicated I/O cards and/or external signal conditioning boards. Appendix J lists a number of major suppliers of PC expansion cards and associated signal conditioning equipment.

Approaches

The system designer can select from a range of options depending upon the complexity and individual requirements of a particular application. The following general approaches are available:

- Stand-alone PC systems based on internally fitted expansion cards, rack modules, or separately enclosed units).
- PC systems based on standard PC expansion cards (and I/O processing cards, where appropriate) fitted into external card frame modules.
- Industrial PC systems (using a ruggedized PC functioning as a dedicated process controller or data-gathering device) fitted with internal or external expansion cards and housed in a rack or freestanding enclosure.
- RS-232 based systems with the PC as controller (peripheral hardware connected via an asynchronous serial link).
- IEEE-488 based systems with the PC as controller.
- Backplane bus-based systems with a PC bus master/controller and a card frame bus.
- Networked/distributed PC systems (e.g. based on Ethernet or BITBUS) with enclosures and expansion cards to meet local requirements.

PC instruments

In addition to the vast range of expansion cards currently available, several manufacturers have developed a range of dedicated PC instruments (see Chapter 11) that emulate conventional items of test equipment (such as oscilloscopes, counters, function generators, and digital frequency meters).

A PC instrument offers many advantages over its conventional counterpart. It is flexible and adaptable and, in many cases, measurements may be automated under programmed control. Furthermore, considerable savings can be achieved from the elimination of redundant hardware (such as displays, operator controls, power supplies, etc.).

PC-based instruments can also offer very significant cost savings when compared with simple IEEE-488 bus-based instrumentation systems. A typical PC-based system for the acquisition of analogue voltages can, for example, be realized for less than 50% of the cost of a similarly specified system based on IEEE-488 hardware and software.

PC-based instruments are available in three general formats (Figure 12.1):

1 Using internally-fitted expansion cards (plugged into a free slot in the PC).
2 Using an external rack with plug-in PC expansion cards.
3 Using separately enclosed modules (which may, if desired, be stacked) based on RS-232, USB or IEEE-488 bus systems.

All three of these approaches have their own particular virtues and the system builder should include all three in his or her portfolio of potential engineering solutions.

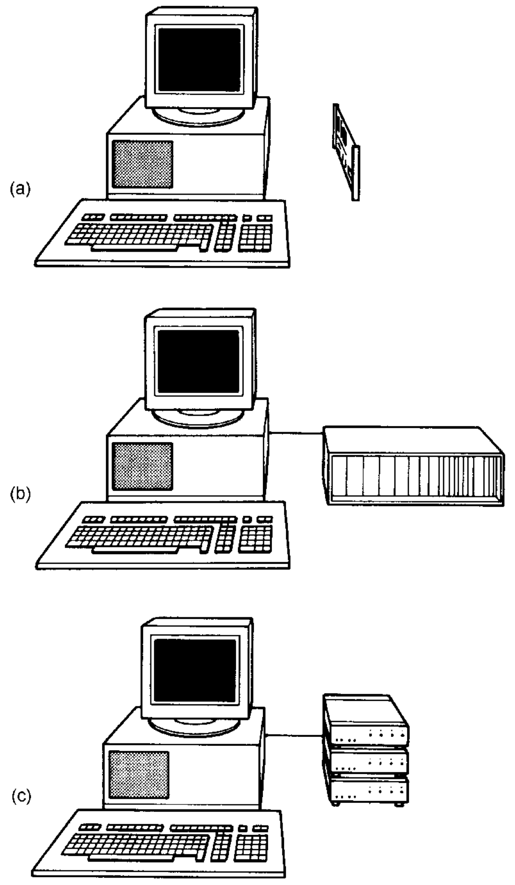


Figure 12.1 *Three basic approaches to PC-based instruments: (a) internally fitted PC expansion cards; (b) PC expansion cards in an external card frame; (c) separately enclosed PC instruments linked via a parallel port, RS-232 serial port, USB or via the IEEE-488 bus*

Internally-fitted cards generally offer the lowest cost approach to building a PC instrument. The disadvantage of this technique is that it necessitates internal fitting and, since there may be a limited number of slots available, the expansion capability may he somewhat limited.

An external rack system allows the PC bus to be extended so that standard PC expansion cards may be fitted into an external card frame. This system is, however, relatively expensive and generally only appropriate where large-scale expansion is required. An alternative to that of extending the PC bus beyond the confines of the system enclosure is that of making use of a proprietary I/O bus. Such systems generally provide for between one and 32 I/O boards mounted in standard rack enclosures.

Separately enclosed modules (which may be interfaced to a PC by means or the RS-232, USB or IEEE-488 bus) provide the third of this trio of potential solutions. With the exception of a front panel display controls, separate PC instrument modules usually resemble the conventional stand-alone instruments which they replace. However, in a relatively recent development, USB instruments provide a low-cost solution and several manufactures are actively developing test instruments for budget conscious application that make use of the PC's USB port.

Several manufacturers have risen to the challenge of producing PC instruments such that the range of equipment currently available includes oscilloscopes, digital multi-meters, universal counters, timers and frequency meters, spectrum analysers, function generators, pulse generators, voltage/current generators and logic analysers. Individual instruments can be combined to provide more complex instrumentation facilities. A data logger, for example, can be assembled from a scanner and multi-meter and controlled flexibly from the PC.

PC instruments are ideal for making repetitive measurements during which data must be accumulated over a period time. The PC allows such measurements to be automated with the data acquired being sent to a file for future analysis.

As an example of the use of a PC instrument, consider an application in which the output frequency of an oscillator has to be monitored accurately over a long period of time. This task can he accomplished by means of a dedicated digital frequency meter with readings taken at appropriate intervals, logged on paper, and a graph showing the long-term variation of frequency can then be drawn.

The alternative approach using a 'PC instrument' simply involves fitting a digital frequency meter expansion card (such as the Guide Technology GT200) to a standard PC-compatible and using simple software (in conjunction with the driver(s) supplied with the card) to automate the measurement and store the results in a data file for import into an analysis package (such as DADiSP). A typical application is discussed later in this chapter.

Industrial PC systems

Ruggedized PCs are the obvious choice for use in the harsh environment found in most industrial plants. Industrial PCs usually offer the same range of facilities associated with conventional PCs and compatibles and invariably support the industry standard bus architecture. Hence an industrial PC will generally accept the same range of expansion cards as mentioned under the previous heading. Alternatively, where additional expansion beyond the limit imposed by the available free slots, industrial PCs may be fitted with bus extenders which are normally based on an external rack assembly.

In difficult environments it is possible to implement a completely *diskless system* using solid-state read/write memory devices (e.g. Flash, SD or XD cards) inserted into IDE slots in order to provide a bootable operating system together with one or more application programs.

Backplane bus-based systems

A backplane bus system offers a reasonable compromise between a standard PC-based system at one extreme and a specialized industrial PC system at the other. Backplane bus systems are inherently flexible and reliable and can simply be fitted with a PC processor card in order to make use of standard PC software packages.

Networked/distributed PC systems

Networked or distributed PC systems are appropriate in large-scale applications where several processes are carried out concurrently. Each individual PC will be responsible for part of the process and data will be shared between the PCs by means of the network. As an example, consider the case of a packaging plant which manufactures and fills cardboard boxes on a continuous basis. One PC may be dedicated to the cutting and folding operation whilst another may be responsible for controlling glueing and stapling. A third PC would be responsible for filling and sealing the boxes. Data from all three PCs would then be collected by a fourth PC which oversees the entire process. Such a system provides an alternative to conventional solutions based on distributed programmable logic controllers (PLC).

