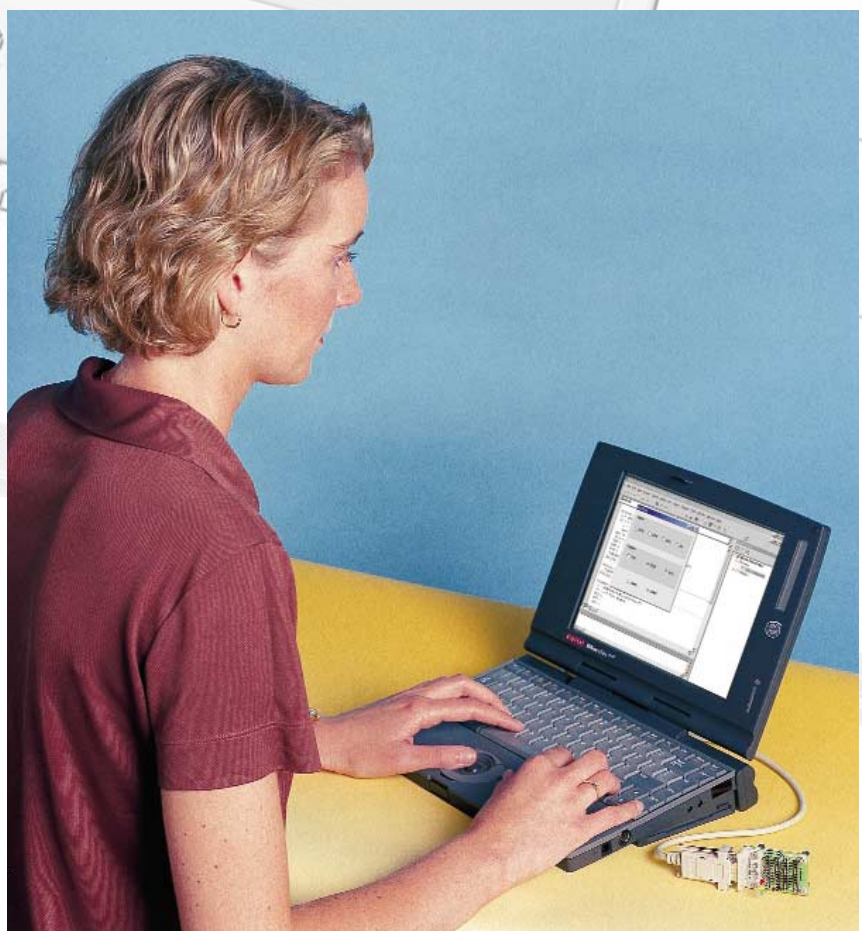
PC Serial Peripheral Design (I)

Part 1: Introduction, course hardware, first program

By B. Kainka

At one time or another many readers may have thought about controlling or monitoring equipment using their PC. Electronics projects using a PC need not be lavish or expensive: often the interfaces provided on the PC can be used directly. In this series of articles we present a range of projects using the serial (RS232) port, controlled using simple programs written in Visual Basic.



Why have we chosen to use the serial port, when it is more complicated and offers fewer connections than the printer port? For a beginners' course using the serial port has several advantages:
- The serial port is well protected against accidental damage. Cables can safely be plugged in while the computer is on.
- There is usually a spare serial port which can be used for experiments.
- The serial port can deliver enough current to power a wide range of projects, and so a separate power supply is not required.

This course is also easy on the pocket, which is important, not least for educational establishments and those on youth employment programmes. Apart from the small printed circuit board, you will need just a few everyday components, such as pushbuttons, NPN transistors, resistors, LEDs, capacitors, diodes and an LDR. All the connections of the serial port are brought out to sockets on the circuit board and there is also a small prototyping area for building experimental circuits.

All the projects are programmed in **Visual BASIC 5**. The programs (available on floppy disk, order code **000074-11**, or for download from *www.elektor-electronics.co.uk*) are clearly commented, so that you can easily

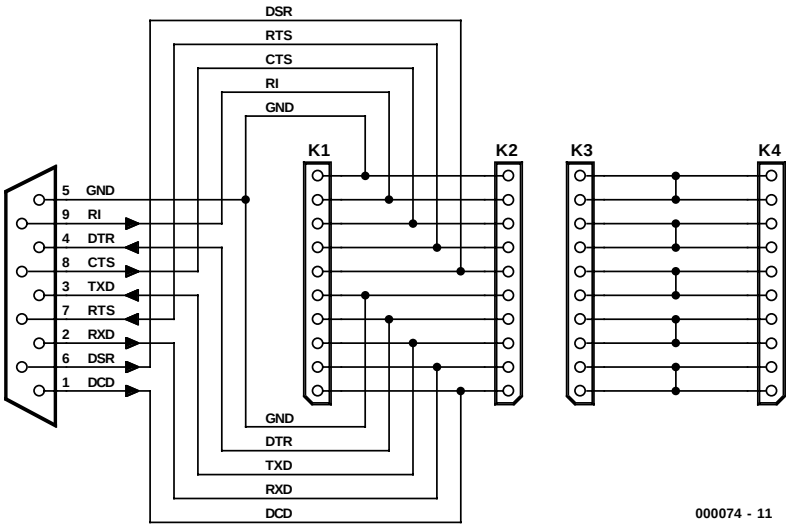


Figure 1. Circuit diagram of the experimental PCB.

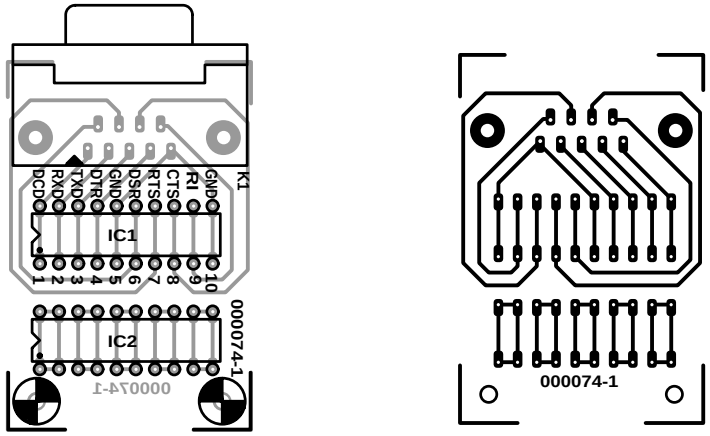


Figure 2. Component layout for the experimental PCB.

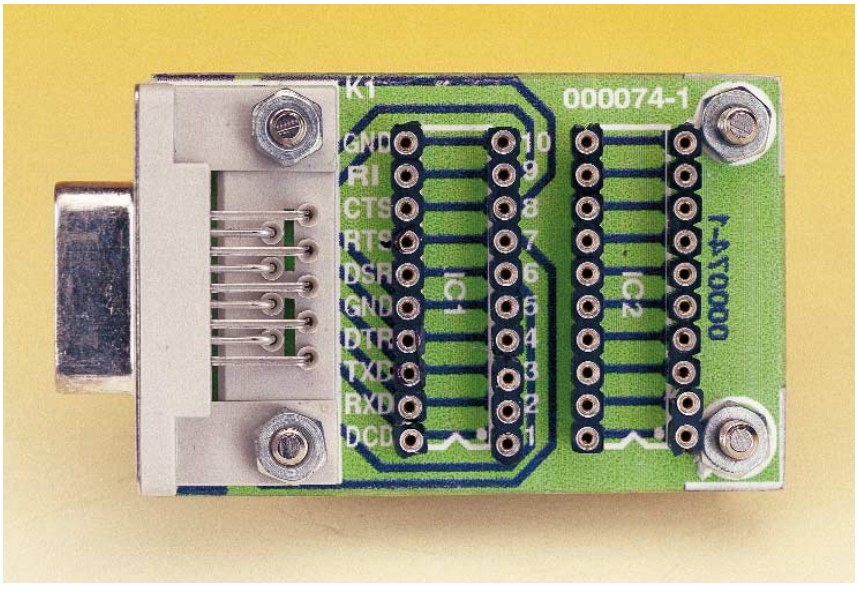


Figure 3. The assembled board.

change them to try out your own ideas. In later instalments, we will deal with complex topics such as real-time control and connection to external circuitry. As a bonus, readers will be introduced to applications of various sensor technologies, which arise naturally through the course.

The printed circuit board

The projects are based on a simple circuit board. As the circuit in **Figure 1** shows, it is a small prototyping board fitted with four SIL socket strips and a 9-way sub-D socket. Two of the four SIL sockets are connected to the sub-D socket, while the other two are simply connected to one another.

Figure 2 shows the component layout for the circuit board, and the assembled board is shown in **Figure 3**. **Table 1** gives a summary of the pinout of the 25-way and 9-way connectors normally employed for serial ports, showing the names and functions of all the signals that make up the interface. A male connector is invariably provided on the PC side, and so we require a female connector: a 9-way extension cable can be used to connect the PC to the circuit board described here. If your PC is equipped with a 25-way connector, you will need to use a suitable adapter.

Data is normally transferred over the serial interface using the TXD (transmit data) and RXD (receive data) signals. The other signals have auxiliary functions concerned with setting up and controlling data transfer. They are known as 'handshake signals', because they are used to acknowledge transfer of data between two pieces of equipment. A particularly useful feature of the handshake signals is that their state can be read or written directly.

The pin labelling on the circuit board follows that of the 9-way sub-D plug. Each pin of this plug is taken to two SIL socket pins, except for GND, which is taken to four. The small prototyping area on the circuit board is divided into five groups of connections with four SIL socket pins in each. Four SIL sockets with turned pins are used in total; two 20-pin DIL sockets — which are easier to obtain — can be used instead.

Visual BASIC

You will need a copy of Visual Basic, version 5 or 6, for this course. All source code is available from the *Elektor Electronics* website in VB5 format. The programs can also be loaded and compiled without modification using VB6. If you do not have a copy of Visual Basic

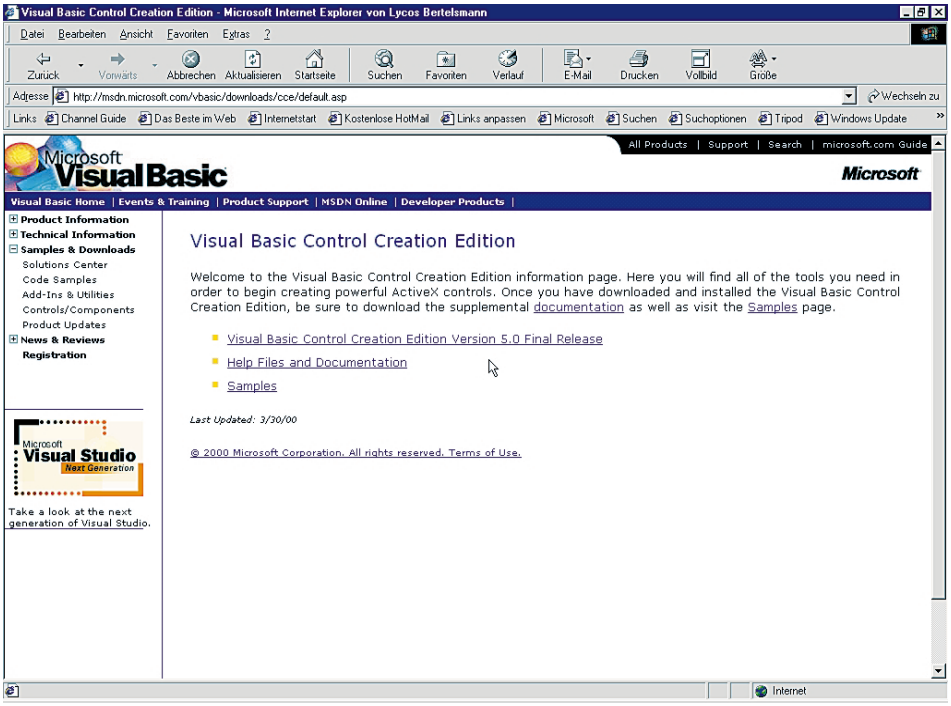


Figure 4. Visual Basic (VB5CCE) is available for download here.

and balk at its price, you can obtain a free version of VB5 from Microsoft: 'Visual Basic Control Creation' VB5CCE is available on the Internet at

http://msdn.microsoft.com/vbasic/downloads/cce/default.asp

Our first program: I/O test

And now, at last, to our first program. It's a simple test program which gives direct access to all serial port connections (except RXD). The three outputs can be toggled with a click of the mouse, while the four inputs are read and displayed. The program is configured to use COM2 by default; however, it will recognise when COM2 is in use and automatically attempt to use COM1 instead. It is assumed that most PCs have a mouse connected to COM1 and that COM2 is unused. Frequently the connectors on PCs are not correctly labelled: in that case this program can help to identify them correctly. If you have two spare COM ports, you can use COM1 or COM2 as you prefer.

What is 'ON', and what is 'OFF'? If you connect a voltmeter between DTR and GND (ground), you will find that 'ON' gives a positive voltage of about 10 V, while 'OFF' gives a voltage of about –10V with respect to GND. Now connect a wire between, for example, the DTR output and the DSR input. As soon as DTR is turned

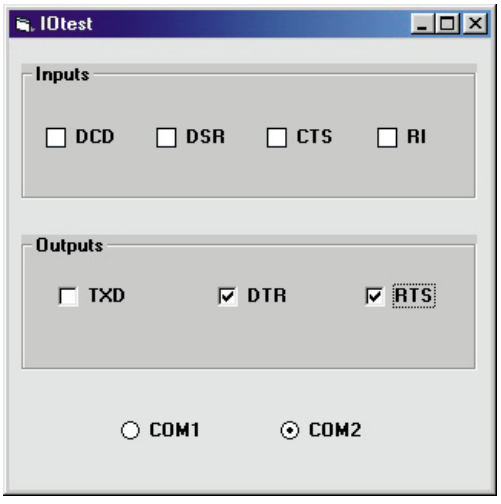
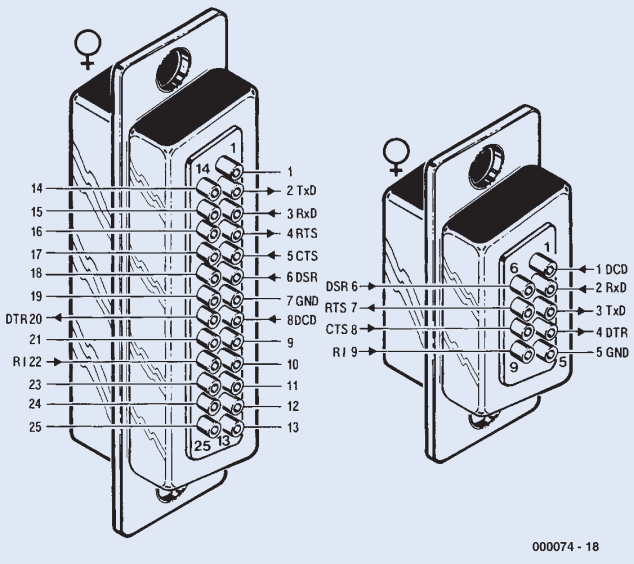


Figure 5. Window showing direct access to serial port signals.

