APRIL 2002
£3.20

ELEKTOR ELECTRONICS

THE ELECTRONICS & COMPUTER MAGAZINE

www.elektor-electronics.co.uk

INTERFACE IT!

CompactFlash-to-IDE
I²C-to-RCX
1-Wire-to-RS232

LPT/DMX Interface

Multi-standard Infrared Receiver

Serial Ports under Windows

EEDTs Pro Protocol

B²Spice Reader Offer

AVR ISD Competition

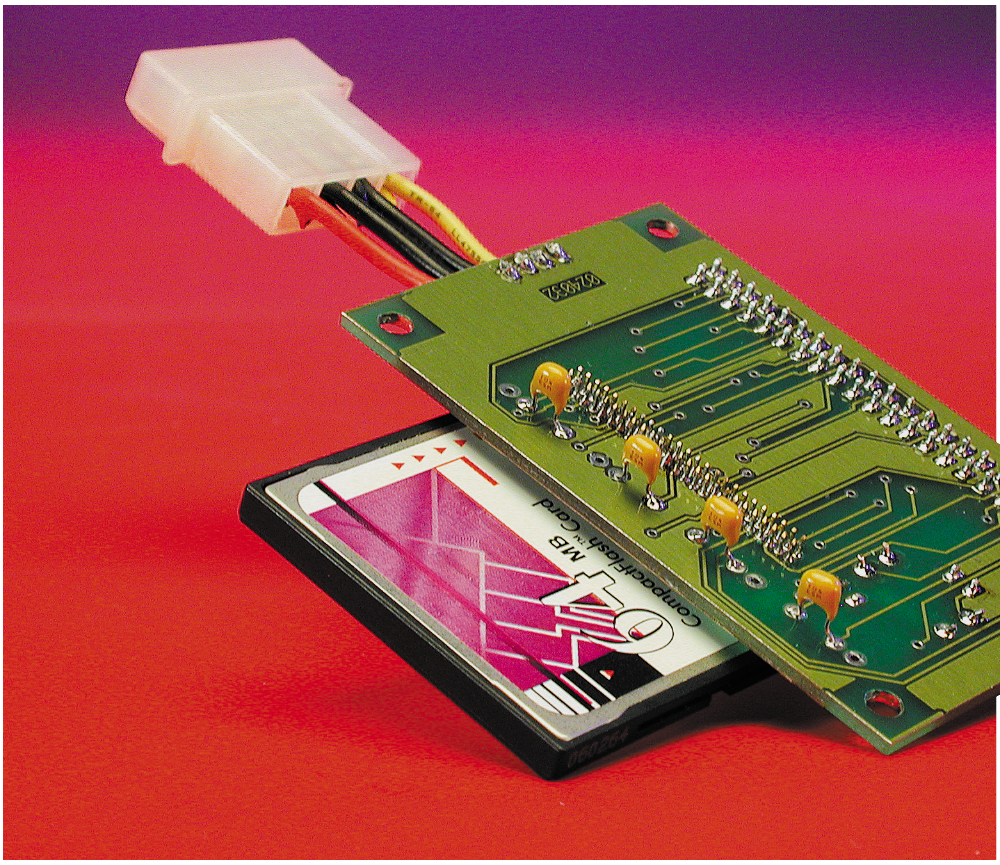
9 770268 451098 04

CompactFlash Drive on IDE Bus

solid-state memory for the PC

Design by Paul Goossens

CompactFlash cards are memory media that retain their contents without a supply voltage. They are used in digital cameras, among other things. Thanks to the 'intelligence' present in such a card, it can also be easily connected to a PC, for example for use as a 'solid-state drive'. The super-simple adapter described here makes it easy to connect all types of CompactFlash cards to a PC.



A CompactFlash card is a small memory card, originally developed by Sandisk, with dimensions of approximately 4×4 cm. Since it uses non-volatile memory cells, the contents of the memory are retained for years, even without a separate source of power. Such cards are used particularly often in Digicams.

Presently, the memory capacity can be as much as 1 GBytes. IBM, by the way, even supplies mini hard disks in CompactFlash format with capacities of up to 1 GBytes.

Figure 1 shows the dimensions and pin designations of the CompactFlash card (which we will abbreviate to 'CF card'). Two different thicknesses are specified: 3.3 mm for Type I cards and 5 mm for Type II. Most solid-state cards are Type I, while CF hard disks are Type II.