Larger scale systems are possible using bus systems based on Process Field Bus (Profibus), Actuator Sensor Interface (AS-Interface), Interbus, Modbus-1, BITBUS and Ethernet. These systems provide remote I/O with a maximum number of nodes ranging from around 32 to 512 depending upon the standard concerned (Ethernet is theoretically unlimited). These arrangements are used for use in large scale manufacturing and process control systems and are thus somewhat beyond the scope of this book.

Intel’s BITBUS (IEEE-1118) provides a simple and elegant solution to applications that require the services of a *multi-drop network*. BITBUS is a serial data bus based on the RS-485 physical and electrical interface standard (RS-485 is a multi-drop version of RS-422) and the datalink protocol employed is a subset of SDLC/HDLC.

BITBUS complements Manufacturing Automation Protocol (MAP) which has gained widespread recognition as the industrial standard for the upper level of factory data communications. At the machine and process level, however, where time critical data from sensors, actuators and alarms has to be transmitted, the response time of MAP, though guaranteed, is inadequate. BITBUS, on the other hand, is well suited to the transfer of short ‘Field Data’ messages.

BITBUS is configured as a single-master, multi-slave network and operates in one of two modes, synchronous and self-clocked. Synchronous operation permits speeds of up to 2.4 megabits/s but requires twin twisted-pair cables and is restricted to transmission over distances less than 300 m. Furthermore, since repeaters cannot be used in this mode, a maximum of 31 nodes is possible. Self-clocked mode, on the other hand, requires only single pair cable, can operate at either 62.5 or 375 kilobit/s and, with repeaters, can cater for up to 250 nodes at distances not exceeding 13km. Access times of three to four milliseconds per command are easily achieved.

Interfacing with BITBUS is usually made possible with the use of an Intel 8044/80154 compatible micro-controller which implements the BITBUS protocol using an on-chip SDLC controller and ROM-based firmware. An interface of this type may be incorporated within a processor card or may be provided as part of an auxiliary communications interface.

Specifying hardware and software

When specifying hardware and software to be used in a given PC bus application, it is essential to adopt a ‘top-down’ approach. An important first stage in this process is that of defining the overall aims of the system before attempting to formalize a detailed specification. The aims should be agreed with the end-user and should be reviewed within the constraints of available budget and time. Specifications should then be formalized in sufficient detail for the performance of the system to be measured against them and should include such items as input and output parameters, response time, accuracy and resolution.

Having set out a detailed specification, it will be possible to identify the main hardware elements of the system as well as the types of sensor and output device required (see Chapter 9). The following checklist, arranged under six major headings, should assist in this process:

1 *Performance specification*

- What are the parameters of the system?
- What accuracy and resolution is required?
- What aspects of the process are time critical?
- What environment will the equipment be used in?
- What special contingencies should be planned for?
- What degree of fault-tolerance is required?

2 *I/O devices*

- What sensors will be required?
- What output devices will be required?
- What I/O and signal conditioning boards will be required?
- Will it be necessary to provide high-current or high-voltage drivers?
- Should any of the inputs or outputs be optically isolated?

3 *Displays and operator inputs*

- What expertise can be assumed on the part of the operator?
- What alarms and status displays should be provided?
- What inputs are required from the operator?
- What provision for resetting the system should be incorporated?

4 *Program/data storage*

- What storage medium and format is to be employed?
- How much storage space will be required for the operating system and/or control program?
- How much storage space will be required for data?
- How often will the control program need updating?
- Will stored data be regularly updated during program execution?
- What degree of data security and integrity must be achieved?

5 *Communications*

- What existing communications standards are employed by the end-user?
- Will a standard serial data link based on RS-232 be sufficient or will a faster, low-impedance serial data communications standard be needed?
- What data rates will be required?
- What distances are involved?
- Will it be necessary to interface with automatic test equipment?
- Will a networking capability be required?

6 *Expansion*

- What additional facilities are envisaged by the end-user?
- What additional facilities could be easily incorporated?
- Will expansion necessitate additional hardware, additional software, or both?
- What provision should be made for accommodating additional hardware?

Hardware design

Start by identifying the principal elements of the system including PC, card frame, power supply, etc. as appropriate. Then itemize the input devices (such as keypads, switches and sensors), and output devices (such as motors, actuators and displays). This process may be aided by developing a diagram of the system showing the complete hardware configuration and the links which exist between the elements. This diagram will subsequently be refined and modified but initially will serve as a definition of the hardware components of the system.

Having identified the inputs required. a suitable sensor or input device should be selected for each input (see Chapter 9). It should then be possible to specify any specialized input signal conditioning required with reference to the manufacturer's specification for the sensor concerned. Input signal conditioning should then be added to the system diagram mentioned earlier.

Next, a suitable driver or output interface should be selected for each output device present (see Chapter 9). Any additional output signal conditioning required should also be specified and incorporated in the system diagram.

Software design

Software design should mirror the 'top-down' approach adopted in relation to the system as a whole. At an early stage, it will be necessary to give some consideration to the overall structure of the program and identify each of the major functional elements of the software and their relationship within the system as a whole. It is important to consider the constraints of the system imposed by time critical processes and hardware limitations (such as the size of available memory). Furthermore, routines to cope with input and output may require special techniques (e.g. specialized assembly language routines).

The software should be designed so that it is easy to maintain, modify and extend. Furthermore, the programmer should use or adapt modules ported from other programs. These modules will already have been proven and their use should be instrumental in minimizing development time.

When developing software, it is advisable to employ only 'simple logic' (i.e. that which has been tried and understood). The temptation to produce untried and over-complicated code should be avoided. Simple methods will usually produce code which is easy to maintain and debug, even if the code produced requires more memory space or executes more slowly (see Chapter 4). If the process is time critical or memory space is at a premium then code can later be refined and optimized. It is also important to consider all eventualities which may arise, not just those typical of normal operation. The following are particularly important:

- Will the system initialize itself in a safe state - will there be momentary unwanted outputs during start-up?
- What will happen if the user defaults an input or if an input sensor becomes disconnected?
- What will happen if the power fails - will the system shut-down safely?
- What input validation checks are required - what steps should be taken if an 'out of range' input is detected?

Applications

The remainder of this chapter provides details of eight representative PC-based applications. These applications are not particularly novel but they do address problems that are typical of those which face the instrumentation and control engineer. The applications have been chosen to illustrate contrasting aspects of design and, while it would be impossible to describe any of these applications in their entirety, they should provide a feel for various aspects within the process of designing and implementing a PC-based system.

Monitoring oscillator stability

The client is a manufacturer of synthesized HF radio transceivers and wishes to develop a prototype voltage-controlled oscillator (VCO) which operates in the range 40 to 60 MHz for use within the frequency-generating circuitry. Several circuits have been constructed and the client wishes to ascertain the short-term and long-term frequency stability of each unit.

Specification

The manufacturer requires that the output frequency is measured at appropriate intervals (e.g. every 100 ms for the short-term stability measurement and every l0 s for the long-term stability measurement). The results of each set of measurements are to be stored in an ASCII file for later graphical analysis. The software is, however, required to determine a number of simple performance indicators for each prototype unit including:

- Maximum frequency during the measurement period.
- Minimum frequency during the measurement period.
- Mean frequency over the measurement period.
- Total frequency drift during the measurement period.

The manufacturer also requires that the entire set of measurements and statistical calculations should be repeated at ambient temperatures of 0°C, 10°C, 20°C, 30°C and 40°C.

This task would require considerable manual effort if it were to be carried out using a conventional digital frequency meter. It is, however, an ideal candidate for automated measurement using a PC and appropriate expansion card.