Table 1 Connector pinouts and signal names

Pin 25-way	Pin 9-way	Input/ Output	Symbol	Function
2	3	Out	TxD	Transmit Data
3	2	In	RxD	Receive Data
4	7	Out	RTS	Request To Send
5	8	In	CTS	Clear To Send
6	6	In	DSR	Data Set Ready
7	5		GND	Ground
8	1	In	DCD	Data Carrier Detect
20	4	Out	DTR	Data Terminal Ready
22	9	In	RI	Ring Indicator



on, the DSR input reports 'ON' too. Our first practical application is to poll a switch (**Figure 6**). In order to read the state of the switch, the DTR signal must be turned on. Now we have a complete PC application: the switch can be placed in some remote location and it state will be relayed to the PC. What that might represent is up to you: you might, perhaps, want to monitor whether that mousetrap in the cellar has been set off.

We round off this first instalment with another neat little application. Here a light-emitting diode is connected to and controlled by the PC (**Figure 7**). Many *Elektor Electronics* readers will protest loudly that one should never connect an LED without a current-limiting resistor. Here, though, it is not necessary, as a resistor is effectively built into the PC.

Another protest: in the datasheets, the maximum allowable reverse voltage of an LED is given as 3 V or 5 V. In this circuit, however, reverse voltages of up to 10 V can appear across the LED. Experience has shown that this is not in practice

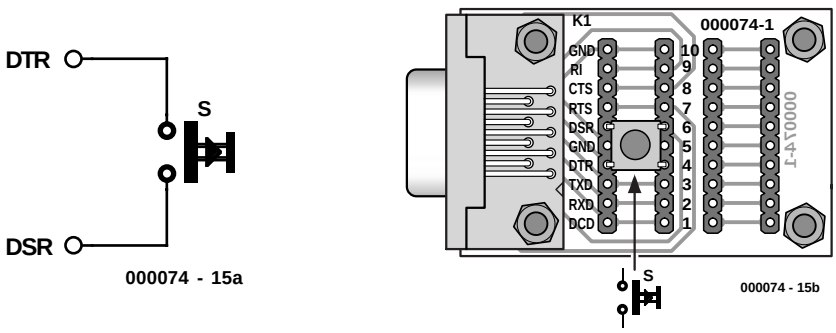


Figure 6. A simple switch is connected for the first experiment.

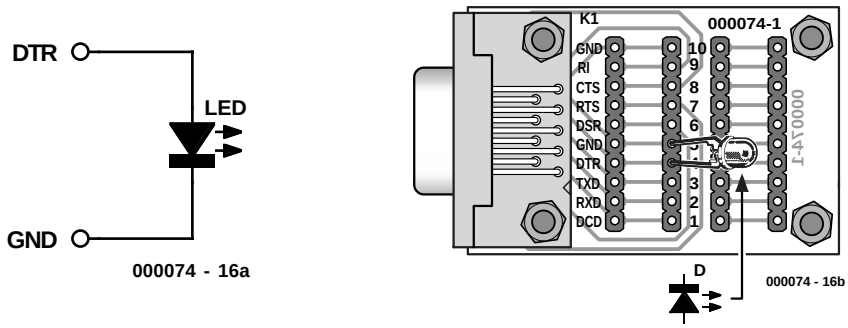


Figure 7. Equally simple: an LED is connected for the second project.

On Project Disk #000074-II:

PORTS.BAS	845	03-28-00	5:31p
ANPEL.FRX	1,094	04-10-00	6:01p
BLINK.FRX	1,094	04-10-00	6:43p
COUNTER1.FRX	1,094	05-04-00	6:49p
COUNTER2.FRX	1,094	05-04-00	7:14p
CONTENTS.TXT	0	06-15-00	9:35a
IOTEST.FRX	1,094	05-04-00	12:21p
AMPEL.VBP	497	04-10-00	6:01p
BLINK.VBP	497	04-10-00	6:43p
COUNTER1.VBP	499	05-04-00	6:49p
COUNTER2.VBP	499	05-04-00	7:14p
IOTEST.VBP	497	05-04-00	1:13p
AMPEL.VBW	81	05-05-00	12:57p
BLINK.VBW	80	05-05-00	4:26p
COUNTER2.VBW	80	05-05-00	4:27p
IOTEST.BW	80	06-08-00	1:15p
PORT.DLL	46,080	02-07-99	1:15p
BLINK.FRM	3,488	04-10-00	6:43p
COUNTER1.FRM	3,422	05-04-00	6:49p
COUNTER2.FRM	6,683	05-04-00	7:14p
IOTEST.FRM	5,317	05-04-00	12:21p
ANPEL.FRM	3,669	04-10-00	6:01p
22 file(s)	77,784 bytes		

a problem. Even with a reverse voltage of 20 V no significant current flows in most LEDs. But we should provide a 'clean' solution: an additional silicon diode (for example, a 1N4148) will protect the LED from excessive reverse voltages. The diode can be connected either in anti-parallel or in series with the LED (**Figure 8**). As a rule of thumb, for serious projects the reverse voltage should be limited to 5 V; for experimental purposes we can allow ourselves a little latitude.

(000074-1)

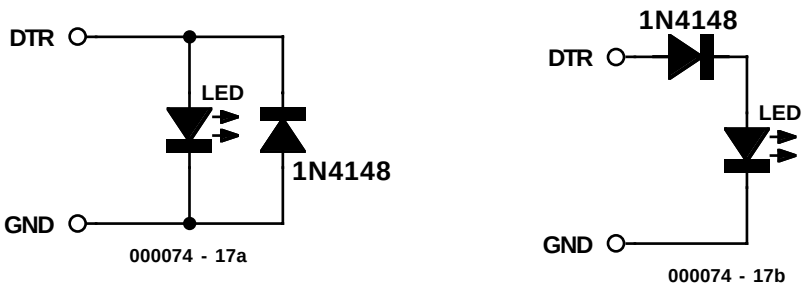
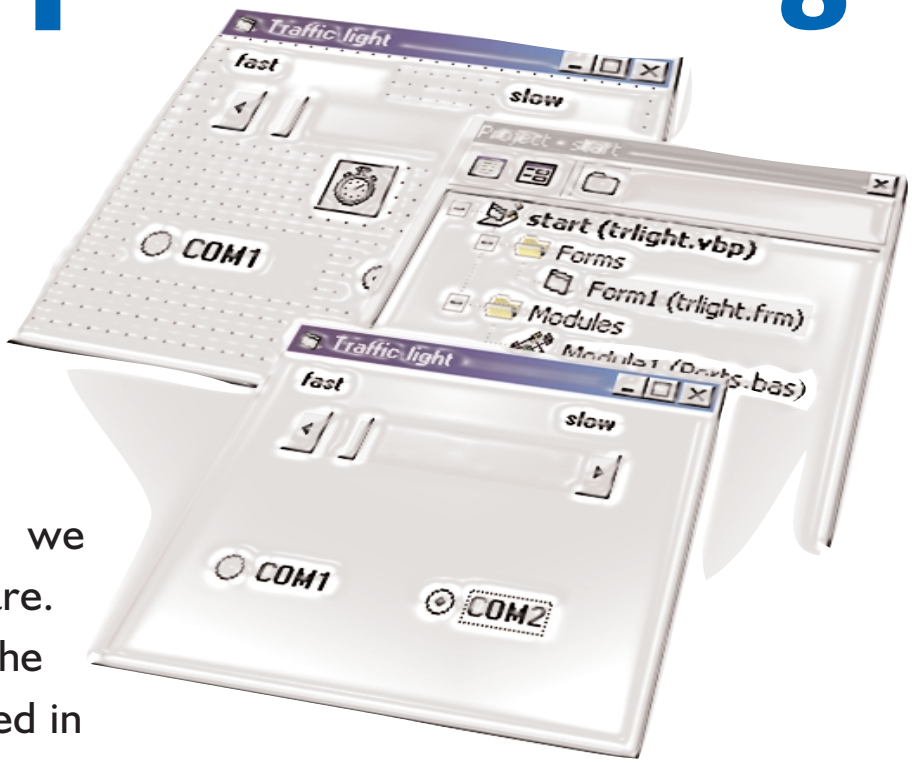


Figure 8. Two ways to protect the LED from excessive reverse voltage.

PC Peripheral Design (2)

Part 2: Software



By B. Kainka

In this second part we concentrate on software. We take a closer look at the IOtest program presented in the first article and use simple program examples in Visual Basic to control a model traffic light and a clock generator.

In the first article in this series we introduced the IOtest program. The majority of readers with some programming experience will be interested in how we access the serial interface using Visual Basic. Here we will use a library of program subroutines called PORT.DLL written by H.-J.Berndt. This DLL file is available from the Free Downloads section of the *Elektor Electronics* website at http://www.elektor-electronics.co.uk. For those with no access to the Internet, the file is also found on the diskette containing the course software (see Readers Services pages).

In Visual Basic, all the procedures and functions defined in the DLL are declared and must be in an external module. In our case this is called PORTS.BAS (**Listing 1**).

The experienced (Visual Basic) user will probably recognise the most commonly used routines for serial communications, e.g., OPENCOM which is used to open the interface for communication and indicates that it is available to the program. SENDBYTE and READBYTE are used to transfer data over the serial interface. For our purposes here, the more important routines are those that access the control and status lines of the interface. The procedures DTR, RTS and TXD control these output signals and CTS, DSR, RI and DCD read the state of these input signals. One input line that you cannot read directly is RXD. This is because it would normally be used to carry the received serial data. Also the PORT.DLL file contains routines for time measurement that we will be using later, as well as many other functions for the other PC interfaces (Parallel port, Joystick, Sound and Video).

I/Otest

The application program I/Otest is shown in **Listing 2**.

The essential core of the program is the timer procedure `Timer1_Timer`, which is automatically called at predetermined intervals. The check boxes 1 to 4 are acti-

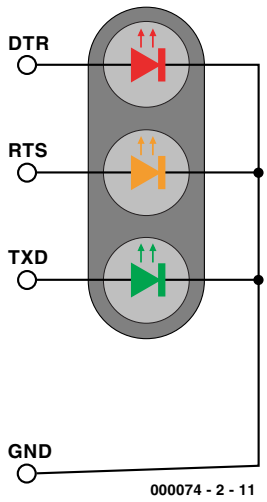


Figure 1. The traffic light LEDs.

Listing 1

Declarations for PORT.DLL

```
Declare Function OPENCOM Lib "Port" (ByVal A$) As Integer
Declare Sub CLOSECOM Lib "Port" ()
Declare Sub SENDBYTE Lib "Port" (ByVal b%)
Declare Function READBYTE Lib "Port" () As Integer
Declare Sub DTR Lib "Port" (ByVal b%)
Declare Sub RTS Lib "Port" (ByVal b%)
Declare Sub TXD Lib "Port" (ByVal b%)
Declare Function CTS Lib "Port" () As Integer
Declare Function DSR Lib "Port" () As Integer
Declare Function RI Lib "Port" () As Integer
Declare Function DCD Lib "Port" () As Integer
Declare Sub DELAY Lib "Port" (ByVal b%)
Declare Sub TIMEINIT Lib "Port" ()
Declare Sub TIMEINITUS Lib "Port" ()
Declare Function TIMEREAD Lib "Port" () As Long
Declare Function TIMEREADUS Lib "Port" () As Long
Declare Sub DELAYUS Lib "Port" (ByVal l As Long)
Declare Sub REALTIME Lib "Port" (ByVal i As Boolean)
```

Listing 2

User program IOtest

```
Private Sub Form_Load()
 i = OPENCOM("COM2,1200,N,8,1")
 If i = 0 Then
    i = OPENCOM("COM1,1200,N,8,1")
    Option1.Value = True
 End If
 If i = 0 Then MsgBox ("COM Interface Error")
 TXD 1
 RTS 1
 DTR 1
 TIMEINIT
End Sub

Private Sub Form_Unload(Cancel As Integer)
  CLOSECOM
End Sub

Private Sub Option1_Click()
 i = OPENCOM("COM1,1200,N,8,1")
 If i = 0 Then MsgBox ("COM1 not available")
 TXD 1
 RTS 1
 DTR 1
End Sub

Private Sub Option2_Click()
 i = OPENCOM("COM2,1200,N,8,1")
 If i = 0 Then MsgBox ("COM2 not available")
 TXD 1
 RTS 1
 DTR 1
End Sub

Private Sub Timer1_Timer()
  Check1.Value = CTS()
  Check2.Value = DSR()
  Check3.Value = DCD()
  Check4.Value = RI()
  If Check5.Value Then TXD 1 Else TXD 0
  If Check6.Value Then DTR 1 Else DTR 0
  If Check7.Value Then RTS 1 Else RTS 0
End Sub
```

vated and will show a tick when the corresponding input line is switched to a '1'. The three output lines are switched when the user activates the corresponding check box.

The other parts of the program are used to select and open the interface and we will look at them a little later. We can see that the interface will be initialised with a communication rate of 1200 baud, no parity bit, 8 data bits and 1 stop bit. For our purposes here, these communication parameters are entirely superfluous because we are only interested in the control lines that form part of the interface. Windows will however configure the port as if it were to communicate with, for example, a modem. But all we need to do here is to read the input status lines and write to the output control lines.

Traffic Light controller

If you are new to Visual Basic it is a good idea to choose some hardware that you can actually see working. For this reason a model traffic light with three LEDs was chosen (**Figure 1**). The effects of current limiting and reverse voltage on the LEDs were discussed in the first article of this series.

The model lamp is well suited to experimenting for our first excursion into programming with Visual Basic. For the program development you will need to start with a new blank form, and using the mouse, select and drag graphic control elements from the list of tools onto the form (**Figure 2**). The size of each element and its position can be easily altered. Each element has a complete range of properties that can be attached to it, including size, colour, text and much more. Those that you are not sure of yet can be simply left as they are. For our application we use the following elements:

- Two labels with the properties (the text) Caption = "fast" and "slow" respectively.