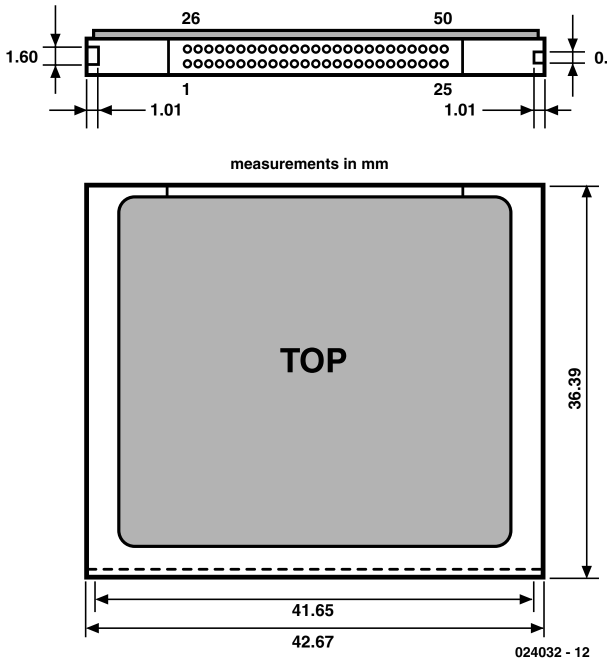


Figure 1. CompactFlash card dimensions and pin designations.

Figure 1: Pin designations

Function	Pin	Pin	Function
GND	1	26	$\overline{\text{CD1}}$
D03	2	27	D11
D04	3	28	D12
D05	4	29	D13
D06	5	30	D14
D07	6	31	D15
$\overline{\text{CE1}}$; $\overline{\text{CE1}}$; $\overline{\text{CS0}}$	7	32	$\overline{\text{CE2}}$; $\overline{\text{CE2}}$; $\overline{\text{CS1}}$
A10	8	33	$\overline{\text{VS1}}$
$\overline{\text{OE}}$	9	34	$\overline{\text{IORD}}$
A09	10	35	$\overline{\text{IOWR}}$
A08	11	36	$\overline{\text{WE}}$
A07	12	37	RDY; BSY; $\overline{\text{IREQ}}$; INTRQ
VCC	13	38	VCC
A06	14	39	$\overline{\text{CSEL}}$
A05	15	40	$\overline{\text{VS2}}$
A04	16	41	RESET; RESET; $\overline{\text{RESET}}$
A03	17	42	$\overline{\text{WAIT}}$; $\overline{\text{WAIT}}$; IORDY
A02	18	43	$\overline{\text{INPACK}}$
A01	19	44	$\overline{\text{REG}}$
A00	20	45	BVD2; $\overline{\text{SPKR}}$; $\overline{\text{DSAP}}$
D00	21	46	BVD1; $\overline{\text{STSCHG}}$; $\overline{\text{PDIAG}}$
D01	22	47	D08

Notes:
$\overline{\text{XX}}$ = inverted signal
For pins with three designations (aa; bb; cc):
aa: in Memory mode
bb: in I/O mode
cc: in True IDE mode

The prices of CF cards have dropped sharply in the last year, so you can presently buy a card with a capacity of 64 MBytes for less than £ 60.

Internally, a CF card consists of a number of memory modules driven by a their own controller. Data input and output take place in parallel, just as with a hard disk. Thanks to this 'intelligent' design, such a card can easily work together with a microcontroller system or PC.

PCMCIA adapters are available to allow CF cards to be used with notebooks. These are actually nothing more than extenders for the connection bus.

There is a wide variety of commercially available card readers, fitted with USB, serial-port or parallel-port connectors, to allow CF cards to be used with PCs.

Given the affordable prices and non-volatile memory characteristics of CF cards, it is unquestionably interesting to use such a card as an extra storage medium for the PC, for instance as a replacement for the floppy-disk drive. For this, you do not even have to buy a separate reader, since the circuit described in this article provides a much simpler solution.

Passive circuit

The intelligence present in the card allows it to be used in three different modes. One of these is the 'true IDE mode', which is obtained by connecting pin 9 to ground. In this mode, the external behaviour of the card is the same as that of a hard disk drive with an IDE interface. The only other thing we need is a sort of adapter with two connectors to allow the CF card to be connected to the IDE bus of the PC.