Hardware

The Guide Technology GT200 Universal Counter was chosen to provide the frequency measuring facility in conjunction with a Samsung AT-compatible microcomputer which already resides in the client's RF laboratory. The GT200 takes the form of a full-size PC-compatible expansion card which is supplied together with a device driver (GT200.SYS) and virtual front panel software (VIRT.EXE) on a floppy disk. The simplified block schematic of the GT200 is shown in Figure 12.2.

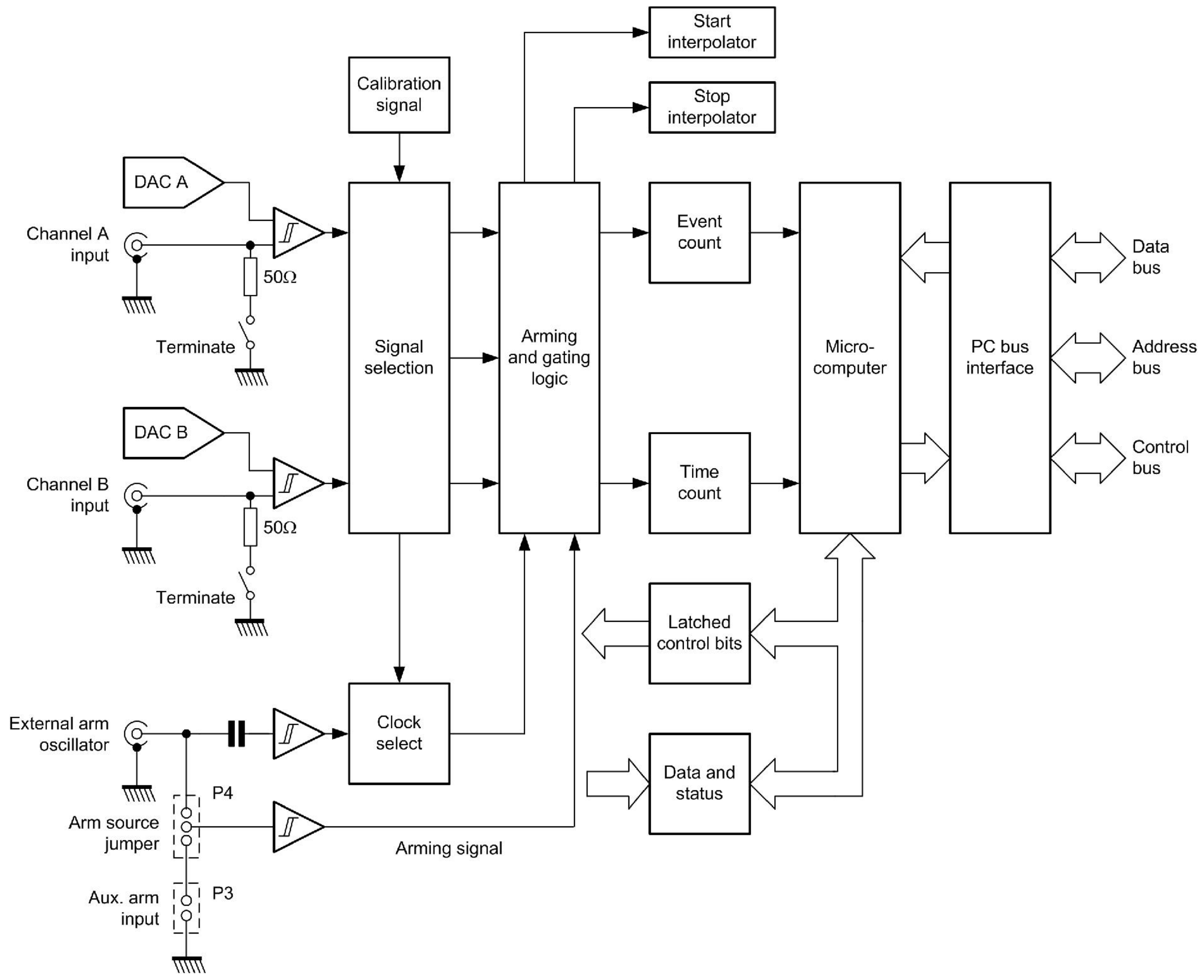


Figure 12.2 *Simplified block schematic of the GT200 digital frequency meter*

The GT200 is supplied together with a device driver (GT200.STS) and virtual front panel software (VIRT.EXE) on floppy disk. The disk also contains software which assists with setting the base address switch and includes a program which allows users to test the GT200's programming commands.

The GT200 offers a variety of measuring facilities including frequency measurement (from DC to 100 MHz, with automatic pre-scaling above about 1 MHz), fast frequency measurement (a special mode for high-speed data acquisition which allows up to 2300 measurements per second), period (both single and multiple), and time interval (i.e. the elapsed time between 'start' and 'stop' events). In addition, a direct data acquisition mode places measurements into a memory array without the usual overheads required to communicate results back to an application program via DOS).

The GT200 software is capable of performing a number of statistical functions (including mean, standard deviation, maximum and minimum measurements within a sample block). These are ideal for determining parameters such as drift and 'jitter'.

The GT200 measures input signal frequencies using the most accurate technique available, reciprocal counting coupled to time interpolation. There are two primary benefits of this method: improved accuracy and reduced measurement time. Fast measurements with high accuracy yield more information concerning the stability of a signal. The GT200 is able to compute the drift rate, mean and peak-peak jitter of a signal in the same time interval that a conventional counter is simply measuring frequency.

Software

The control program sends commands to the GT200 driver as character strings through standard DOS file write operations. Several conventions must be obeyed when incorporating commands into programs (e.g. individual commands must be separated by semicolon, carriage return or line feed delimiters). Commands are not case sensitive and may be abbreviated for convenience. The minimum acceptable abbreviations for each command are listed in the manual. As an example, FREQ may be used instead of FREQUENCY, FU instead of FUNCTION, and soon.

GT200 commands are incorporated in normal program statements such as BASIC:

```
PRINT#, "fu freqa; gate 0.01"
```

or in C:

```
fprintf(COUNTER, "fu freqa; gate 0.01");
```

A simple program, like the QBASIC program shown below, can be easily developed to meet the client's requirements for the long term stability measurement (involving 100 readings taken at 10 s intervals).

```
REM Oscillator test program
REM Declare sub-programs
DECLARE SUB max ()
DECLARE SUB min ()
DECLARE SUB mean ()
REM Dimension array for collected data
DIM freq(100)

REM Get oscillator reference
CLS
INPUT "Enter oscillator reference: "; ref$
LET ref$ = LEFT$(ref$, 6)
INPUT "Enter ambient temperature: "; temp$
osc$ = ref$ + temp$

REM Initialise digital frequency meter
OPEN "GT200$" FOR OUTPUT AS #1
OPEN "GT200$" FOR INPUT AS #2
PRINT #1, "init; timeout 4; function frequency A; gate 0.2"

REM Start collecting readings
PRINT "Hit (RETURN) to start measurement..."
WHILE r$ = ""
  r$ = INKEY$
WEND
```

```
FOR time% = 0 TO 99
PRINT #1, "reset"
'INPUT #2, freq(time%)
PRINT "Time = "; 10 * time%; " sec. Frequency = "; freq(time%); " Hz"
PRINT #1, "wait 10"
NEXT time%
CLOSE #1
CLOSE #2

REM Calculate and print statistics
PRINT
PRINT "Performance data for oscillator ref: "; ref$
PRINT
PRINT "Performance measured at: "; temp$; " deg.C"
max
PRINT "Maximum frequency; "; maxfreq; " Hz"
min
PRINT "Minimum frequency: "; minfreq; " Hz"
mean
PRINT "Mean frequency:    "; meanfreq; " Hz"
PRINT "Frequency drift:   "; maxfreq - minfreq; " Hz"
PRINT

REM Save data in an ASCII file
LET file$ = osc$ + ".DAT"
OPEN file$ FOR OUTPUT AS #3
FOR time% = 0 TO 99
PRINT #3, freq(time%)
NEXT time%
CLOSE #3

END

SUB max
SHARED freq()
SHARED maxfreq
maxfreq = 0
FOR i% = 0 TO 99
IF freq(i%) > maxfreq THEN maxfreq = freq(i%)
NEXT i%
END SUB

SUB mean
SHARED freq()
SHARED meanfreq
total = 0
FOR i% = 0 TO 99
total = total * freq(i%)
NEXT i%
meanfreq = total / 100
END SUB

SUB min
SHARED freq()
SHARED minfreq
minfreq = 1E+09
FOR i% = 0 TO 99
IF freq(i%) < minfreq THEN minfreq = freq(i%)
NEXT i%
END SUB
```