- A horizontal slider or scroll bar (HScrollBar) with the properties Min = 50, Max = 500 and Position =100
- A timer (Timer) with the property Interval = 100, i.e. 100 ms

- Two Option Buttons with the Caption = "COM1" and "COM2" respectively, the button for COM2 is "true"
- The form itself contains the Caption = "Traffic Light"

Figure 3 gives an overview of the project. The aforementioned module PORTS.BAS is also included in the project and contains all the

Listing 3

Traffic lights program

```
Dim Time As Integer

Private Sub Form_Load()
 i = OPENCOM("COM2,1200,N,8,1")
 If i = 0 Then
    i = OPENCOM("COM1,1200,N,8,1")
    Option1.Value = True
 End If
 If i = 0 Then MsgBox ("COM Interface Error")
 TXD 0
 RTS 0
 DTR 0
 Time = 0
End Sub

Private Sub Form_Unload(Cancel As Integer)
  CLOSECOM
End Sub

Private Sub HScroll1_Change()
  Timer1.Interval = HScroll1.Value
End Sub

Private Sub Option1_Click()
 i = OPENCOM("COM1,1200,N,8,1")
 If i = 0 Then MsgBox ("COM1 not available")
 TXD 1
 RTS 1
 DTR 1
End Sub

Private Sub Option2_Click()
 i = OPENCOM("COM2,1200,N,8,1")
 If i = 0 Then MsgBox ("COM2 not available")
 TXD 1
 RTS 1
 DTR 1
End Sub

Private Sub Timer1_Timer()
  Time = Time + 1
  If Time = 1 Then red
  If Time = 40 Then redyellow
  If Time = 50 Then green
  If Time = 90 Then yellow
  If Time = 100 Then Time = 0
End Sub

Sub red()
RTS 1
DTR 0
TXD 0
End Sub

Sub redyellow()
RTS 1
DTR 1
TXD 0
End Sub

Sub yellow()
RTS 0
DTR 1
TXD 0
End Sub

Sub green()
RTS 0
DTR 0
TXD 1
End Sub
```

necessary declarations for PORT.DLL. It needs to linked with the software in each of the projects so that they can all have access to the procedures and functions that are needed to control the serial interface.

In a Visual Basic program, events are produced by procedures that are isolated from each other. The overall flow of the program is controlled by Windows. The procedure 'Private Sub Form_Load()' will be called right at the beginning of the program (**Listing 3**) This procedure contains all the instructions needed to initialise the serial interface. In this case, the serial interface will be open and all the outputs of the interface will be switched off. Lastly, the global variable 'Time' is reset to zero.

The function OPENCOM in PORT.DLL returns a value indicating if the interface has been successfully opened. It will not be possible to open it if it is already in use by another program. Initially the program will use this function to open COM2. If the return value from this function is 0 i.e. it is busy then COM1 will be opened. This will be displayed on the screen with the value of option button 1 (COM1) 'True'. If both COM1 and COM2 are busy then a failure message will pop up in a MessageBox.

In the normal course of events the program will open COM2. If however you want to use COM1 then you can point to the corresponding button and click. This will cause Windows to call the procedure Option1.Click which will open COM1 and check for a successful returned status. This automatic selection and manual override has already been used in the IOtest program and will also used in the upcoming program examples. It would be a simple matter to add extra buttons to control the COM3 and COM4 interface.

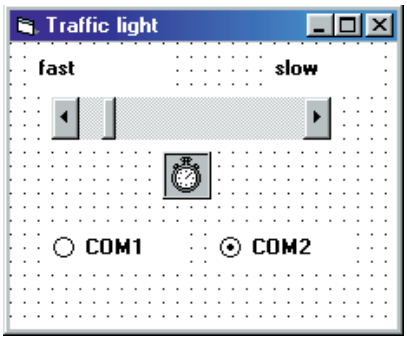


Figure 2. The traffic light controller form.

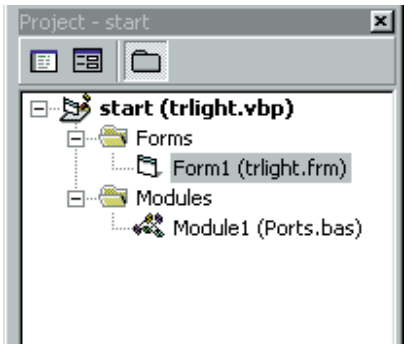


Figure 3. Project overview.

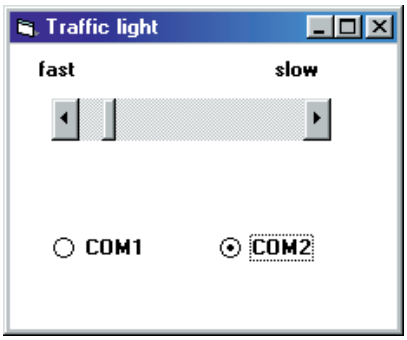


Figure 4. The traffic light program during run time.

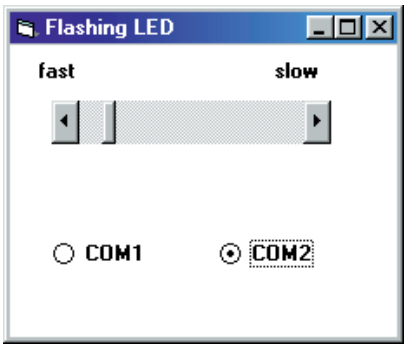


Figure 5. The Blinker Program.

For the traffic light controller we will need some timer function to introduce a delay between changing the lights. In the DLL there is, for example, a procedure called DELAY. However, unlike programming in DOS, individual Windows programs do not have the entire processing time devoted to them. It would be pointless to write a procedure containing a simple timing loop to give us the delays that we need for the traffic light. Instead, the traffic light timer will be event controlled. For this we will use a Windows timer. The timer period is set to 100 ms. The procedure will be called every 100 ms.

In the timer procedure there is a variable called "Time" which is incremented and gives the time in tenths of a second, so the procedure can be called to find out if a specific period has elapsed before, for example, the traffic light LEDs are changed. The speed of switching has been chosen arbitrarily and can be easily altered. The IF statements are used to compare the time values and switch the corresponding LED.

The form (**Figure 4**) also contains a horizontal slider or scroll bar. This slider is used to alter the speed of the traffic light changing. As soon as the slider is moved, the procedure *Hscroll1.Change* is called. This procedure will change the interval in the timer corresponding to the actual position (Value) of the slider control. The slider is calibrated from 50 to 500; the timer can therefore be altered in the range from 50 ms to 500 ms.

Listing 4

Timerprocedure for blink program

```
Private Sub Timer1_Timer()
  Time = Time + 1
  If Time = 1 Then
    RTS 1
    DTR 0
  End If
  If Time = 2 Then
    RTS 0
    DTR 1
  End If
  If Time = 2 Then Time = 0
End Sub
```

Blinker/Clock generator

This next application is a blinker or clock generator with adjustable frequency (**Figure 5**). The circuit is the same as for the traffic light experiment. This time, the TXD output is permanently switched on while DTR and RTS are switched in anti-phase, i.e., when one is on the other is off and vice versa. It is not too difficult to see how this program can be expanded to turn it into a mini running light display.

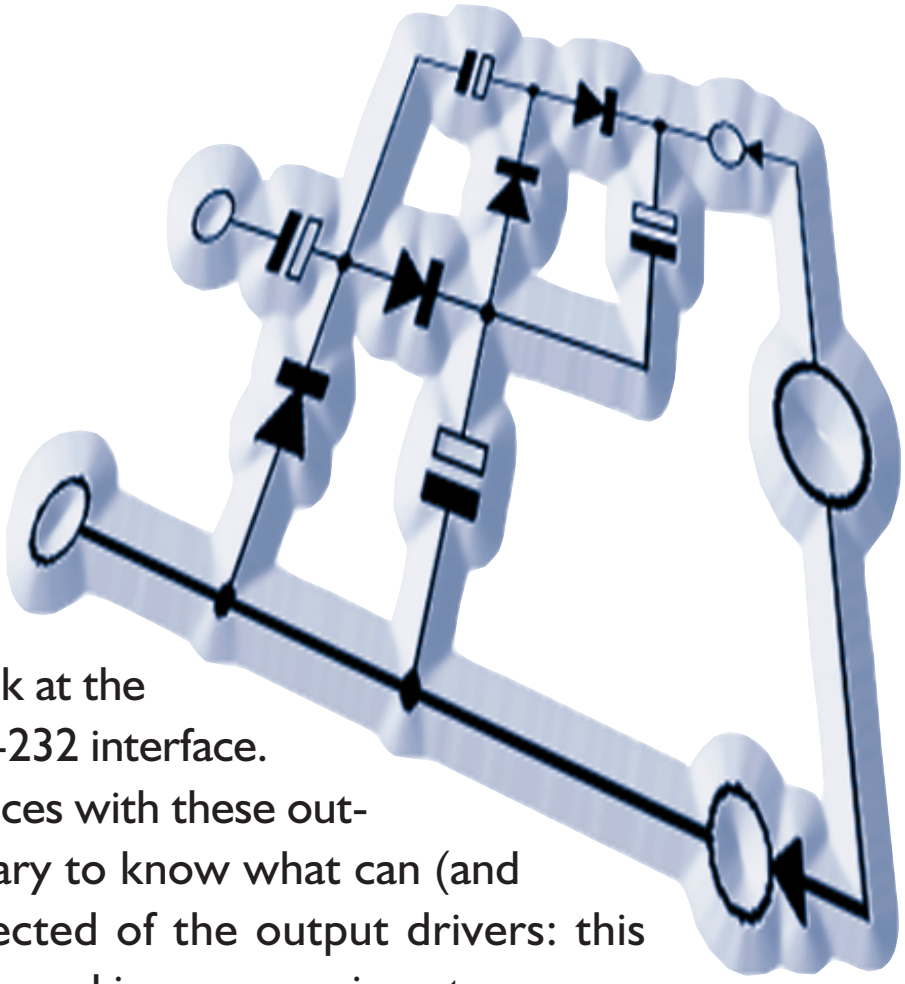
The software from the traffic light program is used again for this exercise. The timer procedure is, however, new (**Listing 4**). One difference here is that no additional procedures are called to control the output lines, they are switched directly from inside 'timer-procedure'.

The LEDs on the DTR and RTS lines should now blink alternately. The speed of the blinking can be increased or decreased. As you increase the speed you may notice that the timing for each blink is not precisely regular. The reason for this is that Windows cannot maintain a time interval of exactly 50 milliseconds because it is a multitasking operating system many other processes will be running at the same time. For this reason Windows is generally considered to be not capable of 'real time' operation. However, there are some tricks and techniques that we can use and we shall be investigating them in the coming articles.

(00074-2e)

PC Serial Peripheral Design (3)

By B. Kainka



In this third part we look at the output signals of the RS-232 interface. In order to control devices with these output signals, it is necessary to know what can (and what cannot!) be expected of the output drivers: this knowledge can then be used in our experiments.

We have already seen in this course that the output signals of the RS-232 interface swing between –10 V and +10 V. However, because the output current is limited, it is safe to connect an LED directly. But what exactly are the characteristics of the outputs? We shall attempt to answer this question now.

In general, manufacturers of PC equipment follow the RS-232 standard, in which the voltage levels used to send data are specified. It was originally specified that the output voltages should be ±15 V, while at the inputs a signal of at least ±3 V is required. Voltages below –3 V are considered to be a logic '1', and voltages above +3 V are considered logic '0'. Since the outputs deliver ±15 V, and the inputs require only ±3 V, reliable data transfer is assured even over long cables. Noise is much less of a problem than it is with for example the TTL logic levels of 0 V and 5 V.

The standard in theory and practice

Modern PCs no longer adhere to the requirement to generate output voltages of ±15 V. The PC's power supply produces +12 V and –12 V, which are used instead. The ultimate voltage is somewhat lower than this again: this is due to the output drivers used. On separate interface cards the 1488 driver and 1489

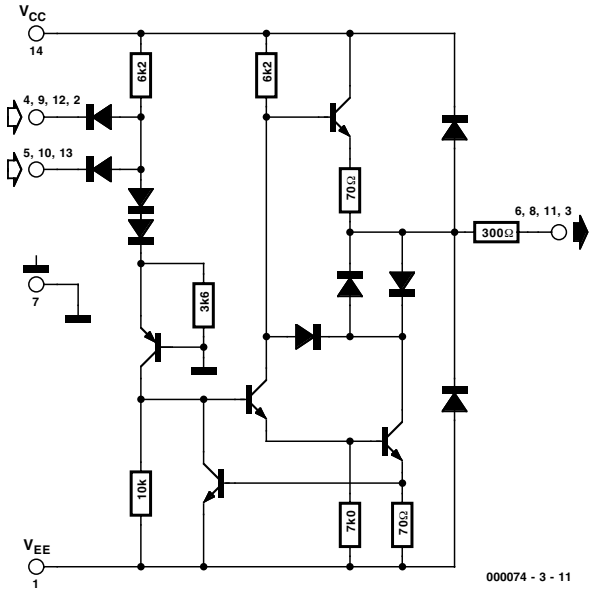


Figure 1. Schematic of the 1488 (source: Motorola)

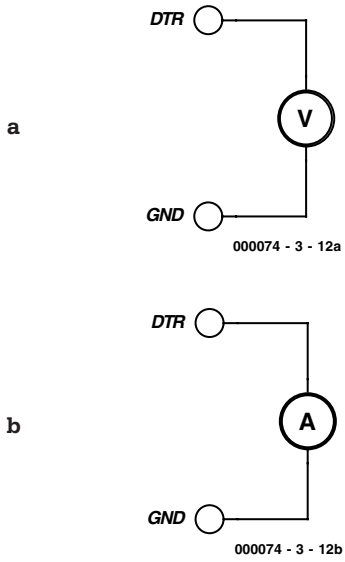


Figure 2. Measuring the open-circuit voltage (2a) and the short-circuit current (2b).

receiver devices are generally used: these have set a *de facto* standard for the behaviour of RS-232 interfaces. **Figure 1** shows the circuit of the driver in the IC. It is clear how the current limiting (set at 10 mA according to the datasheet) is provided. If the output is short-circuited, no more than 10 mA can flow. It can also be seen that 12 V cannot be expected at the output even when a 12 V power supply is used: the voltage drop of the output stage is such that only about 10 V will be produced.

Many PCs are now built with more integrated components, with output drivers already included, which can produce rather different results. It is interesting to determine the exact voltages and currents available from a particular PC. The trend among PC manufacturers is always faster, bigger, better: that means, as far as the RS-232 interface is concerned, higher baud rates over longer cables with greater reliability. To ensure that the higher capacitance of longer cables does not have too great an adverse effect on the pulse shape, the maximum output current must be increased. The outputs of more recent PCs therefore are capable of delivering around 20 mA rather than 10 mA. A total of up to 60 mA can be drawn from the interface. That is handy for our experiments, and is another reason for wanting to make accurate measurements. On some PCs output voltages of ±12 V can indeed be found: these presumably use MOSFETs in their driver output stages.