The schematic diagram of this adapter circuit is shown in **Figure 2**. It primarily consists of two connectors, augmented by some passive components. K1 is the connector for the IDE bus, while K2 provides the connection to the CF card.

The remaining components can be quickly described. LED D1 makes the read and write activities of the CF card visible, with R1 determining how much current flows through the LED. Pin 39 of the CF connector is connected to +5 V via resistor R2. This makes the CF card act as a 'master' on the IDE bus. If pin 39 is connected to ground by means of jumper JP1, the CF card will act as a 'slave' on the bus. The two capacitors C1 and C2 decouple the supply voltage.

There is yet another connector (K3) on the circuit board; it is used to connect the supply voltage. In spite of the fact that the CF card needs only +5 V and ground, a four-way connector has been chosen here in order to remain compatible with the supply connectors inside the PC. The 12-V pins of these connectors are thus not used.

Construction

The double-sided printed circuit board shown in **Figure 3** is quite handy for building the circuit. Although all that we have here is two connectors and a few auxiliary components, the 50-pin CF connector is a bit difficult. The pin pitch of a normal connector is 0.1 inch, but with a CF connector it is only 0.05 inch (slightly more than 1 mm). It is thus practically impossible to link the two connectors together using loose lengths of wire. Even with the printed circuit board, it is essential to work carefully and use a fine soldering iron tip.

If your enthusiasm has been aroused and you want to build this circuit, we would like to give you some good advice: first see whether you can obtain the CF connector! It is available from Farnell, among others, but most electronic components shops won't have such a connector in stock. Once you have the connector, you can buy the circuit board and the rest of the components.

The connectors, the resistors and the LED are all fitted to the top side of the board. The two decoupling capacitors are fitted to the bottom side, which means that you must cut their leads short on the top side so they won't touch the CF card.

For the power supply connector, take an extension cable or splitter for the PC power supply (a standard item that can be bought in any PC shop) and cut off the plug. Then solder the wires attached to the (male) mating connector to the K3 position. The red wire is +5 V, the two black wires are ground and the yellow wire is +12 V.

After all of this is done, you can fit JP1 as needed to have the CF card be seen by the PC as a slave drive (if there is already a hard disk or CD-ROM drive on the IDE bus in question and it is configured as the master). Without the jumper, the card is automatically the master.

Following this, the adapter board can be fitted into the PC and connected. One option is to make a slot in a blank 5.25-inch front panel and