Three sub-programs, `max()`, `min()`, and `mean()` are declared at the beginning of the program. The array, `freq()`, (which will contain the returned data from the GT200 card) is then dimensioned for a total of 100 values.

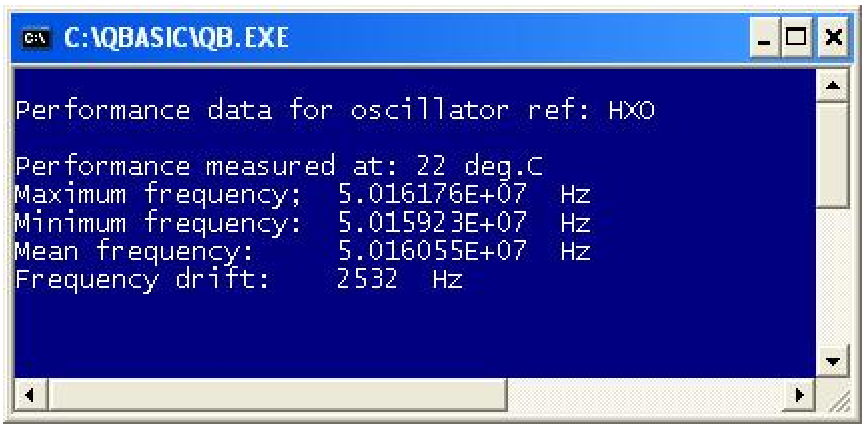
The user is then prompted to enter the oscillator reference (which is truncated to include only the first six characters) and the ambient temperature used for the measurement.

The GT200 digital frequency meter is then associated with channel I for output and 2 for input by means of the OPEN statements. The instrument is initialized to measure frequency using input A with a timeout and gate times of 4s and 0.2 s, respectively.

The program then waits for the user to indicate that he or she is ready to begin a measurement by hitting the RETURN key. Once the key has been hit, the program takes 100 readings of frequency, placing each returned reading into the `freq()` array. The time between readings is set at l0s by means of the `wait` command. Times and corresponding frequency readings are displayed on the screen on each pass through the main `FOR ... NEXT` loop so that the user is kept informed of the current state of measurement.

When the main loop has been completed, the two communications channels are closed. Thereafter, the performance data for the oscillator in question is printed with calls to the three sub-programs which determine the maximum, minimum and mean frequency values. The total frequency drift is calculated simply by subtracting the minimum frequency from the maximum frequency.

The three sub-programs, `max()`, `min()`, and `mean()`, are quite straightforward and need no comment. A typical résumé of oscillator performance (printed by the program) is shown in Figure 12.3.



```
Performance data for oscillator ref: HXO

Performance measured at: 22 deg.C
Maximum frequency;  5.016176E+07  Hz
Minimum frequency:  5.015923E+07  Hz
Mean frequency:     5.016055E+07  Hz
Frequency drift:    2532  Hz
```

Figure 12.3 *Sample printed oscillator performance data*

Finally, the data is stored in an ASCII file. Note that the filename is constructed from the concatenation of the first six (or less) characters of the oscillator reference and the ambient temperature which was entered by the user, together with the file extension, .DAT. The file is opened for output (via channel 3) and all 100 values stored in the array are written to it. The channel is then closed.

Backup battery load test

The client is a manufacturer of low-power FM radio relays. Each relay is fitted with a standby battery comprising four 2V sealed lead-acid cells, each rated at 2V 80 Ah.

Specification

The battery load test is to capture backup battery voltage data at regular intervals ranging from 10ms to 100s for periods of between 1 minute (accelerated load test) and 10 days (prolonged load test). Voltage readings are to be within the range 0V to 10V DC and they are to be accurate to within ±50 mV (±0.05V). Data is to captured in a form that is compatible with a standard spreadsheet (e.g. MS Excel) for subsequent display and analysis). A dedicated PC is unavailable for this application so the hardware interface is required to be external and available for fitting to any one of several bench PCs in the production test lab.

Hardware

This application makes use of the Measurement Computing PMD-1208LS USB Personal Measurement Device. This device (see Chapter 2) has four differential or eight single-ended analogue input channels and is easily moved from one PC to another. Screw terminals permit connection of test leads and no further adjustment or configuration is necessary other than ensuring that the software is loaded and appropriate Measurement Computing library is installed on the host computer.