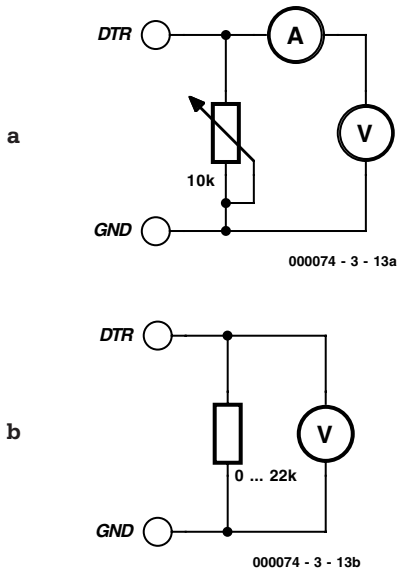


Figure 3. Measurements with variable load.

Table 1

R [kΩ]	I [mA]	U [V]
infinite	0	10.9
22	0.48	10.6
10	1,04	10,4
4.7	2.12	10
2.2	4.18	9.2
1	7.8	7.8
0.47	12.55	5.9
0.33	14.84	4,.9
0.01	25	0.25

Measuring the output characteristics

The output characteristics can be measured very roughly using, for example, a digital multimeter. **Figure 2** shows how to measure the open-circuit voltage and short-circuit current. 'Short-circuit' sounds rather dangerous, but because of its current limiting the RS-232 interface will not be damaged.

The author's PC measures as follows:

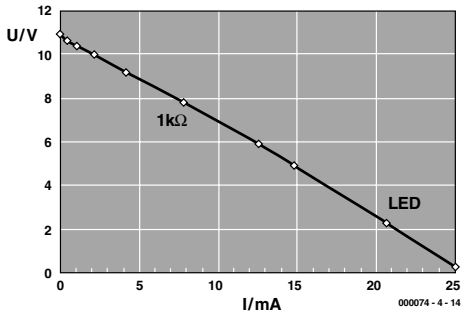


Figure 4. Characteristic curve for one output.

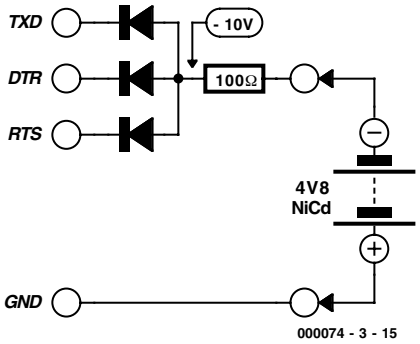


Figure 5. Charging a battery from the RS-232 interface.

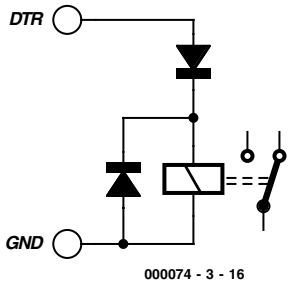


Figure 6. Connecting a relay.

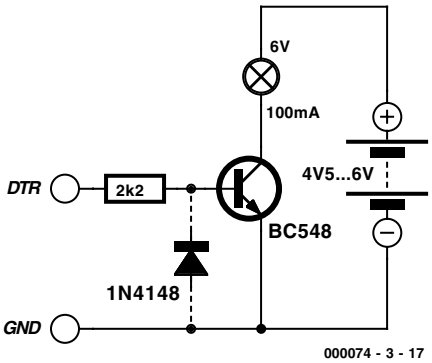


Figure 7. Connecting a transistor switching stage.

Open-circuit voltage: U_0 = –10.9 V (off), U_0 = +10.9 V (on)
Short-circuit current: I_{max} = –22.4 mA (off), I_{max} = 24.4 mA (on)

A number of measurements must be taken in order to get more exact data for a particular PC: the voltage and current must be measured for a range of loads. A potentiometer (say 10 kΩ) or a set of fixed resistors of various values can be used as a load. If two meters are available, the voltage and current can be measured simultaneously (**Figure 3a**); if only one, the voltage can be measured, and then, as long as the resistor value is known, the current can be calculated (**Figure 3b**). For the example here, we used a set of standard resistors. The voltage was measured and the current calculated using the formula $I = U/R$. The results can be seen in **Table 1**.

Figure 4 shows how the output voltage decreases under increasing load. The lowest-valued resistor, at 10 Ω, gives practically a short-circuit, but a voltage can still be measured. The graph shows, to a first approximation, that the voltage falls linearly with increasing current. The internal resistance of the output can be calculated from the gradient of the graph to be about 430 Ω. A modern bright LED has a forward voltage drop of about 2.2 V. From the graph we can read off the current: about 20 mA. This is about the maximum allowable continuous current for a normal LED. So our measurements show that we can connect an LED without a series resistor.

We used Microsoft Excel to analyse the measurements. If you wish, you can carry out the corresponding experiments for your own PC, which makes a good exercise in measurement and analysis; or you can move straight on to the following experiments.

Uses for the serial interface

If the interface can drive LEDs directly, the current must also be enough for other purposes. As we have already stated, up to approximately 60 mA can be drawn from the interface. We look at a few applications here.

The following practical application is a small battery charger operating from the serial interface. In the circuit shown in **Figure 5** the charging current is about 30 mA. That is adequate for small batteries or for trickle charging. We will work with a negative output voltage: the advantage is that the circuit works immediately the PC is switched on, without having to run a program. If it is desired to use the opposite polarity, the diodes can be reversed and the outputs switched to +10 V in software. You could also control charging automatically in software.

Another example of a device that can be driven directly from the serial interface is a small DC motor. A smooth-running motor, such as the ones found in cassette recorders, can be used, and will run satisfactorily on 30 mA. To obtain this current, several outputs are connected together to drive the motor. With a small modification to the 'flasher' program two outputs can be made to switch simultaneously. The motor can then be made to run in either direction.

Small relays can also be connected directly to the interface (**Figure 6**). A diode is normally required to ensure that the relay drops out when the polarity is reversed.

Transistor switching stages

Relays can be used to switch currents greater than that required by LEDs. Cheaper, more elegant and less extravangant is a transistor stage. **Figure 7** shows how a 100 mA bulb can be switched, using an external power supply. The transistor used is a BC548. The principle of the switching stage is very simple: the interface delivers a relatively small base current to the transistor which is amplified, so that the collector current is sufficient to drive the lamp.

How are the components for the switching stage selected? The most important considerations will be discussed briefly here. The operating current of the bulb is 100 mA. This can be provided by a BC548 without difficulty: the maximum collector current allowed is 300 mA. If the lamp current is measured at the moment it is switched on, it will be found that it is considerably greater. When the bulb is cold it has a resistance of about 1/10 of that when it is operating. In theory, then, an instantaneous current of 1 A will flow. In practice, however, the current switched by the transistor is limited.

The bulb filament warms up after only a few milliseconds and so the overload is brief enough to be withstood. If you are not sufficiently reassured by this, you can substitute a BC338 which can withstand a peak current of 800 mA.

The 2.2 kΩ base resistor will have a potential of around 10 V across it. When the output is on, the base current will be about 4.5 mA. The BC548 is available in three gain groups with gains of between 110 and 800. Usually the BC548C is used, with a gain of between 420 and 800. In the pessimistic case a base current of 100 mA/400 = 0.25 mA is required; the actual base current with a 2.2 kΩ base resistor is comfortably more than this. Note, however, that the gain falls at higher collector currents. In any case, the transistor will be driven well into saturation, guaranteeing the smallest possible voltage drop between emitter and collector and keeping power dissipation low both during operation and at the instant the transistor is switched on.

As can be seen, the choice of base resistor is not arrived at by exact calculation, but rather by estimation, since normally the gain of the transistor is not known precisely. It is interesting to try various base resistors to discover over what range reasonable results are obtained. With too high a value, the transistor will not be driven fully into saturation; then the voltage drop becomes greater and the transistor becomes noticeably warm. Also, a resistor which gives satisfactory results in the 'on' state can nevertheless not be able to provide sufficient drive at the instant of turn-on. The lamp appears to turn on slowly, during which time there is considerable power dissipated in the transistor. With too low a base resistor power is wasted in the control circuit, leading also to higher dissipation and possibly to overloading the base-emitter diode by exceeding the maximum allowable base current. The absolute maximum allowable base current is given in the datasheet.

In the circuit diagram a diode is shown connected in reverse between the base and emitter of the transistor. This prevents excessive negative voltages appearing at the base. The maximum reverse voltage of the base-emitter diode generally lies at around –5 V. Breakdown occurs at about –9 V, when a considerable reverse current flows. The transistor behaves like a Zener diode with a voltage of about 9 V. Since in the 'off' state the voltage will be around –10 V, it must be limited. With the diode fitted, the base voltage cannot fall below –0.6 V. It is a worthwhile experiment to investigate the effect of omitting the diode. A reverse current indeed flows, but the transistor nevertheless does not conduct: the lamp remains off. The circuit works even without the diode. It is said that a continuous reverse current over time impairs the noise performance of the transistor; but that will not be a problem here. As so often, it makes a difference whether one is experimenting or designing a circuit in earnest; in the latter case, the diode belongs in the circuit.

AC experiments

The flasher program (Flasher.vbp) produces an alternating voltage on the DTR and RTS outputs, whose frequency can be adjusted up to a limit of about 10 Hz. This lets us try some simple experiments with alternating current. As is well known, a capacitor appears as a resistor to alternating current, conducting the current better at higher frequencies. This can be demonstrated using the circuit of **Figure 8**. Here a capacitor is constructed from two electrolytics connected back-to-back to enable operation with an alternating voltage. This arrangement is sometimes used in driving loudspeakers.

If the flasher program is set to a low frequency, the LEDs alternately flash briefly. At each transition the capacitor is recharged. The current then falls to zero as the voltage across the capacitor approaches the voltage on DTR. If the frequency is increased, the flashes become more frequent and the average brightness of the LEDs also increases. The capacitor presents a lower resistance to the alternating current, and so the (average) current increases.

The alternating voltage produced has a peak value of about 10 V. The peak-to-peak voltage is around 20 V.

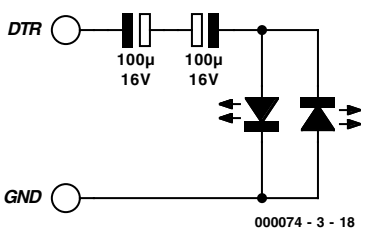


Figure 8. Capacitors as AC resistors.

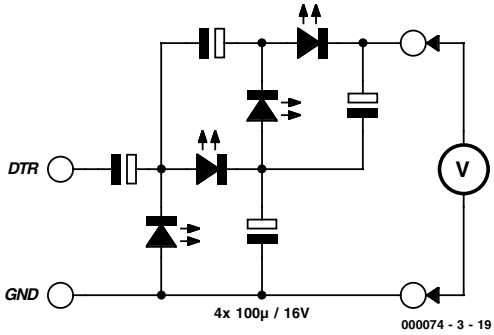


Figure 9. 40 Volts from the serial interface.

Using a special multiplier circuit, a DC voltage of about 20 V can be obtained; using a multiplier cascade a factor of 4 can be achieved, giving about 40 V. The circuit consists of four capacitors and four diodes (**Figure 9**). Here LEDs are used so that the circuit's operation can be seen directly.

When the program is first run, the LEDs flash relatively brightly. The brightness decreases gradually to practically nothing, when the capacitors have charged to their final voltages. A voltage of 40 V can be measured at the output of the circuit. If the capacitors are discharged using a 1 kΩ resistor across the output, the process starts again from the beginning.

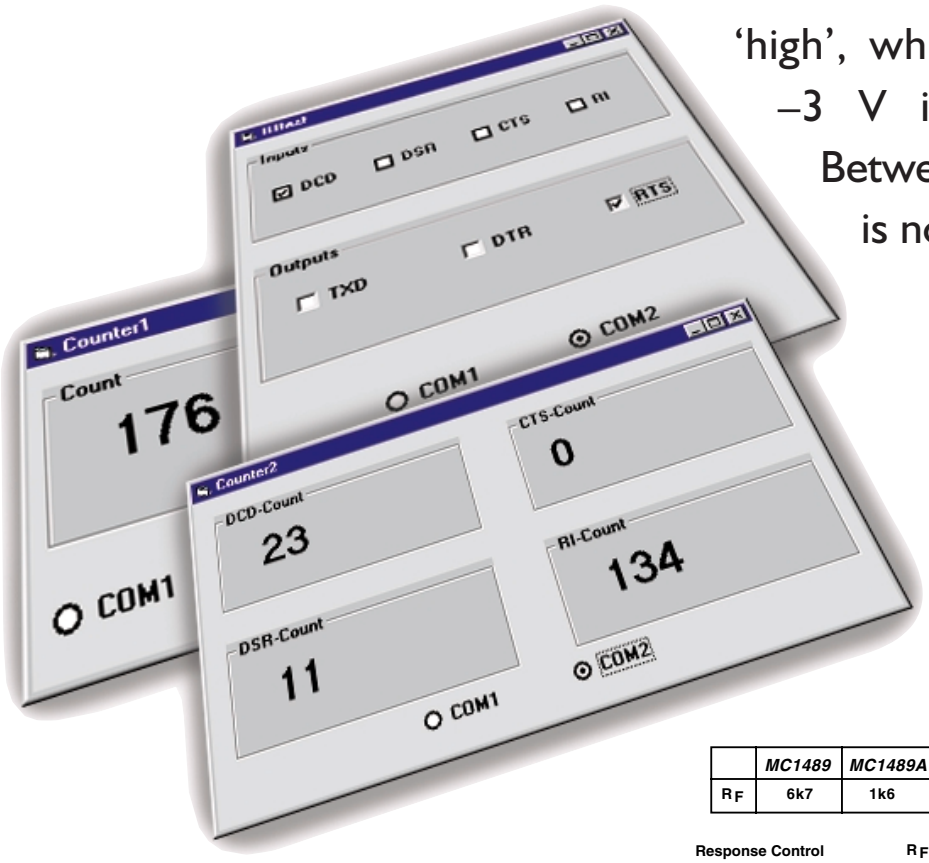
This circuit is an example of how LEDs can be 'stressed' by high voltages. This is fine for our experimental purposes, since experience shows that LEDs can cope with the high voltages without difficulty. One should be aware, however, that such circuits hardly represent exemplary engineering practice; for professional applications, silicon diodes (such as the 1N4148) should of course be used.

(00074-3)

PC Serial Peripheral Design (4)

By B. Kainka

So far we have studied the characteristics of the RS-232 outputs in detail. Now it is the turn of the inputs. The standard demands that an input voltage of more than +3 V is considered as 'high', while a voltage lower than –3 V is considered as 'low'. Between these voltages the level is not defined.