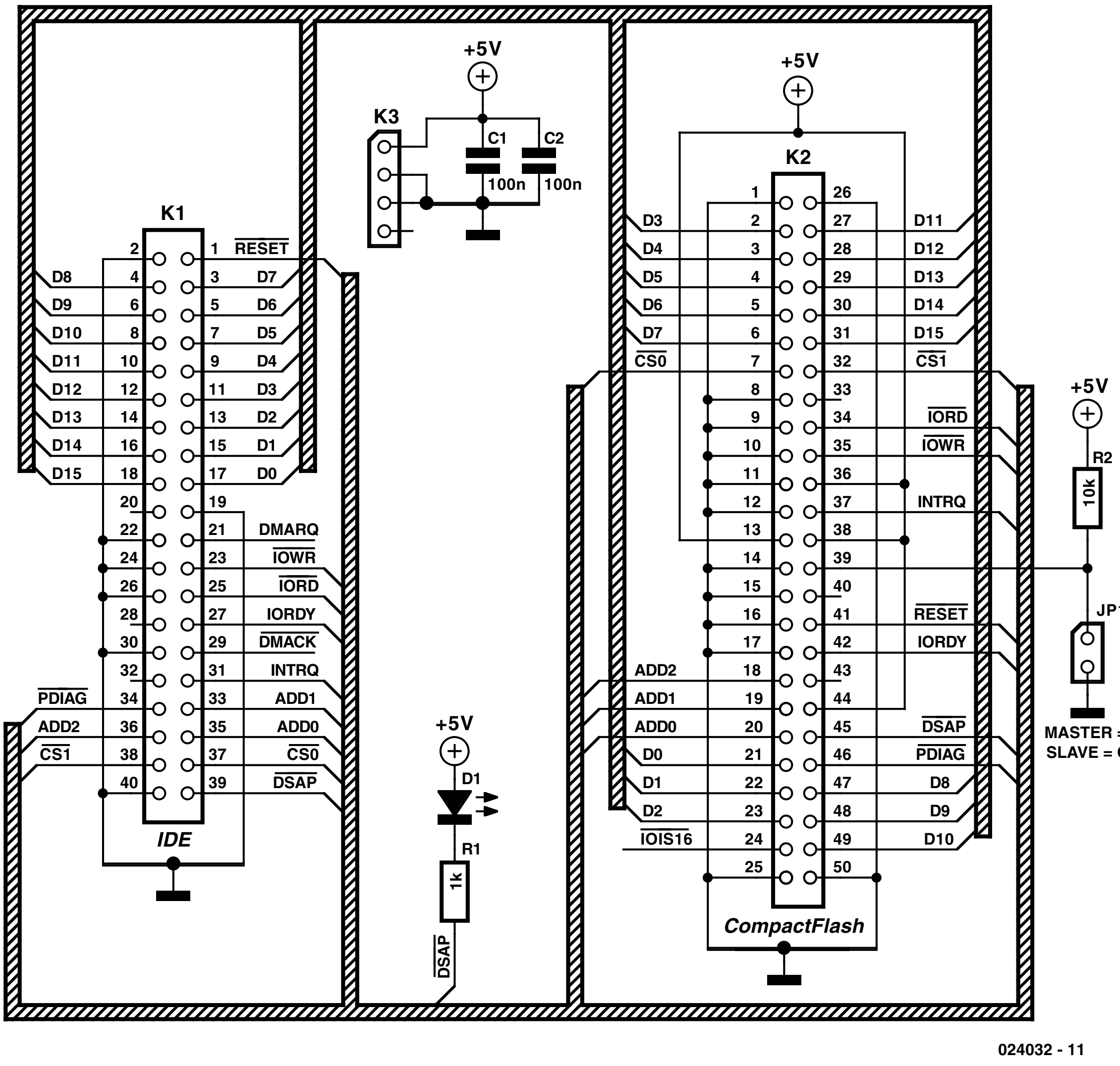
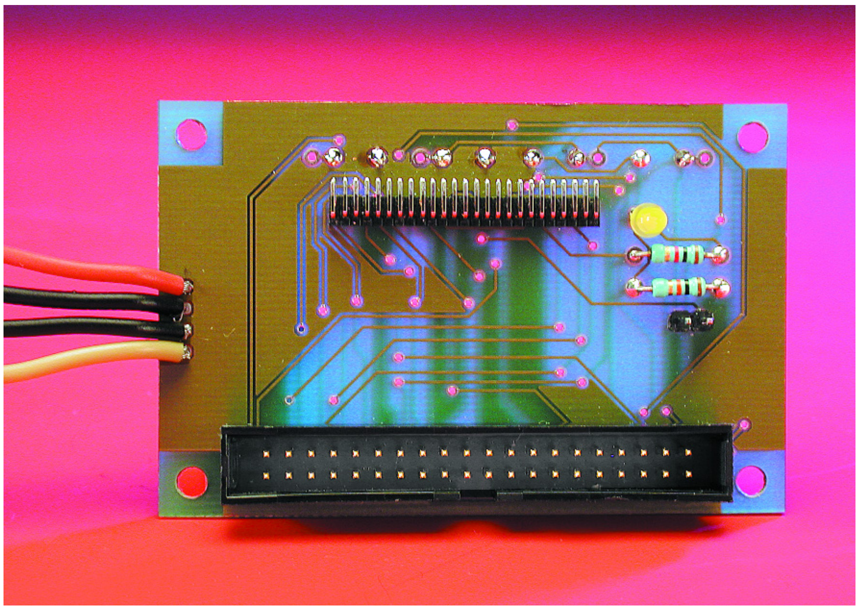
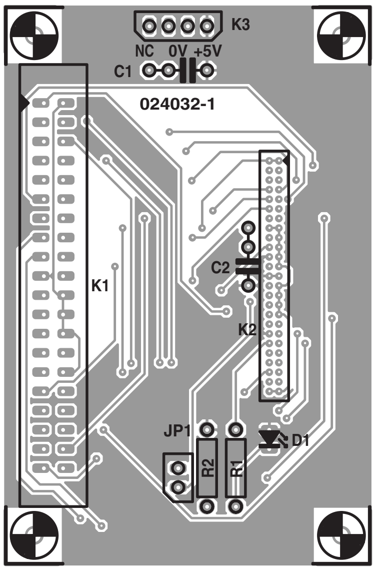
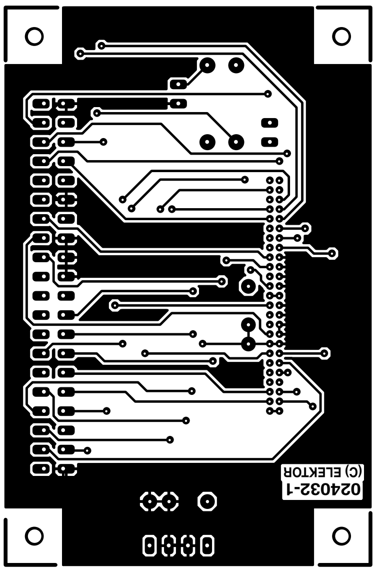
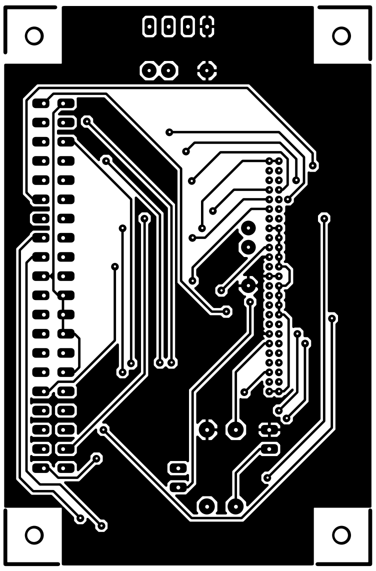