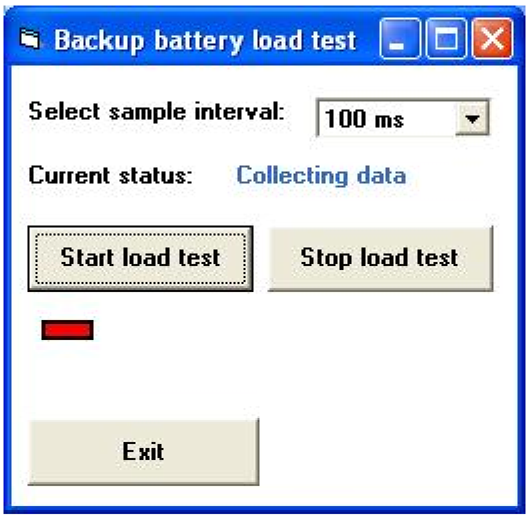
Software

The software was written using MS Visual Basic (see below) and the application is shown in Figure 12.11. A combo-box provides a means of selecting the sampling rate (from 10ms to 100s) with buttons provided to start and stop the load test. A virtual-LED and text field provides status indication. When the stop button is operated data is sent to a data file in a format that can subsequently be imported into MS Excel (see Figure 12.12).

```
'==================================================================
' Name:                 loadtest
' Purpose:              collects backup battery voltage data
' Library calls:        cbAIn%() and cbErrHandling%()
' Hardware:             PMD-1208LS USB HID
'==================================================================

Const BoardNum% = 1              ' Board number
Dim Index As Integer
Dim Record As Integer
' Dimension data array
Dim data_array(10000)
Dim Gain As Integer

Private Sub cmdExit_Click()
End
End Sub
```

```
Private Sub cmdStart_Click()
tmrConvert.Enabled = True
' Turn LED indicator on
sample_led.FillColor = "&H000000FF"
Index% = 0
lblStatus.Caption = "Collecting data"
End Sub

Private Sub cmdStop_Click()
tmrConvert.Enabled = False
' Turn LED indicator off
sample_led.FillColor = "&H00E0E0E0"
' Initial status message
lblStatus.Caption = Format$(Index, "0") + " samples collected"
' Prepare to write data file
CommonDialog1.DialogTitle = "File Save"
CommonDialog1.InitDir = App.Path
CommonDialog1.DefaultExt = "dat"
CommonDialog1.FILTER = "Data (*.dat)"
CommonDialog1.FileName = "sample.dat"
CommonDialog1.ShowSave
If CommonDialog1.FileName <> "" Then
    Open CommonDialog1.FileName For Output As #1
    For Record% = 1 To Index%
    Write #1, Record%, data_array(Record%)
    Next Record%
    Close #1
    lblStatus.Caption = "Data file written"
' Turn LED indicator green
sample_led.FillColor = "&H0000FF00"
Else
    lblStatus.Caption = "Data file not written"
' Turn LED indicator grey
sample_led.FillColor = "&H00E0E0E0"
End If
End Sub

Private Sub Form_Load()
' Declare revision level of Universal Library
ULStat% = cbDeclareRevision(CURRENTREVNUM)
' Initiate error handling
ULStat% = cbErrHandling(PRINTALL, DONTSTOP)
If ULStat% <> 0 Then Stop
' Set channel number and gain
Chan% = 0
' Set default range on start-up
Gain = BIP20VOLTS
' Set default maximum number of samples
max_samples = 100000
Index% = 0
' Disable timer
tmrConvert.Enabled = False
cmbInterval.Text = "10 ms"
' Initial status message
lblStatus.Caption = "Waiting for Start button"
' Sample LED set to off
sample_led.FillColor = "&H00E0E0E0"
' Default filename
file_name = "sample.dat"
End Sub

Private Sub tmrConvert_Timer()
Index% = Index% + 1
```

```
If cmbInterval.Text = "10 ms" Then tmrConvert.Interval = 10
If cmbInterval.Text = "100 ms" Then tmrConvert.Interval = 100
If cmbInterval.Text = "1 s" Then tmrConvert.Interval = 1000
If cmbInterval.Text = "10 s" Then tmrConvert.Interval = 10000
If cmbInterval.Text = "100 s" Then tmrConvert.Interval = 100000
' Collect the data
ULStat% = cbAIn(BoardNum%, Chan%, Gain, DataValue%)
If ULStat% = 30 Then MsgBox "Gain setting not valid", 0, "Unsupported Gain"
If ULStat% <> 0 Then Stop
ULStat% = cbToEngUnits(BoardNum%, Gain, DataValue%, EngUnits!)
If ULStat% <> 0 Then Stop
data_array(Index%) = Format$(EngUnits!, "0.00")
End Sub
```


Backup battery load test

Select sample interval: 100 ms

Current status: Collecting data

Figure 12.11 *Backup battery load test application written in Visual Basic*

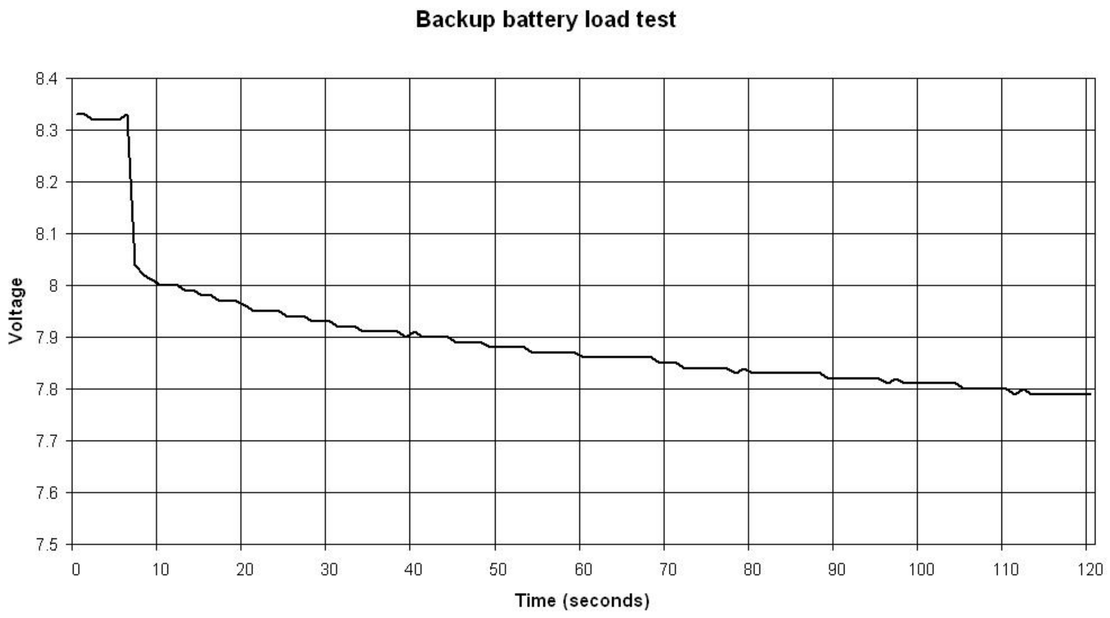


Figure 12.12 *Backup battery load test results (graph produced from data exported to MS Excel). Note that the load is applied at t = 7 seconds.*

Load sequencer

The client uses a manufacturing process based on eight devices that operate from a nominal 8A 115V AC supply. Unfortunately, the momentary surge current taken by each device (each of which involves a degaussing component) is greatly in excess of the rated current. Furthermore, when all devices operate simultaneously (or within a few milliseconds of one another) the surge current will invariably trip out the mains supply. This, in turn, causes disruption to the manufacturing process because each device has to be individually turned off before the mains trip can be manually reset. The client requires a simple and reliable means of automatically sequencing the application of power to the loads.

Specification

The time delay in applying the AC mains supply to each device is to be configurable to within one second up to a maximum of 30 seconds. The operator is to be provided with a simple graphical interface that shows the status of each load and allows the delay to be set using a simple slider control.

Hardware

Since there are eight loads and they are only required to be switched on and off, this application requires a simple 8-bit parallel port interface module. However, it is expected that the production system may be expanded at some point in the future and it could be advantageous to provide a solution that can be easily expanded on a modular basis (see Photo 12.2).