Typical RS-232 interface cards use a type 1489 receiver (**Figure 1**). This IC requires a single 5 V power supply. The schematic shows a simple switching stage consisting of three transistors. As can be seen, the

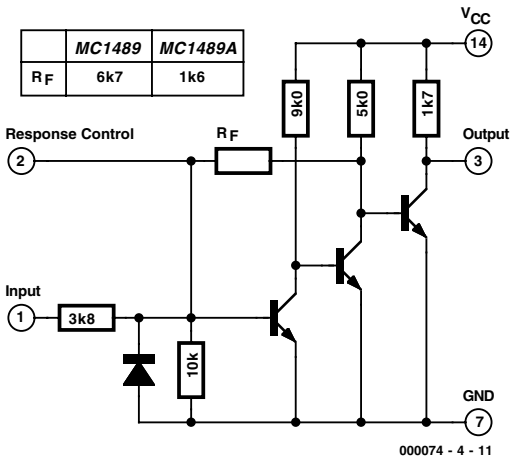


Figure 1. Schematic of 1489 (source: Motorola).

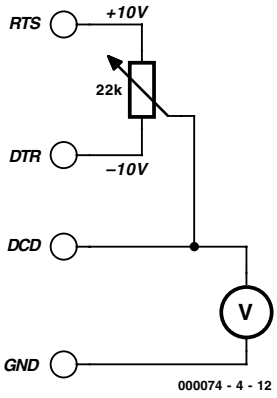


Figure 2. Measuring the input thresholds.

threshold voltage will not be very different from the base-emitter threshold of about 0.6 V. Taking into account the voltage divider at the input (consisting of a 3.8 kΩ and a 10 kΩ resistor) we arrive at a figure of about 0.8 V. A further resistor R_F links the output of the second transistor stage back to the base of the first, providing feedback. The circuit therefore behaves as a Schmitt trigger. There are therefore two switching thresholds, one when the input voltage is rising and one when it is falling. With the input in between these two levels the output will remain in its previous state. According to the datasheet the lower threshold for the 1489 is 1 V, while the higher is 1.25 V, a difference of 0.25 V. The 1489A, in contrast, has a lower resistor R_F, giving thresholds of 1 V and 1.95 V.

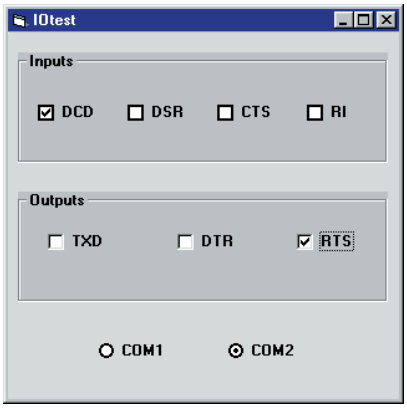


Figure 3. Observing the state of the inputs.

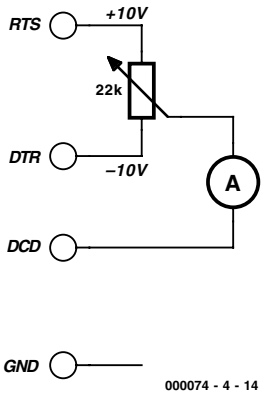


Figure 4. Measuring the input current.

Measurements

The 1489 sets the *de facto* standard for RS-232 inputs. In modern PCs the receiver is normally integrated into a more complex IC, giving the manufacturer the choice of adhering to the quasi-standard, or to interpret the RS-232 standard differently. It is therefore interesting to determine the exact behaviour of the inputs on a particular PC. For this, an adjustable voltage source is required: this does not mean that we will need a piece of laboratory equipment, since the interface itself provides the voltages we shall need. A simple potentiometer suffices to help make our measurements (**Figure 2**).

The IOTEST program from the first instalment of this series is used to make the measurements. The RTS signal is turned on, and the DCD signal is monitored (**Figure 3**). Readings are taken from the voltmeter as the potentiometer is adjusted. The author's PC gave the following results:
Lower threshold: 1.0 V
Upper threshold: 2.0 V

The input circuit in the PC is therefore a 1489A rather than a 1489.

These measurements can be repeated for all four inputs on the serial interface, and similar readings will be obtained. Knowing the upper threshold is useful for some possible applications. For example, it is not possible to detect a 1.5 V battery connected to an input, whereas a direct connection between an RS-232 output and an RS-232 input will always be detected correctly. However, if an LED is connected, the results may be different: it can happen that the LED will glow, but the input will still read as zero.
For some experiments it is important to know the input current rather than the threshold voltage. Here again we can make some simple measurements (**Figure 4**), and, again, we observe a hysteresis. We measured:
Upper threshold: 0.18 mA
Lower threshold: 0.36 mA

From these results we can deduce that an RS-232 output can easily drive, in parallel, more inputs than the four available. From the voltages and currents measured we can calculate an input resistance of 5.6 kΩ. This result seems plausible, looking at the schematic of the 1489.

An important result from these measurements is that a negative voltage is not required for an input to read as zero, even though the RS-232 standard prescribes a voltage below –3 V. Many experiments can therefore be carried out with only a single power supply. There may, of course, be the odd PC which behaves differently from our example. In particular, some laptops require negative input voltages. This must be taken into account in some of our experiments.

Listing I

The Counter1 program

```
Dim DSRold, Counter1

Private Sub Form_Load()
 i = OPENCOM("COM2,1200,N,8,1")
 If i = 0 Then
   i = OPENCOM("COM1,1200,N,8,1")
   Option1.Value = True
 End If
 If i = 0 Then MsgBox ("COM
Interface Error")
 TXD 1
 RTS 1
 DTR 1
 Counter1 = 0
 DSRold = DSR()
 Timer1.Interval = 20
End Sub

Private Sub Timer1_Timer()
DSRNew = DSR()
If DSRNew > DSRold Then
   Counter1 = Counter1 + 1
   Label1.Caption =
Str$(Counter1)
End If
DSRold = DSRNew
End Sub
```

Listing 2

Pulse counter with four inputs

```
Private Sub Timer1_Timer()
 DCDnew = DCD()
 DSRnew = DSR()
 CTSnew = CTS()
 RInew = RI()
 If DCDnew > DCDold Then
   Counter1 = Counter1 + 1
   Label1.Caption =
Str$(Counter1)
 End If
 If DSRnew > DSRold Then
   Counter2 = Counter2 + 1
   Label2.Caption =
Str$(Counter2)
 End If
 If CTSnew > CTSold Then
   Counter3 = Counter3 + 1
   Label3.Caption =
Str$(Counter3)
 End If
 If RInew > RIRold Then
   Counter4 = Counter4 + 1
   Label4.Caption =
Str$(Counter4)
 End If
 DCDold = DCDnew
 DSRold = DSRnew
 CTSold = CTSnew
 RIold = RInew
End Sub
```

Reading the state of a switch

It is easy to use the serial interface to read the state of up to four switches (**Figure 5**). One output, for example DTR, is required: this is set high, in order to generate the required voltage. The switches can be connected via a cable of practically any length. Up to three typical 'reset-switches' can be fitted to the circuit board (**Figure 6**), in which case three input signals are used.

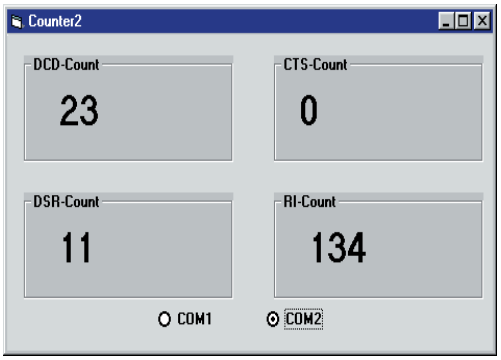


Figure 8. The four-way counter Counter2.

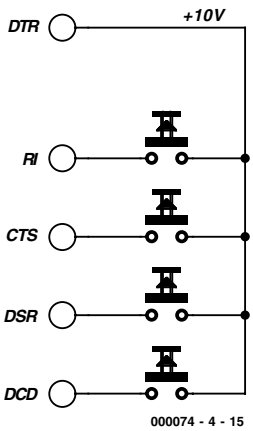


Figure 5. Up to four switches can be connected.

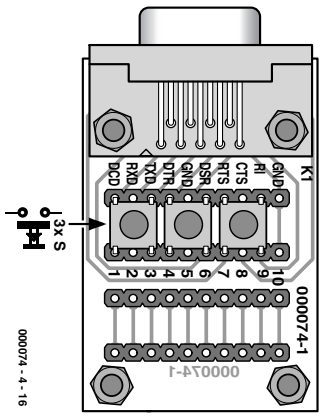


Figure 6. Three buttons can be fitted directly to the circuit board.

Pulse counter

Building a counter in digital hardware is a relatively complicated job. When a PC is available, however, it is easy. Here a pulse counter is constructed in Visual Basic. The DSR signal serves as an input: a simple push-button can be used to provide pulses. Any other sensor could be used, as long as it can provide the correct voltage.

Listing 1 shows the program Counter1; the results on the screen are shown in **Figure 7**. Two global variables are used. DSRold stores the previous state of the DSR signal and Counter1 stores the count. The two variables are initialised in the first routine. The counter is set to zero, and DSRold is loaded with the state of the DSR signal. The outputs are also all turned on. This allows any of the outputs to be connected to the DSR input to provide a count signal.

The actual counting is done in the timer routine. Windows calls this routine roughly once every 20 ms. Each time the state of the DSR input is compared with the state on the previous call. If the new state is greater than the old state, a change from 0 to 1 must have happened: in other words, a rising edge. These edges are counted by incrementing the variable Counter1 by one each time. The value on the screen is only refreshed when the variable changes.

The pulses are counted reliably whether the button is pressed rapidly or slowly, up to about 5 pulses per second. When the pulse rate is faster than this, however, it will be found that some pulses may be missed. The exact limit depends on the PC. With a timer interval of 20 ms a signal with low and high periods of 20 ms should theoretically be read correctly. A total period of 40 ms corresponds to a frequency of 25 Hz (25 pulses per second). However, problems are observed at lower frequencies than this. From this we deduce that Windows cannot reliably maintain a timer interval of 20 ms. A similar observation was made when we discussed the LED flasher program. Even with a timer interval of 50 ms definite irregularity can be seen.

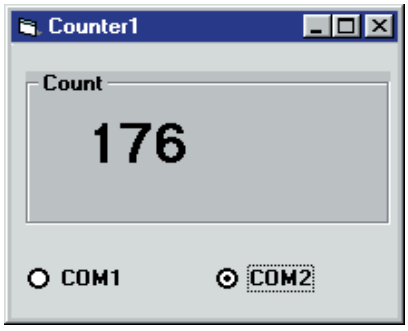


Figure 7. The Counter1 program.

This counter program is certainly not the fastest possible. Furthermore, we can easily construct a four-way counter, as shown in **Figure 8**: we simply need to write out the code four times. Each time a different input is read and processed. **Listing 2** shows the modified timer routine.

(000074-4)

PC Serial Peripheral Design (5)

Analogue measurements

By B. Kainka

So far in this series we have used the PC only with digital signals: switching, monitoring and counting. Now we turn to the analogue domain: our programs will understand not just 'yes' and 'no', but 'larger' and 'smaller'.

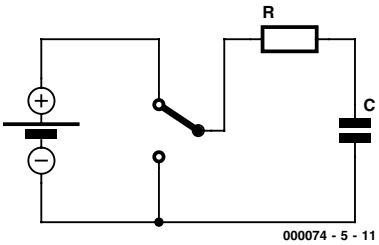


Figure 1. Charging and discharging a capacitor

If a voltage is applied to a serial port input, it will be read as either a logic Low (0) or a logic High (1). The PC cannot determine the actual voltage present. Likewise, if a resistor is connected between an output that has been set high and an input, there are only two possible results: either the resistor is sufficiently small that a clear logic one is read at the input, or it is not. If more precision is required, a new approach is needed.

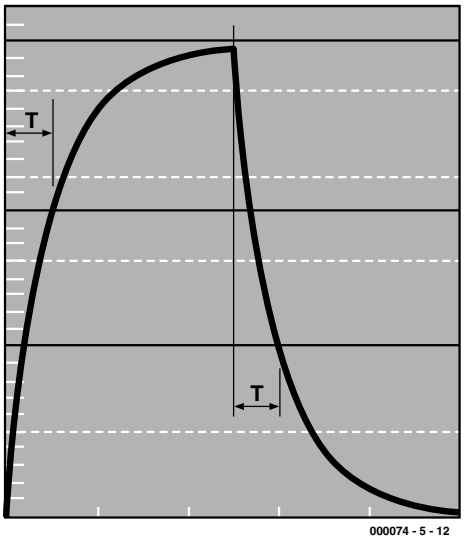


Figure 2. Charge/discharge curves and definition of the time constant T

Charging and discharging

If computers are good at one thing, it is counting. We can use this to measure time: a program simply counts the seconds (or milliseconds) that pass until a certain event happens, for example when an input changes state. If we can convert an analogue quantity such as a resistance into a time period, then it will be easy to measure it with a computer. In this example, we can use an RC network. **Figure 1** shows a capacitor C charged and discharged through a resistor R. A time constant T is associated with an RC network:

$T = RC$.

Here T is the time taken for the voltage across the capacitor to reach 63.2 % (=1-1/e) of its final value. This can be derived from the exponential charging characteristic, shown in **Figure 2**. We shall not go into the details of the physics here: it is enough to know that the time taken to charge to a given voltage is directly proportional to the capacitance and to the value of the resistor.

For a 100 μF capacitor and a 1 kΩ resistor we have:

$T = 1000\ \Omega \times 0.0001\ \text{F} = 0.1\ \text{s} = 100\ \text{ms}$

A doubling of the resistance leads to a doubling of the charging time. The same goes for a doubling of the capacitance. So, we can measure the charging time and deduce the value of either one of the resistor or the capacitor, if the value of the other is known. All that is needed is a program that replaces the switch in Figure 1. **Figure 3** shows a simple circuit where the capacitor is connected not to ground but to the TXD output. There is a good reason for this: if an electrolytic capacitor is used, it must never be charged with the wrong polarity, and this is guaranteed as long as TXD remains at –10 V.

The circuit charges and discharges the capacitor via DTR and uses the DSR signal as an input to determine when the set voltage is reached. The threshold voltage will be around 1.5 V. Comparing this with the overall voltage range of –10 V to +10 V, we see that the threshold is about 11.5 V/20 V = 0.575 = 57.5 % of the final voltage. This is not too far from our value of 63.2 % given above for the time constant. In any case, the error factor introduced is constant and can be compensated for later. There are other sources of error, which we discuss below.