Figure 2. Schematic diagram of the adapter: no active components.









fit the card on the back of the panel, **upside down**. K2 must be located just behind the opening. The circuit board can be connected to one of the IDE busses on the motherboard using K1 and a standard IDE cable. Connect the mating connector attached to K3 to one of the PC's free power supply plugs.

Now everything is ready for use. To be on the safe side, we recommend that you insert or remove the CF card only when the PC is off.

After you have connected a CF card for the first time and started up the computer, Windows will report that a driver for the new device is being installed. Normally, the driver will already be present and the rest of the procedure is automatic, but sometimes you may have to insert the Windows installation CD in the CD-ROM drive.

After the driver has been installed, Windows will automatically recognise the CF card as a sort of hard disk drive and you can read and write files in the usual manner.

In concluding this article, we wish to give you a **clear warning: always plug the CF card into the connector on the circuit board the right way around.** This means that **the top side of the CF card must face towards the circuit board**. This is also why the board is fitted in the computer with the components to the bottom. The photo at the beginning of the article clearly shows what we mean. If you want to play safe here, you can fit a small nib in the insertion slot for the CF card. As can be seen in Figure 1, the guide grooves on the two opposite edges of the CF card have different widths, and you can take advantage of this.

(024032-1)

COMPONENTS LIST

Resistors:
R1 = 1kΩ
R2 = 10kΩ

Capacitors:
C1,C2 = 100nF

Semiconductors:
D1 = LED, yellow, low current

Miscellaneous:
K1 = 40-way boxheader
K2 = 50-way pinheader, angled pins, 0.05”-grid (Farnell # 3078127)
JP1 = 2-way pinheader with jumper
Extension cable for PC supply
PCB, order code **024032-1** (see Readers Services page)

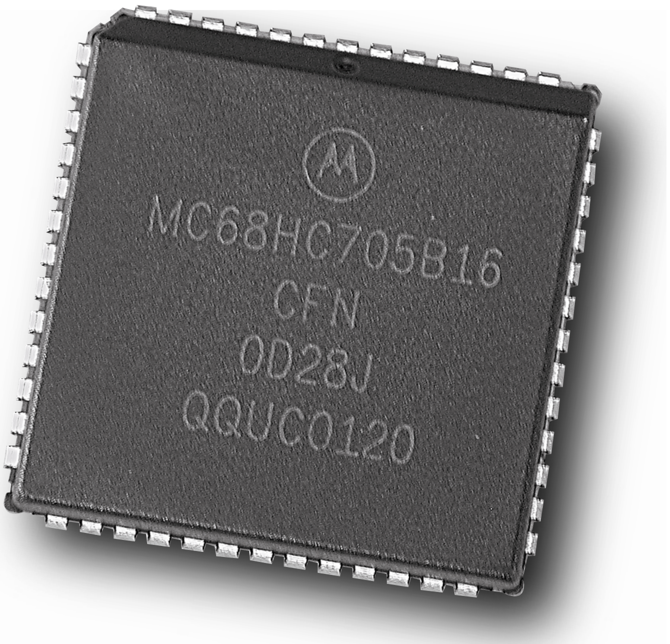
Figure 3. This double-sided printed circuit board makes it easier to build the circuit, especially as regards fitting K2.