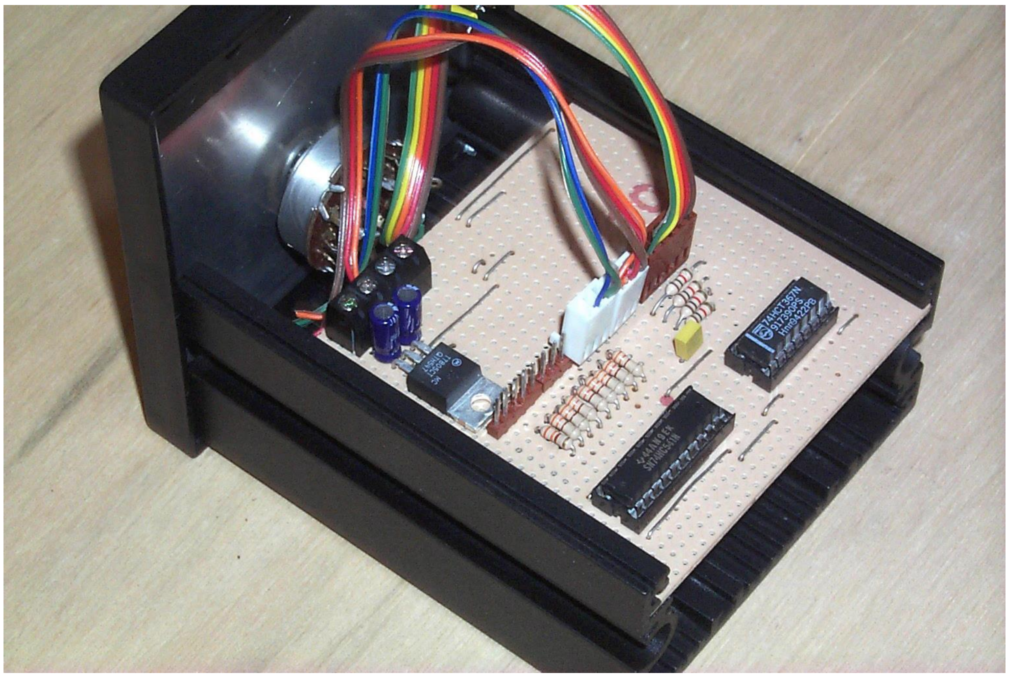


Photo 12.2 *Prototype parallel port interface module*

The circuit of one 8-bit parallel interface module is shown in Figure 12.13. The module is connected to the PC by means of a standard parallel port (see Chapter 2). In order to cater for future expansion, the module can be assigned to one of four controlled groups by means of a group channel select switch. Each channel group (A , B, C and D) will then have eight controlled channels (Channel 1 to Channel 8) and each of these Channels will correspond to a particular device. Hence the system has potential for controlling up to 32 devices using four identical interface modules.

Each channel output from the interface module is connected to a relay driver (see Figure 12.14). This circuit is capable of switching a load of up to 10A at 115V AC. A status LED is included for test purposes.