Listing 1. Measuring the time constant in milliseconds

```
Private Sub Form_Load()
 i = OPENCOM("COM2,1200,N,8,1")
 If i = 0 Then
    i = OPENCOM("COM1,1200,N,8,1")
    Option1.Value = True
 End If
 If i = 0 Then MsgBox ("COM Interface Error")
 TXD 0
 RTS 0
 DTR 0
 Counter1 = 0
 Timer1.Interval = 2000
End Sub

Private Sub Form_Unload(Cancel As Integer)
  CLOSECOM
End Sub

Private Sub Timer1_Timer()
  DTR 1
  TIMEINIT
  While (DSR() = 0) And (TIMEREAD() < 1501)
    DoEvents
  Wend
    Label1.Caption = Str$(TIMEREAD()) + " ms"
  DTR 0
End Sub
```

Counting time

The time measurement takes place in a 'while' loop as shown in **Listing 1**. The loop runs until either DSR becomes set or the count reaches 1.5 seconds (timeout). The measurement loop includes the command DoEvents: this allows Windows to process other events that may be occurring in the system. During measurements the user can still use the mouse and other applications, and indeed stop the program itself. This is reassuring for the user, especially when bugs in the program cause it to function incorrectly. In general it is always necessary, when programming loops, to consider how the loop can be forced to terminate. Otherwise if the program gets into an infinite loop the PC will need to be reset, either by switching it off and on again, or by using the familiar Ctrl-Alt-Del chord. Adding a DoEvents call makes the loop safe; but this has a cost in timing accuracy, adding an uncertainty of around 1 to 3 milliseconds to the measured time.

µF not ms

If the units 'ms' in the window in **Figure 4** are replaced by 'µF', the value shown is not too far off the true one. As we said above, a 100 µF capacitor and a 1 kΩ resistor give a time constant of 100 ms. Similarly, with a 10 µF capacitor we have a time constant of 10 ms. This can be tested by trying various capacitors from the junk box. Often there will be quite a large discrepancy between the value measured and the value printed on the capacitor: this is down to the large tolerance (as much as 50 %) quoted for electrolytics. The capacitance of an electrolytic often changes if it is stored for a long time.

The measurements are less accurate for very large electrolytic capacitors, with values of say around 1000 µF. The indicated value will be too small. The reason for this lies in the program: the charged capacitor must be discharged, which also requires time. Our program uses a timer with a period of two seconds: one second is used for charging the capacitor up to the input threshold voltage; the remaining second, however, is not enough to discharge the capacitor completely. The next charge therefore takes less time. There is a simple solution: a diode can be used to accelerate the discharge cycle. **Figure 5** shows the circuit diagram, and **Figure 6** shows how it can be constructed. With this modification to the circuit we can

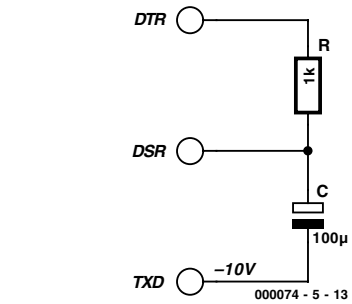


Figure 3. Automatic charging from the PC

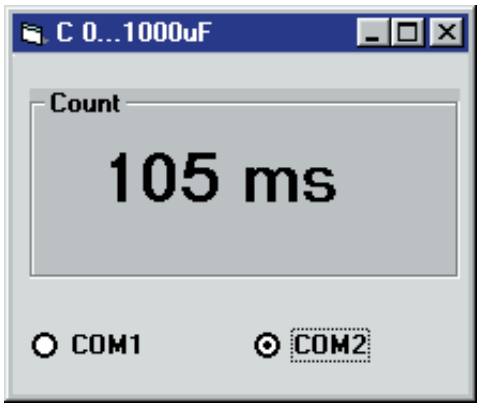


Figure 4. Measurement with 100 µF and 1 kΩ

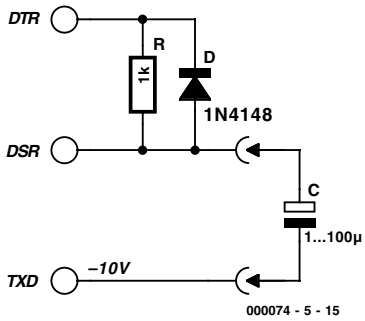


Figure 5. Improved capacitance measurement circuit

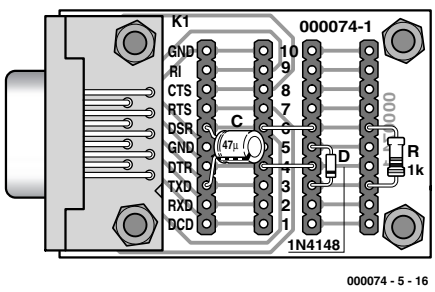


Figure 6. Construction details.

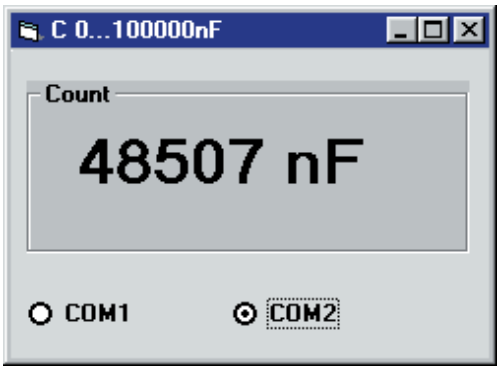


Figure 7. Capacitance display in nF

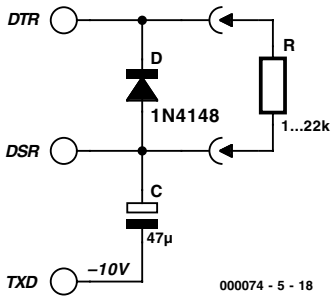


Figure 8. Circuit 1 with 10 kΩ potentiometer.

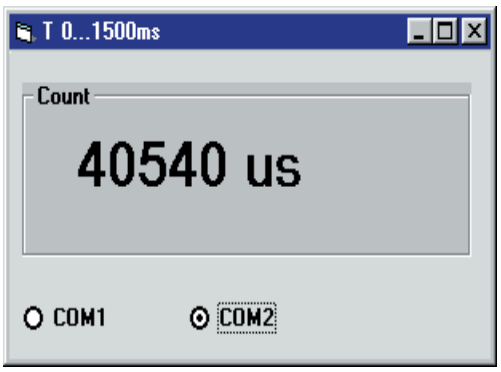


Figure 9. Measuring the charging time to microsecond resolution

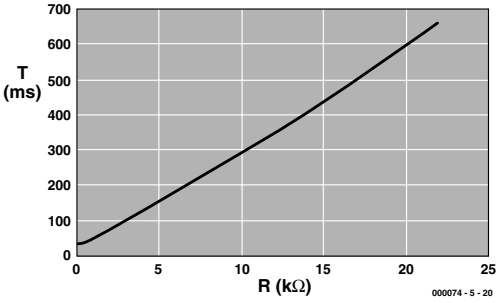


Figure 10. Charging time plotted against charging resistance

Listing 2. Modifications for program Cmeas2.frm

```
Private Sub Timer1_Timer()
  F = 1.19
  DTR 1
  REALTIME (True)
  TIMEINITUS
  While (DSR() = 0) And (TIMEREADUS() < 1500000)
  Wend
  T = TIMEREADUS() * F
  REALTIME (False)
  T = Int(T)
  Label1.Caption = Str$(T) + " nF"
  DTR 0
End Sub
```

Listing 3. Measuring the time constant in microseconds

```
Private Sub Timer1_Timer()
DTR 1
  REALTIME (True)
  TIMEINITUS
  While (DSR() = 0) And (TIMEREADUS() < 1500000)
  Wend
  t = TIMEREADUS()
  REALTIME (False)
  Label1.Caption = Str$(t) + " us"
  DTR 0
End Sub
```

Listing 4. Determining the charging resistance

```
Private Sub Timer1_Timer()
  DTR 1
  REALTIME (True)
  TIMEINITUS
  While (DSR() = 0) And (TIMEREADUS() < 1500000)
  Wend
  T = TIMEREADUS()
  T = T * 1.0000000001
  R = 2200 + 7800 * (T - 76300) / (294600 - 76300)
  REALTIME (False)
  R = Int(R)
  Label1.Caption = Str$(R) + " Ohm"
  DTR 0
End Sub
```

reliably measure capacitors up to about 1500 µF. Larger values are possible if more time is allowed by making the appropriate changes to the program. The timeout value in the measurement routine and the overall timer period must both be increased, so that the program waits long enough for the measurement to be completed.

What about capacitors smaller than 1 µF? In principle the charging resistor could be increased. However, a problem then arises: the impedance of the DSP input (around 3 kΩ) introduces measurement errors that get more and more severe as the value of the charging resistor is increased. This problem could for example be overcome using an operational amplifier with a high input impedance, but this is outside the scope of this series.

Software enhancements

It is much more interesting to try to get around these limitations in software. In particular it is possible to use a timer with a resolution of one microsecond. This increases the resolution of the capacitance measurement one thousandfold, allowing measurements in nanofarads (**Figure 7**). Library PORT.DLL provides functions TIMEINITUS and

TIMEREADUS for this purpose, where 'US' stands for microseconds (µs). These functions are used in **Listing 2**.

When we consider making measurements in microseconds, we must look at the effect of Windows on the timing accuracy. In principle other process running in parallel can interrupt the measurement program and cause large inaccuracies. This can be prevented by raising the priority of the measurement task, for which a special function is provided in PORT.DLL. Using REALTIME (True) we can obtain greater reliability. It is essential to set REALTIME (False) after the measurement has been taken. The exact accuracy achieved depends on the PC: with a 200 MHz Pentium MMX we measured variations of about 50 µs in the measured value; with a faster PC this may be reduced. If the same experiment is carried out in Delphi, the timing accuracy is about 20 times better. The method is described in the *Elektor Electronics* book 'PC Interfaces under Windows', to be published soon). It is nonetheless impressive that an interpreted language such as Visual Basic can achieve such timing accuracy.

Alongside these changes to the program, we can also improve the basic accuracy of the measurements. We have already seen two sources of error in the simple equation t/ms = C/µF, namely the threshold voltage being slightly too low and the input impedance of the DSR signal. There is a third source of error: the DTR output does not switch exactly between -10 V and +10 V, but has an internal resistance of roughly 430 Ω. The charging resistance is therefore in effect about 1430 Ω. Further, this internal resistance is non-linearly dependent on the current. The TXD output also has an internal resistance, making the voltage on the capacitor slightly higher than expected. The effects are too complicated to be analysed mathematically, and so we take the course preferred by experienced engineers in the face of a complicated calculation: test; measure; calibrate. All the errors can be condensed into a single correction factor F which can be determined with a calibration measurement. For this we require a capacitor whose value is accurately known (or which can be accurately measured). Then the correction factor can be adjusted until the reading is correct. On the author's PC the value of F was found to be 1.19; this value can of course be used on any PC, but there will be individual variations from machine to machine which can only be compensated for by determining the correct value of F in each case.

Resistance measurement

We can measure resistance using the same method; this is similar to the way that potentiometers are read by PC games cards. The circuit for measuring resistance is shown in **Figure 8** and is essentially the same as the capacitance-measuring circuit. Here, however, we use a fixed capacitor and work with various resistor values.

To test this circuit we use the program (**Listing 3**) which measures the charging time to the highest accuracy. We use again the technique described above for measuring small capacitors: REALTIME (True) gives us good timing accuracy. **Figure 9** shows how the results appear on the screen.

The circuit can be tested using metal film resistors with a tolerance of 1 %. Measurements with an accurate ohmmeter indicate that in general the tolerance of such resistors is rather better. A set of tests using C=47 µF gave the following results:

R/kΩ	T/ms
0	33.7
0.1	34
0.22	34.5
0.56	37.9
1	45.5
2.2	76.3
4.7	147.5
6.8	204.9
8.2	245.9
10	294.6
15	433.7
22	661.2

From the numbers alone a strong linear dependency can be seen. An exact determination of the linearity is possible using a graph: an analysis using Excel produced the curve shown in **Figure 10**. It can be seen that linearity is good between around 2.2 kΩ and 10 kΩ.

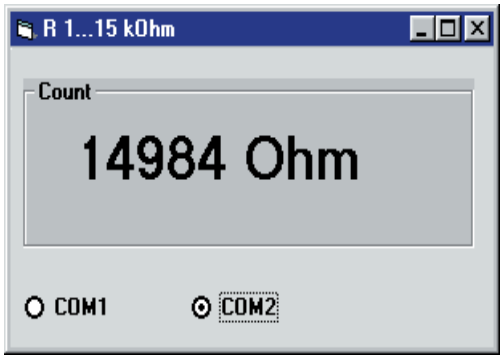


Figure 11. Measurement using a 15 kΩ resistor.

The reasons for deviation from the ideal linear curve are the same as were found in measuring capacitance. With very small charging resistances we get an error due to the non-linear internal resistance of the DTR output driver; with large resistances (say around 22 kΩ) the low impedance of the DSR input causes error.

The data obtained can be converted into an explicit equation which will let us calculate a resistance with high accuracy. **Listing 4** shows the measurement routine for resistance. The calculation in question reads:

```
R = 2200 + 7800 * (T - 76300) / (294600 - 76300)
```

Using this formula we can obtain a measurement accuracy of around 1 % in the range 1.5 kΩ to 15 kΩ. It should be borne in mind, however, that this function, specific to a particular PC, will not give the same accuracy on a different machine. The various measurements and calculations must be carried out afresh for each PC. It is sufficient to measure the charging times for two resistors, say 2.2 kΩ and 10 kΩ and substitute these in the formula. Failing this, it is possible simply to consider where the greatest sources of error lie: in this case our attention turns immediately to the electrolytic capacitor. Electrolytics often have a capacitance very different from the value printed on them, which has a significant effect on the charging time for a given resistance. If a suitable correction factor is inserted in the line

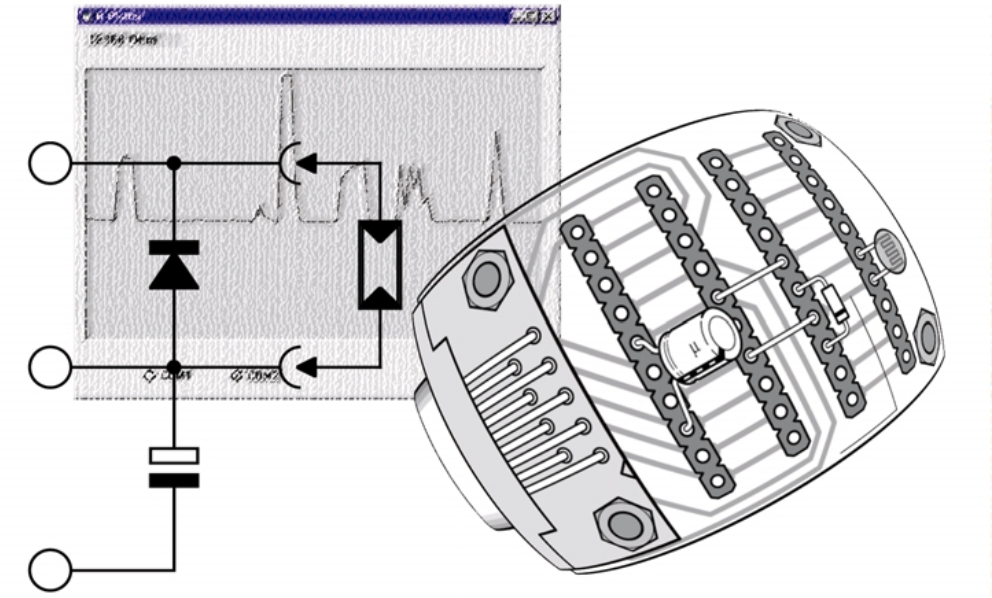
```
T = T * 1.0000000001
```

then this error will be compensated for, and reasonably usable results (see **Figure 11**) can be obtained. A correction factor of greater than or less than one will be required according to whether the capacitor has a value lower or higher than nominal.