EEDTS Pro Control Unit Protocol

protocol extensions

By S. van de Vries

Without question, the EEDTS Pro protocol has been the subject of most discussions related to EEDTS Pro, due to the fact that it differs from the old EEDTS and the Motorola protocol.



Furthermore, many users and software suppliers have found it difficult to convert addresses and date to the trits coding required by the EDTS controller.

Consequently, the protocol has been extended to include the command set listed in **Table 1**.

Naturally, the existing command set (as described in the EEDTS Pro book) is still fully supported, and the new control unit is downwards compatible with the old version.

The Märklin protocol formed the starting point for this move, but since that protocol does not have any commands for the extra functions, an extra byte has been specifically added (in particular, for the locomotive control commands).

General command structure

The EEDTS Pro control unit communicates with the PC via the RS232 port using fixed settings of 9600 baud with no parity, eight data bits and one stop bit. In order to ensure correct data transfers, a design has been used in which a response byte is sent back for every byte that is sent to the control unit.

A control unit instruction may consists of 1, 2, 3 or 4 bytes, with the first byte being the command byte.

In general, the command byte is sent back *verbatim* as an indication that the instruction was correctly received, except in the case of the return commands (single-byte control unit instructions), for which return information is sent back directly in order to achieve a high data rate.

If the control unit cannot send the information to the track (for instance, if the booster is out of service due to a short circuit), the value '65' is sent back following the command byte, in order to inform the PC of this situation.

In order to inform the control unit in turn that the booster is out of service, a small modification to the booster interface is necessary. This will be described in the following instalment, along with the addressing of the keyboard and stand-alone controller.

Locomotive command

The first byte of the locomotive command can have a value of 0–15. This value represents the speed, bearing in mind that in the old format, speed level '1' is the 'reverse' command (in the Märklin protocol, it is '15').

The second byte that is sent represents the locomotive address. The first 80 addresses correspond to the 80 available addresses in the Märklin format. However, since there is room for 256 addresses in the new format, the command has been extended to 256 addresses.

The only existing 'standard' for this sequence is to be found at Uhlenbock, which is why this sequence has been chosen.

The third byte provides information regarding the format, direction of travel and extra functions. The meanings of the individual bits in the old format are shown in **Table 2**, while the same information for the new format is shown in **Table 3**. For convenience, the values are also shown in decimal notation.

It is now possible to generate old-format functions. An important difference between the two tables is that in the old format, all four functions are modified by each byte, while in the 'new style' table a separate byte is used to set each function.

In the old format, if it is not desired to modify the extra func-

Table 1. The extended command set.

	Command byte	Normal response	Short circuit	Second byte	Response	Third byte	Response	Fourth byte	Response
Locomotive command	0 - 15	0 - 15	65	0 - 255	0 - 255	See Tables 2 & 3	Third byte		
Last changed	190	1..64	65						
Return-signalling module	192 - 255	0 - 255	65						
Turnout (switch) reset	32	32	65						
Turnout (switch) to the left	33	33	65	0 - 255	0 - 255				
Turnout (switch) to the right	34	34	65	0 - 255	0 - 255				
Program manual controller	40	40	65	16 - 23 24 - 31	0 - 255 Speed	0 - 255	Previous address	0 - 255	Previous status
Read manual controller	41	41	65	16 - 23 24 - 31	0 - 255 Speed	0	Address	0	Status

tions, a '0' or a '3' is sent (the 'n' in the table indicates that F1–F4 do not change). The new format also includes the 'reverse' and 'forward' commands.

The observant reader may have already noticed that two bits (b1 and b2) are reserved for the F0 function and that these bits always have the same values in the table ('00' or '01'). The values '01' and '10' are also allowed, but they are reserved for assignment to future functions.