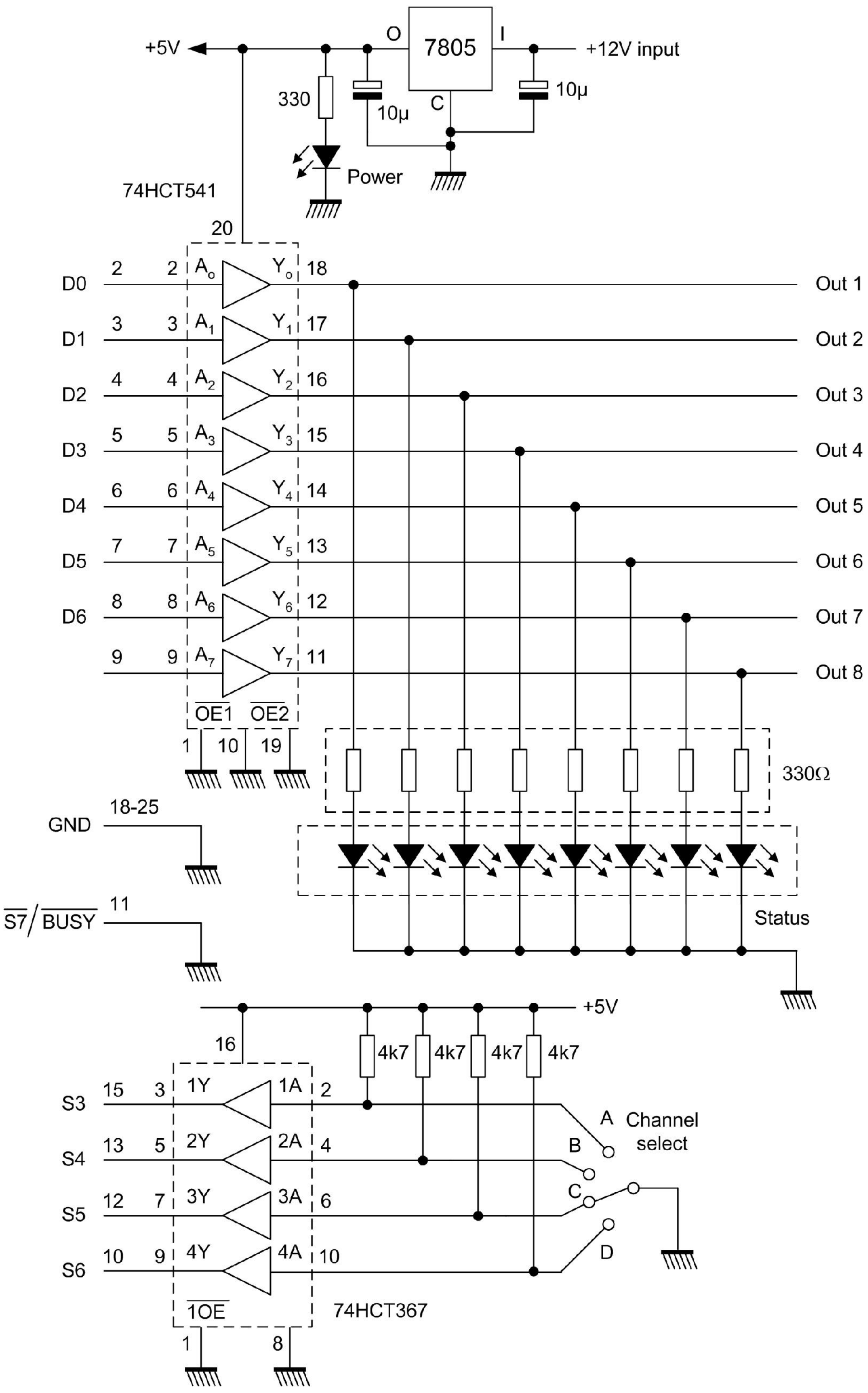


Figure 12.13 *Parallel port I/O interface*

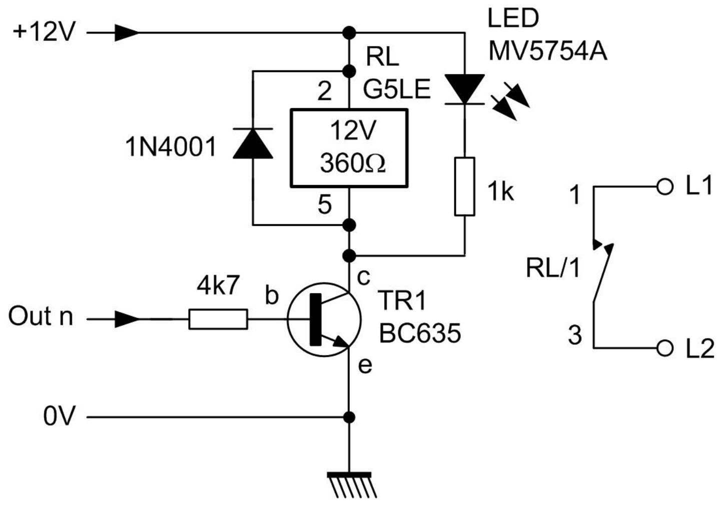


Figure 12.14 *Relay driver (a high output from a port line enables the load)*

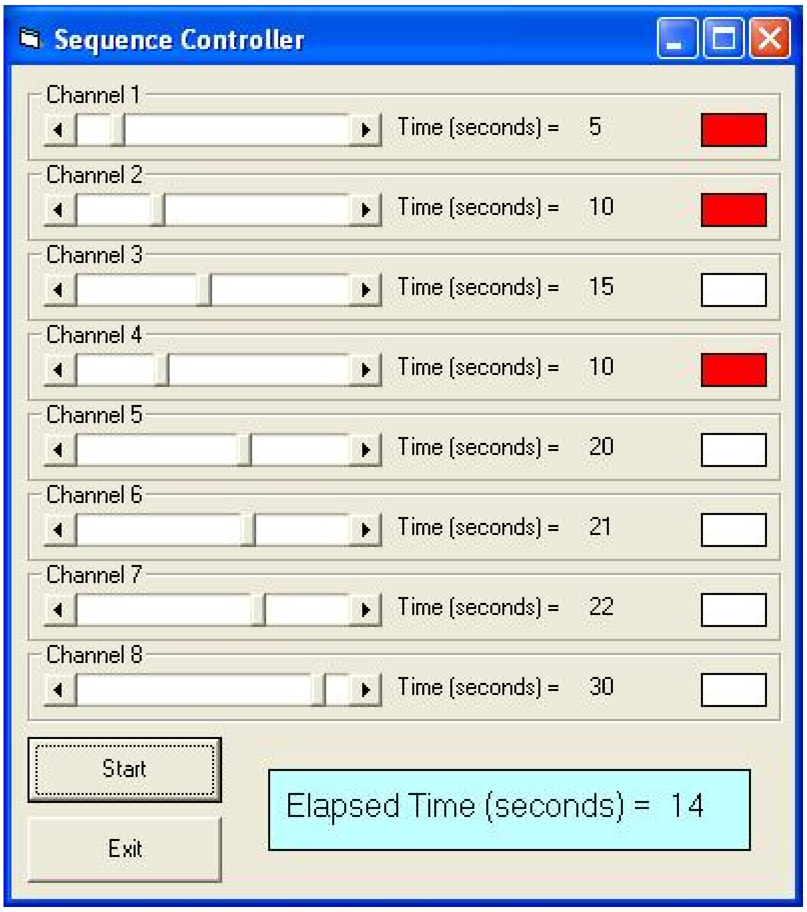
Software

The application uses a simple Visual Basic routine (see below). The user interface is designed so that the operator can set the delay on any channel to any time between zero and 30 seconds using a simple slider control. Each channel is fitted with a virtual LED indicator so that the operator knows which loads have become active. In addition, a further status field shows the elapsed time (see Figure 12.15). This application makes extensive use of the Visual Basic Timer (Chapter 6 contains more information on Visual Basic programming).

```
'================================================================
' Name:          seqcon2
' Purpose:       controls switching sequence on channels 1 to 8
' Library calls: requires inpout32.bas for I/O
' Hardware:      parallel port with relay modules
'================================================================

Dim Port1 As Integer
Dim Port2 As Integer
Dim Port3 As Integer
Dim OutData As Integer
Dim ETime As Integer

Private Sub Start_Click()
Timer1.Enabled = True
Timer2.Enabled = True
Timer3.Enabled = True
Timer4.Enabled = True
Timer5.Enabled = True
Timer6.Enabled = True
Timer7.Enabled = True
Timer8.Enabled = True
MasterClock.Enabled = True
End Sub

Private Sub Exit_Click()
End
```

```
End Sub

Private Sub Form_Load()
Port1 = 888
Port2 = 889
Port3 = 890
OutData = 0
Out Port1, OutData
SetTime1.Caption = 10
SetTime2.Caption = 10
SetTime3.Caption = 10
SetTime4.Caption = 10
SetTime5.Caption = 10
SetTime6.Caption = 10
SetTime7.Caption = 10
SetTime8.Caption = 10
Timer1.Interval = 10000
Timer2.Interval = 10000
Timer3.Interval = 10000
Timer4.Interval = 10000
Timer5.Interval = 10000
Timer6.Interval = 10000
Timer7.Interval = 10000
Timer8.Interval = 10000
ETime = 0
End Sub

Private Sub HScroll1_Change()
Timer1.Interval = HScroll1.Value
SetTime1.Caption = Int(Timer1.Interval / 1000)
End Sub

Private Sub HScroll2_Change()
Timer2.Interval = HScroll2.Value
SetTime2.Caption = Int(Timer2.Interval / 1000)
End Sub

Private Sub HScroll3_Change()
Timer3.Interval = HScroll3.Value
SetTime3.Caption = Int(Timer3.Interval / 1000)
End Sub

Private Sub HScroll4_Change()
Timer4.Interval = HScroll4.Value
SetTime4.Caption = Int(Timer4.Interval / 1000)
End Sub

Private Sub HScroll5_Change()
Timer5.Interval = HScroll5.Value
SetTime5.Caption = Int(Timer5.Interval / 1000)
End Sub

Private Sub HScroll6_Change()
Timer6.Interval = HScroll6.Value
SetTime6.Caption = Int(Timer6.Interval / 1000)
End Sub

Private Sub HScroll7_Change()
Timer7.Interval = HScroll7.Value
SetTime7.Caption = Int(Timer7.Interval / 1000)
End Sub

Private Sub HScroll8_Change()
Timer8.Interval = HScroll8.Value
```

```
SetTime8.Caption = Int(Timer8.Interval / 1000)
End Sub

Private Sub Timer1_Timer()
Shape1.FillColor = "&H000000FF"
OutData = Inp(Port1)
Out Port1, (OutData Or 1)
End Sub

Private Sub Timer2_Timer()
OutData = Inp(Port1)
Out Port1, (OutData Or 2)
Shape2.FillColor = "&H000000FF"
End Sub

Private Sub Timer3_Timer()
OutData = Inp(Port1)
Out Port1, (OutData Or 4)
Shape3.FillColor = "&H000000FF"
End Sub

Private Sub Timer4_Timer()
OutData = Inp(Port1)
Out Port1, (OutData Or 8)
Shape4.FillColor = "&H000000FF"

End Sub

Private Sub Timer5_Timer()
OutData = Inp(Port1)
Out Port1, (OutData Or 16)
Shape5.FillColor = "&H000000FF"
End Sub

Private Sub Timer6_Timer()
OutData = Inp(Port1)
Out Port1, (OutData Or 32)
Shape6.FillColor = "&H000000FF"
End Sub

Private Sub Timer7_Timer()
OutData = Inp(Port1)
Out Port1, (OutData Or 64)
Shape7.FillColor = "&H000000FF"
End Sub

Private Sub Timer8_Timer()
OutData = Inp(Port1)
Out Port1, (OutData Or 128)
Shape8.FillColor = "&H000000FF"
End Sub

Private Sub MasterClock_Timer()
ETime = ETime + 1
Clock.Caption = ETime
End Sub
```



Channel 1 Time (seconds) = 5 🟥
Channel 2 Time (seconds) = 10 🟥
Channel 3 Time (seconds) = 15 ⬜
Channel 4 Time (seconds) = 10 🟥
Channel 5 Time (seconds) = 20 ⬜
Channel 6 Time (seconds) = 21 ⬜
Channel 7 Time (seconds) = 22 ⬜
Channel 8 Time (seconds) = 30 ⬜

Elapsed Time (seconds) = 14

Figure 12.15 *Sequence controller display 14 seconds into the power-up sequence*

Icing flow tunnel

A college department is engaged in research into the effectiveness of various methods of aircraft deicing based on the application of anti-icing fluids. The department has a wind tunnel capable of speeds of up to 80 m/s (Mach 0.28) supplied by a fan driven by a 10 HP variable speed DC motor. The test section can be adjusted through a pitch angle of ± 20 º and instrumentation can be attached to parts and components mounted in this section. The moving air stream is cooled by means of a refrigerated cooling unit such that airflow temperatures of between -18ºC and ambient can be produced.

Specification

The system must provide control for the variable speed fan motor (to an accuracy of ± 1 m/s), air temperature (to an accuracy of ± 1ºC), and pitch angle (to an accuracy of ± 2 º). The system is to provide a graphical display of the controlled variables with digital readout of the controlled variables. In addition, an on/off spray control is to be provided. The system is to be controlled from a low-cost dedicated PC controller. The test component is to be fitted with temperature sensors so that differential readings are available for display. The overall range of differential measurement is to be from 0ºC to 20ºC with a resolution of better than 0.5% of reading and an accuracy of better than ± 0.5ºC.