(00074-5)

PC Serial Peripheral Design (6)

Measurements with sensors



By B. Kainka

In the previous instalment we measured capacitances and resistances. You may have wondered if this was all worth the effort, when accurate and inexpensive digital multimeters are readily available, but we shall see that we are able to take readings from a wide variety of sensors by measuring their resistance.

The best demonstration involves a light-dependent resistor (LDR). **Figure 1** shows a simple light-measuring circuit; **Figure 2** shows how it might be constructed. In this case we are frequently interested in how the reading changes over time, and this is best shown on a graph.

Light measurement

In order to produce a graph on the screen using Visual Basic, we need to use a so-called PictureBox, which can be dragged from the form's toolbar. It is very important to set the correct size. Visual Basic can use a variety of units and here we want to measure the size in pixels. The ScaleMode property should be set to '3 - pixel', and the box can be

Listing I. Resistance Plotter

```
Dim y1, y2, x1, x2, n
Private Sub Form_Load()
 i = OPENCOM("COM2,1200,N,8,1")
 If i = 0 Then
    i = OPENCOM("COM1,1200,N,8,1")
    Option1.Value = True
 End If
 If i = 0 Then MsgBox ("COM Interface Error")
 TXD 0
 RTS 0
 DTR 0
 Counter1 = 0
 Timer1.Interval = 1000
 n = 0
End Sub

Private Sub Form_Unload(Cancel As Integer)
  CLOSECOM
End Sub

Private Sub Option1_Click()
 i = OPENCOM("COM1,1200,N,8,1")
 If i = 0 Then MsgBox ("COM1 not available")
End Sub

Private Sub Option2_Click()
 i = OPENCOM("COM2,1200,N,8,1")
 If i = 0 Then MsgBox ("COM2 not available")
End Sub

Private Sub Timer1_Timer()
  DTR 1
  REALTIME (True)
  TIMEINITUS
  While (DSR() = 0) And (TIMEREADUS() < 900000)
  Wend
  T = TIMEREADUS()
  DTR 0
  T = T * 0.932
  R = 2200 + 7800 * (T - 76300) / (294600 - 76300)
  REALTIME (False)
  R = Int(R)
  Label1.Caption = Str$(R) + " Ohm"
  y2 = 300 - R / 100
  If n = 0 Then y1 = y2
  x1 = n
  n = n + 5
  x2 = n
  Picture1.Line (x1, y1)-(x2, y2)
  y1 = y2
End Sub
```

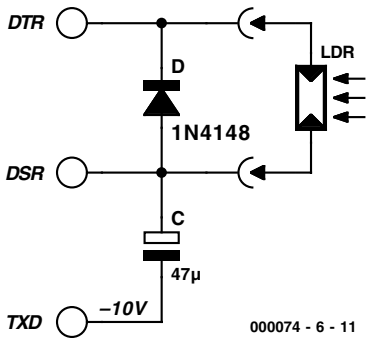


Figure 1. Measurement using a light-dependent resistor

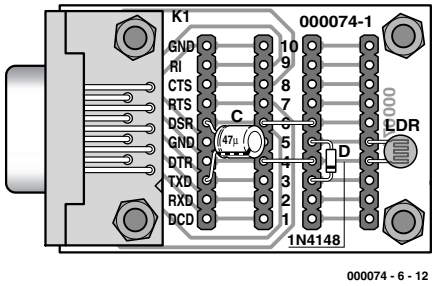


Figure 2. Construction of the LDR circuit

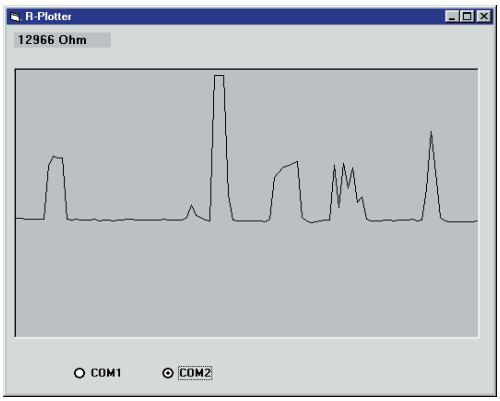


Figure 3. Results from the LDR measurement

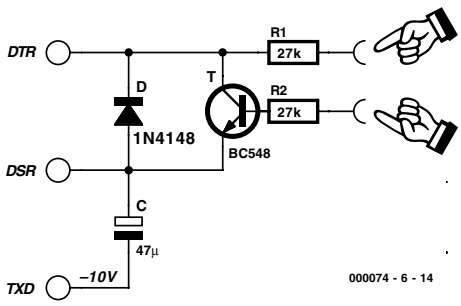


Figure 4. Amplifier for measuring skin resistance

dragged out to the correct size using the mouse. For the present example (plotter1.frm) a PictureBox 300 by 500 points is required.

We can draw in the PictureBox using the Line command: see **Listing 1** for an example. The plot is drawn over a period of 100 seconds, the x-coordinate advancing by 5 pixels as each reading is taken. A global variable n must be specially declared to store the current position across calls to the timer procedure. On the y-axis resistance (R/100) is plotted, with a maximum value of 300 (30000 Ω=30 kΩ). Every second a new value is plotted in the graph in **Figure 3** and a new line segment is drawn from the last point to the new one. The old coordinate values must be stored, and global variables are used for this purpose.

Skin resistance measurements

Another interesting 'sensor' is the human skin: a pair of bare wires can be wound around two fingers. The resistance of the skin is not constant,

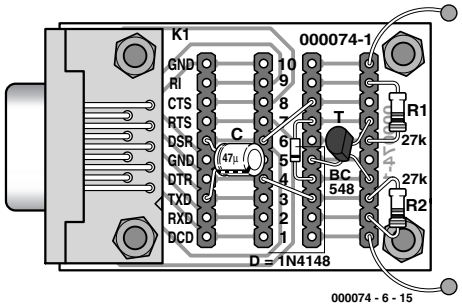


Figure 5. Measuring skin resistance

but varies according to its moisture levels. When people lie, they sweat: so we can build a lie detector by determining when the skin resistance falls. The problem is that skin resistance is typically rather higher than the maximum value we can measure with our circuit: we need to use a transistor for amplification (**Figure 4, Figure 5**).

An NPN transistor amplifies the small current flowing through the skin, allowing our simple A/D converter to measure relatively high resistances. The input to the circuit is protected by two additional resistors, so that the measured resistance will not be too small even if the wires are shorted. The reading obtained depends on the contact area of the wires and the moisture level of the skin. The results can be viewed using the resistance plotter program.

Listing 2. Extensions for temperature measurement

```
Private Sub Timer1_Timer()
  DTR 1
  REALTIME (True)
  TIMEINITUS
  While (DSR() = 0) And (TIMEREADUS() < 1500000)
  Wend
  T = TIMEREADUS()
  T = T * 1.0000000001
  R = 2200 + 7800 * (T - 76300) / (294600 - 76300)
  REALTIME (False)
  R = Int(R)
  Temp = 1 / (Log(R / 10000) / 4300 + 1 / 298) - 273
  Temp = Int(Temp * 10) / 10
  Label1.Caption = Str$(Temp) + "°C"
  DTR 0
End Sub
```

Listing 3. Conversion to the Fahrenheit scale

Although the German physicist Gabriel Daniel Fahrenheit died in 1736, some people are still more comfortable working with temperatures measured in Fahrenheit. They need only change a couple of lines of the program, as follows:

```
Temp = 32 + (Temp / 100 * 180)
Temp = Int(Temp * 10) / 10
Label1.Caption = Str$(Temp) + " F"
```

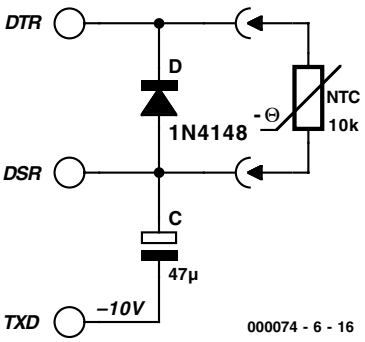


Figure 6. Connecting an NTC thermistor

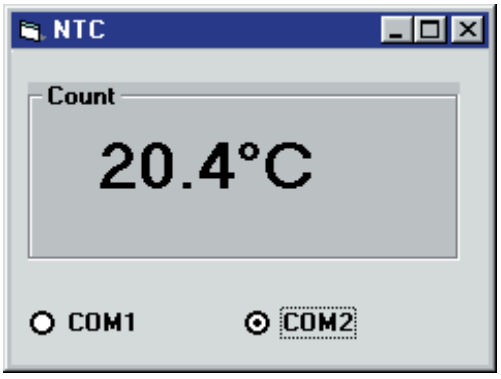


Figure 7. Temperature display

Temperature measurement

Temperature can also be measured using our resistance measuring circuit. The circuit of **Figure 6** shows how easy it is to connect a 10 kΩ NTC thermistor to build a very simple yet perfectly usable thermometer.

The resistance characteristic of an NTC thermistor can be approximated by an exponential curve. In the following formula, T is the absolute temperature in Kelvin (T/°C + 273), while B is a value, typically between 2000 and 500 Kelvin, provided by the thermistor manufacturer. In fact the value of B is not perfectly constant but rises gradually with temperature: for this reason the value $B_{25/85}$ is often quoted, where calibration has been performed at 25°C and 85°C.

$$R(T) = R_{25} \cdot e^{B \cdot \left(\frac{1}{T} - \frac{1}{298K}\right)}$$

An NTC thermistor is therefore specified to a first approximation by two values: B and the thermistor's nominal resistance R_{25}. For a typical 10 kΩ NTC thermistor the value of B might be 4300 K. Rearranging the expression above and substituting R_{25} = 10 kΩ we can derive the following line of Visual Basic to calculate the value of T in Celsius:

Temp = 1 / (Log(R / 10000) / 4300 + 1 / 298) – 273

The VB 'Log' function calculates the natural logarithm, often abbreviated to 'ln' in mathematics textbooks.

In the program shown in **Listing 2** we first carry out a resistance measurement and then convert the measured value into temperature. The results appear to one decimal place as shown in **Figure 7**. Finally, **Listing 3** shows how to convert the reading from the Celsius scale into Fahrenheit.

(000074-6)

PC Serial Peripheral Design (7)

voltage measurement

By B. Kainka

In previous articles in this series we have described an A/D converter based on a counter. A 'real' A/D converter, however, converts a voltage into a measured value. Usually, this is thought to require an IC; however, as we shall see, a much simpler alternative is available.

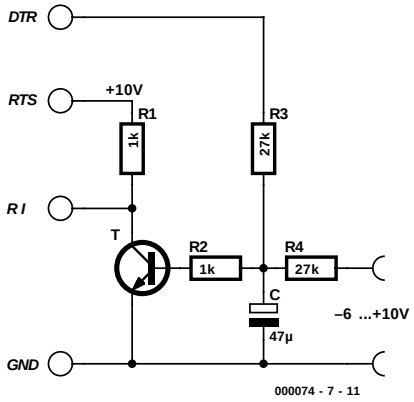


Figure 1. The simple A/D converter

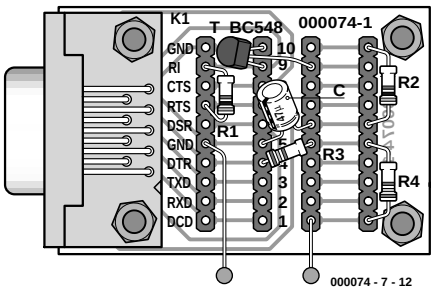


Figure 2. Voltage measurement with the simple A/D converter.

A real A/D converter with a voltage input can be constructed using a single transistor. The transistor in **Figure 1** operates as a comparator, comparing the voltage on the capacitor with a reference value, here 0.7 V. The capacitor is alternately charged and discharged by the DTR output via a resistor, in such a way that the voltage across it is always near to the comparator reference. Depending on the input voltage the output will need to be turned on more or less frequently in order to establish the desired voltage. The count of these events leads to the converted value. **Figure 2** shows how the circuit can be constructed.

A/D converter program

The program in **Listing 1** contains a loop which attempts to keep the voltage across the capacitor as close as possible to the comparator reference voltage. Information on the voltage across the capacitor is available on the RI input: if the voltage lies above the switching threshold of the transistor (about 0.7 V), a collector current flows and the voltage on RI falls. Conversely a lower capacitor voltage turns off the transistor and RI goes high. The measurement loop must drive the voltage on DTR in the opposite direction: when the voltage is too low, DTR must be turned on; when it is too low, DTR must be turned off, setting the output voltage to –10 V. It is important that the switching happens at precisely regular intervals, for example exactly one per millisecond. The number of millisecond periods for which DTR is turned on must be counted. If no input voltage is present, we might expect that the positive and negative states of DTR would be equally common, making the averaged voltage across the capacitor zero. If we look a little more closely, however, we see that we need to consider the threshold voltage of the transistor (about 0.7 V), and so the positive state will be slightly more frequent. With an applied voltage the ratios change. The charging current from the DTR signal must compensate for the charging current from the input.