Software

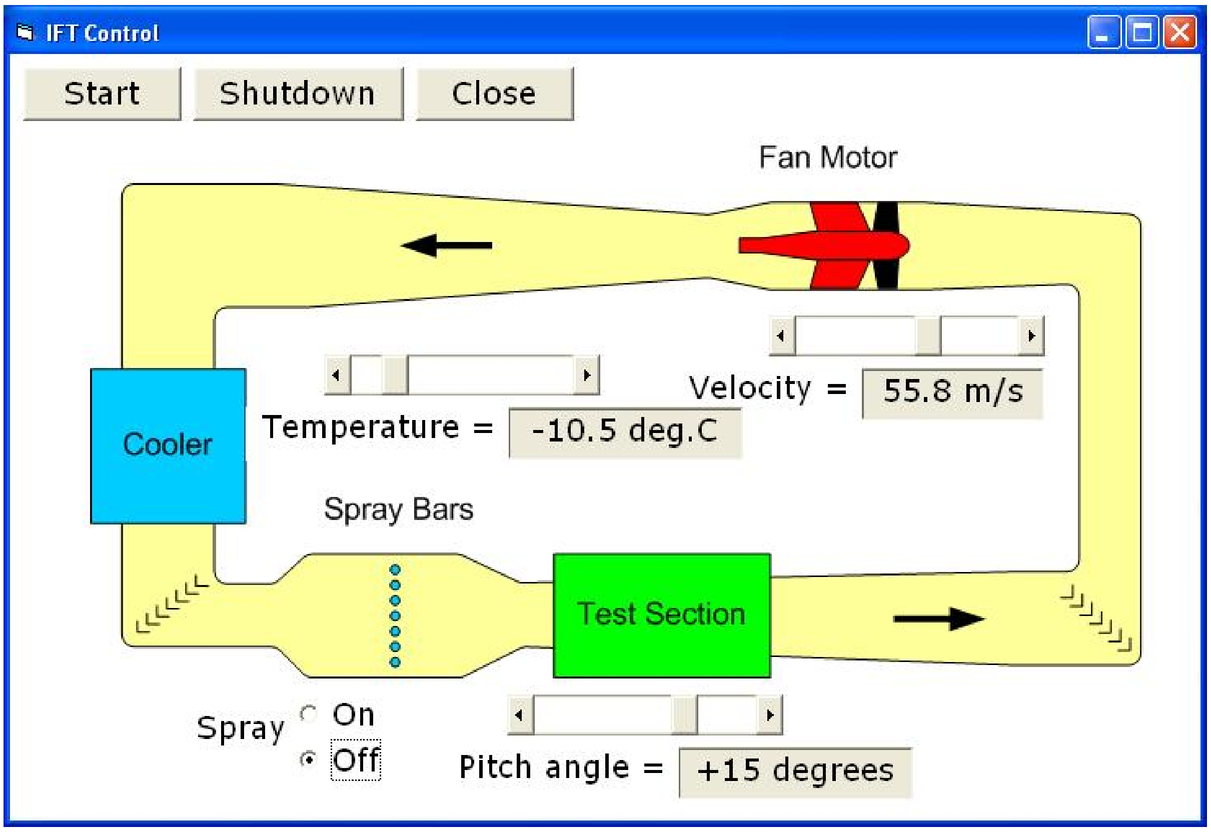
Two Visual Basic 6 applications are used concurrently in this application. Visual Basic 6 was chosen for the software development because of the ease of creating visual controls and because the language was already being used extensively within the department. One of the Visual Basic applications provides control for the Ice Flow Tunnel (IFT Control) whilst the other (TDC Control) is responsible for collecting data from the AD590 differential sensing arrangement and then storing this for later analysis. The graphical displays were first produced using MS Visio and then imported into the Visual Basic forms. The Visual Basic controls were then superimposed.

The IFT Control application provides slider controls for setting the air temperature, velocity, and pitch angle of the component on test (see Figure 12.22).

Hardware

The PC is fitted with an ISA I/O card which has eight analogue inputs and two analogue outputs. A further ISA I/O card provides 48 digital I/O lines arranged in six groups of eight.

The variable speed drive (VFD) for the 10 HP fan motor requires an input of 10V DC for frequency adjustment (over the range 0.1 to 400Hz) and an airflow sensor is used to determine the actual air velocity produced. The sensor produces an output of 0 to 10V. The speed control thus requires an analogue input port and an analogue output port. Both ports are to a full-scale range of 0 to 10V.



Temperature = -10.5 deg.C

Velocity = 55.8 m/s

Spray On [✓ Off]

Pitch angle = +15 degrees

Figure 12.22 *Ice flow tunnel (IFT) control application (the form was created using MS Visio and Visual Basic)*

The refrigeration unit is controlled with on/off digital control using a dedicated I/O card and a comparator controlled from a DC 0 to 5V control signal. A temperature sensor is suspended in the airflow output from the refrigeration unit and a signal (10mV/ºC) is fed to a further analogue input port.

The pitch angle control uses a digital output port with a stepper motor (see Chapter 9) and the spray bar control uses a single bit on a further digital output port to provide simple on/off control.

Temperature sensing within the test section is based on a differential sensing arrangement with pairs of AD590 temperature sensors (see Chapter 9). The AD590 is well suited to this application as it offers excellent linearity (better than ± 3°C over the entire range) and the ability to operate well in remote sensing applications with simple twisted pair connections. Lead wire compensation, filters and circuits to ensure linearity are unnecessary with this type of sensor.

The output voltage from the differential sensing arrangement (see Figure 12.23) is 20mV/°C for every one degree difference in temperature. Hence an output of 100mV will result from a temperature difference of 5°C. Additional signal gain is applied within the analogue input card.

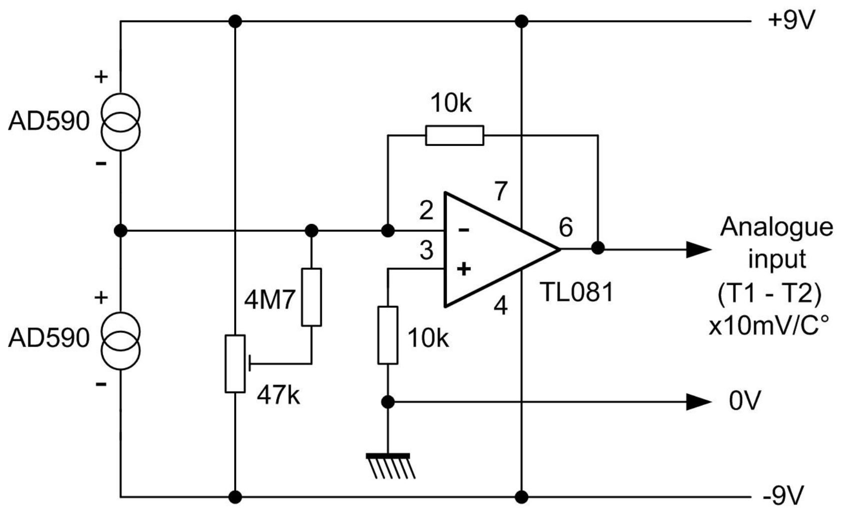


Figure 12.23 *Arrangement of AD590 temperature sensors for differential measurements*

Please note that several further applications (including strain measurement, environmental monitoring, audio recording and production tetsing) are described in the full book version of Chapter 12