A negative input voltage leads to more positive DTR states, and con-

Listing 1. Conversion of an 8 bit value

```
Private Sub Form_Load()
 i = OPENCOM("COM2,1200,N,8,1")
 If i = 0 Then
    i = OPENCOM("COM1,1200,N,8,1")
    Option1.Value = True
 End If
 If i = 0 Then MsgBox ("COM Interface Error")
 RTS 1
 DTR 1
 Counter1 = 0
 Timer1.Interval = 500
End Sub

Private Sub Form_Unload(Cancel As Integer)
 CLOSECOM
End Sub

Private Sub Option1_Click()
 i = OPENCOM("COM1,1200,N,8,1")
 If i = 0 Then MsgBox ("COM1 not available")
 RTS 1
 DTR 1
End Sub

Private Sub Option2_Click()
 i = OPENCOM("COM2,1200,N,8,1")
 If i = 0 Then MsgBox ("COM2 not available")
 RTS 1
 DTR 1
End Sub

Private Sub Timer1_Timer()
  RTS 1
  DTR 0
  U = 0
  TIMEINIT
  While (RI() = 0) And (TIMEREAD() < 300)
  Wend
  TIMEINIT
  For n = 1 To 255
    If RI() = 1 Then DTR 1 Else DTR 0: U = U + 1
    While TIMEREAD() < n
    Wend
  Next n
DTR 1
  Label1.Caption = Str$(U) + "  "
End Sub
```

versely, a positive voltage leads to more negative DTR states.

The program starts by setting DTR to 1. This is important to ensure that the capacitor is not charged with the wrong polarity. The voltage across the capacitor does not rise above about 1 V, however, because the base-emitter diode in the transistor starts to conduct. In this quiescent state the transistor is fully turned on. In the measurement procedure proper, Timer1.Timer, a new measurement is carried out ever 500 ms. Initially DTR is turned off until RI changes state for the first time. At this point the voltage across the capacitor is equal to the threshold voltage of the transistor. This first loop has a timeout condition to trap errors and runs for at most 300 ms.

The main measurement loop is executed exactly 255 times, during which the number of cases where DTR is low is counted. The measurement always lasts 255 ms and produces 256 different possible results from 0 to 255 (**Figure 3**). This gives the same resolution as an 8-bit A/D converter.

Testing and calibration

Our first test delivered the following results:

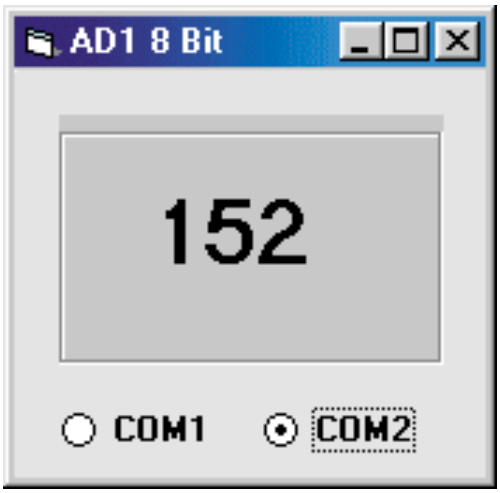


Figure 3. Display of digitised value.

Input	Display
open	118
0 V	110
–3.6 V	68
+3.6 V	152

We can see that an open-circuit input does not display zero, but a rather higher value. Comparing the measurement results for the voltages -3.6 V, 0 V and +3.6 V, we can immediately see that the difference between the negative voltage and zero is 42, as is the difference between the positive voltage and zero. This is encouraging, since it indicates that the converter is linear.

The digital values can now be converted into voltages. The following calculation does the trick:

Voltage = (Display – 110)/11.2

This expression contains the measured zero value and a multiplicative factor giving the number of digital steps per Volt. Both these values will vary from circuit to circuit. It is therefore worthwhile calibrating the circuit. Program AD2.frm (**Listing 2**) provides two slider controls which start off at preset values. The left-hand slider, with a range of 105 to 115 and an initial setting of 110, sets the zero offset. The right-hand slider, with a range of 124 down to 100 and an initial setting of 112, sets the slope. To use the program first short the input leads together and set the zero offset slider appropriately. Then a known voltage must be applied to the input, and the display set to the correct value using the right-hand slider. The A/D converter constructed here uses as a reference the voltage on the DTR output, which is not particularly

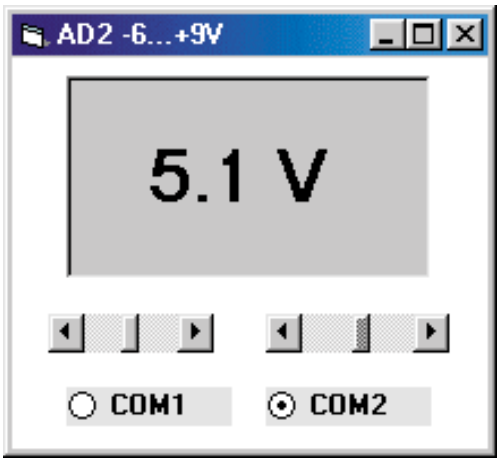


Figure 4. A complete voltmeter.

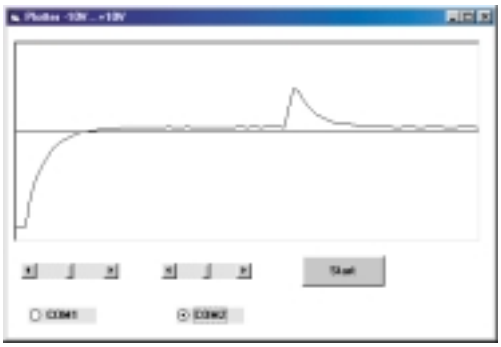


Figure 5. Voltage plotter Plotter2.frm.

reliable. Recalibration is therefore often necessary. **Figure 4** shows the program in action.

This simple A/D converter is reasonably accurate and reliable. The resolution is about 0.1 V, and the measurement range is about –6 V to +9 V. This can already find practical application, for example in testing batteries, and becomes all the more useful when the input voltage can be plotted (**Listing 3**). There are many other applications. **Figure 5** shows the measured voltage across a capacitor being briefly negatively and positively charged. The characteristic exponential discharge curves can be clearly seen.

Hardware improvements

When the possible sources of error in our simple A/D converter are considered, we can see that there are practically no aspects that cannot be improved.

– The basic accuracy depends on the voltage on the serial interface. It would be better to use a proper voltage reference, although that would make the circuit rather more complicated.
– The switching threshold of a comparator constructed from a simple NPN transistor is not 0 V, bit rather about 0.7 V. The exact value depends on the chosen transistor and on temperature. A temperature variation of one degree Celsius changes the threshold by about 2 mV.
– The ratio of the two 27 kΩ resistors affects the measurement; but if 5 % tolerance types are used, the contribution will be dominated by the other sources of error.
– The measurement results do NOT depend on the exact value of the capacitor. This is a particular advantage of this measurement method. Even if a 100 µF capacitor is used in place of the 47 µF capacitor, the results are not affected.

Listing 2. The modified Timer procedure in AD2.frm

```
Private Sub Timer1_Timer()
  RTS 1
  DTR 0
  U = 0
  REALTIME (True)
  TIMEINIT
  While (RI() = 0) And (TIMEREAD() < 300)
  Wend
  TIMEINIT
  For n = 1 To 255
    If RI() = 1 Then DTR 1 Else DTR 0: U = U + 1
    While TIMEREAD() < n
    Wend
  Next n
  REALTIME (False)
  U = (U - HScroll1.Value) / HScroll2.Value * 10
  U = Int(U * 10) / 10
  DTR 1
  Label1.Caption = Str$(U) + " V"
End Sub
```

It is relatively straightforward to improve on the design of the comparator (**Figure 6**). Instead of a single transistor, two are used. The pair of NPN transistors forms a differential amplifier, similar to the input stage of an operational amplifier. In this way the DC base-emitter voltage is reduced to a few millivolts. Temperature variation is no longer a problem because the two transistors are affected to the same extent. The total emitter current through the 4.7 kΩ resistor is about 2 mA. When the input voltage is zero the current is

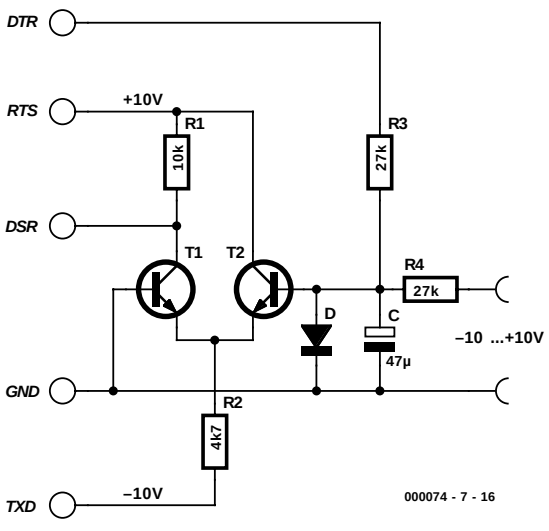


Figure 6. The improved comparator.

Listing 3. Plotter procedures Plotter2.frm

```
Dim y1, y2, x1, x2, n

Private Sub Command1_Click()
 n = 0
End Sub

Private Sub Timer1_Timer()
  RTS 1
  DTR 0
  U = 0
  REALTIME (True)
  TIMEINIT
  While (RI() = 0) And (TIMEREAD() < 300)
  Wend
  TIMEINIT
  For i = 1 To 255
    If RI() = 1 Then DTR 1 Else DTR 0: U = U + 1
    While TIMEREAD() < i
    Wend
  Next i
  REALTIME (False)
  U = (U - HScroll1.Value) / HScroll2.Value * 10
  DTR 1
  y2 = 100 - U * 10
  If n = 0 Then y1 = y2: Picture1.Cls
    x1 = n
  n = n + 5
  x2 = n
  Picture1.Line (x1, y1)-(x2, y2)
  y1 = y2
End Sub
```

divided equally between the two transistors. The voltage drop across the collector resistor of the second transistor is about 10 V, and so the collector voltage remains around zero and close to the switching threshold of the DSR input.

Unlike in the original circuit, the input transistor no longer limits the voltage across the capacitor during intervals between measurements, and so this voltage will rise. For this reason a silicon diode is included to limit the voltage to 0.6 V. The effect of all of these changes is that an open-circuit input gives a reading of zero, and that the measurement range is extended to 10 V with either polarity. **Figure 7** shows the construction of the improved converter.

Listing 4. Measurement routine with 1000 quantisation steps

```
Private Sub Timer1_Timer()
  RTS 1
  DTR 0
  U = 0
  REALTIME (True)
  TIMEINIT
  While (DSR() = 1) And (TIMEREAD() < 300)
  Wend
  TIMEINIT
  For n = 1 To 1000
    If DSR() = 0 Then DTR 1 Else DTR 0: U = U + 1
    While TIMEREAD() < n
    Wend
  Next n
  REALTIME (False)
  U = (U - HScroll1.Value) / HScroll2.Value * 10
  U = Int(U * 100) / 100
  DTR 1
  Label1.Caption = Str$(U) + " V"
End Sub
End Sub
```

Software optimisations

Changes are also needed in the software. First it must be taken into account that the DSR input signal is read in the inverted sense: the input is high when the capacitor voltage is above the threshold voltage. The improved characteristics of the measurement circuit also make it worthwhile to increase the accuracy of the measurements. The main loop is now executed 1000 times. **Listing 4** shows the results. With a total input range of 20 V we have a resolution on 0.02 V. As seen in **Figure 8**, a further decimal place can be shown on the display.

(000074-7)

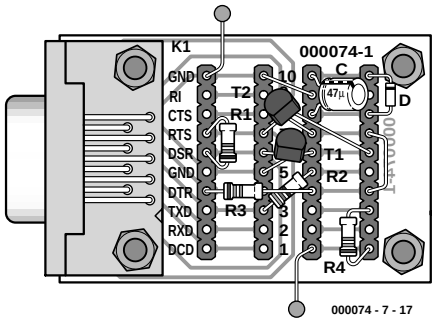


Figure 7. Construction of the improved comparator on the prototyping board.

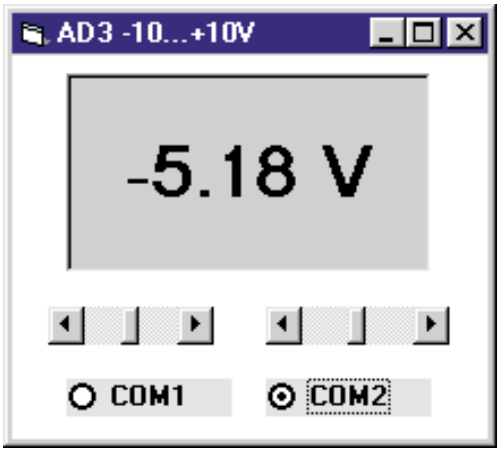


Figure 8. Result displayed to two decimal places.

HandsOn Technology

http://www.handsontec.com

creativity for tomorrow's better living...

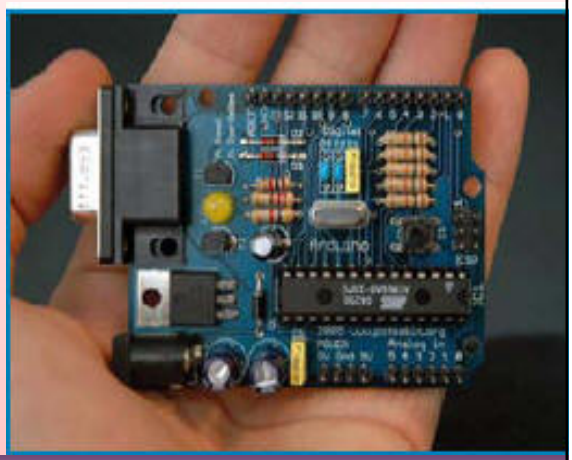
HandsOn Technology is a manufacturer of high quality educational and professional electronics kits and modules, uController development/evaluation boards. Inside you will find Electronic Kits and fully assembled and tested Modules for all skill levels. Please check back with us regularly as we will be adding many new kits and products to the site in the near future.

Do you want to stay up to date with electronics and computer technology? Always looking for useful hints, tips and interesting offers?

Inspiration and goals...

HandsOn Technology **provides a multimedia and interactive platform for everyone interested in electronics. From beginner to diehard, from student to lecturer... Information, education, inspiration and entertainment. Analog and digital; practical and theoretical; software and hardware...**

HandsOn Technology provides Designs, ideas and solutions for today's engineers and electronics hobbyists.



Creativity for tomorrow's better living...

HandsOn Technology believes everyone should have the tools, hardware, and resources to play with cool electronic gadgetry. HandsOn Technology's goal is to get our "hands On" current technology and information and pass it on to you! We set out to make finding the parts and information you need easier, more intuitive, and affordable so you can create your awesome projects. By getting technology in your hands, we think everyone is better off

We here at HandsOn like to think that we exist in the same group as our customers >> curious students, engineers, prototypers, and hobbyists who love to create and share. We are snowboarders and rock-climbers, painters and musicians, engineers and writers - but we all have one thing in common...we love electronics! We want to use electronics to make art projects, gadgets, and robots. We live, eat, and breathe this stuff!!

If you have more questions, go ahead and poke around the website, or send an email to sales@handsontec.com. And as always, feel free to let your geek shine - around here, we encourage it...