Return commands

Last changed

The 'last changed' command ('190') will doubtless prove to be useful for real-time control in layouts having a large number of return-signalling units. The response byte contains the address of the return signaller whose input state has changed (with respect to the time at which this unit was last read). This makes it possible to quickly determine which unit must be read.

Return-signalling modules

The return-signalling units can be read using commands 192–255.

Command '192' is used to read the first unit, '193' is used to read the second unit and so on, up to '255' for the 64th EEDTS return signalling unit (or 9–16 of the 32nd Märklin S88).

The value that the control unit sends back provides a binary representation of the inputs (in the case of a detector module, this is the locomotive address).

Turnout (switch) commands

The turnout (switch) commands are exactly the same as the Märklin commands. For instance, command '33-(1–4)' activates the 'green' output of a k73, while command '34-1–4)' activates the 'red' output. Even the peculiarity that '33-0' and '34-0' represent the final two outputs of decoder 64 has been implemented. Turnout activation is rescinded using command '32'.

These commands limit the number of decoders to 64. If a larger number of decoders must be supported, recourse can be made to

Table 2. The old Motorola format.

b8	b7	b6	b5	b4	b3	b2	b1	Decimal	F0	F1	F2	F3	F4
0	0	0	0	0	0	0	0	0	0	n	n	n	n
0	0	0	0	0	0	1	1	3	1	n	n	n	n
0	1	0	0	0	1	0	0	68	0	0	0	0	0
0	1	0	0	0	1	1	1	71	1	0	0	0	0
0	1	0	0	1	0	0	0	72	0	1	0	0	0
0	1	0	0	1	0	1	1	75	1	1	0	0	0
0	1	0	0	1	1	0	0	76	0	1	1	0	0
0	1	0	0	1	1	1	1	79	1	1	1	0	0
0	1	0	1	0	0	0	0	80	0	0	0	1	0
0	1	0	1	0	0	1	1	83	1	0	0	1	0
0	1	0	1	0	1	0	0	84	0	1	0	1	0
0	1	0	1	0	1	1	1	87	1	1	0	1	0
0	1	0	1	1	0	0	0	88	0	1	1	1	0
0	1	0	1	1	0	1	1	91	1	1	1	1	0
0	1	0	1	1	1	0	0	92	0	0	0	0	1
0	1	0	1	1	1	1	1	95	1	0	0	0	1
0	1	1	0	0	0	0	0	96	0	1	0	0	1
0	1	1	0	0	0	1	1	99	1	1	0	0	1
0	1	1	0	0	1	0	0	100	0	0	1	0	1
0	1	1	0	0	1	1	1	103	1	0	1	0	1
0	1	1	0	1	0	0	0	104	0	1	1	0	1
0	1	1	0	1	0	1	1	107	1	1	1	0	1
0	1	1	0	0	1	0	0	108	0	0	0	1	1
0	1	1	0	0	1	1	1	111	1	0	0	1	1
0	1	1	0	1	1	0	0	112	0	1	0	1	1
0	1	1	0	1	1	1	1	115	1	1	0	1	1
0	1	1	1	1	1	0	0	116	0	1	1	1	1
0	1	1	1	1	1	1	1	119	1	1	1	1	1

the normal EEDTS command set, which can address up to 240 decoders.

Table 3. The new Motorola format.

b8	b7	b6	b5	b4	b3	b2	b1	Decimal	Description	
0	0	0	0	0	1	0	0	4	F1 off	F0 off
0	0	0	0	0	1	1	1	7	F1 off	F0 on
0	0	0	0	1	0	0	0	8	F1 on	F0 off
0	0	0	0	1	0	1	1	11	F1 on	F0 on
0	0	0	0	1	1	0	0	12	F2 off	F0 off
0	0	0	0	1	1	1	1	15	F2 off	F0 on
0	0	0	1	0	0	0	0	16	F2 on	F0 off
0	0	0	1	0	0	1	1	19	F2 on	F0 on
0	0	0	1	0	1	0	0	20	F3 off	F0 off
0	0	0	1	0	1	1	1	23	F3 off	F0 on
0	0	0	1	1	0	0	0	24	F3 on	F0 off
0	0	0	1	1	0	1	1	27	F3 on	F0 on
0	0	0	1	1	1	0	0	28	F4 off	F0 off
0	0	0	1	1	1	1	1	31	F4 off	F0 on
0	0	1	0	0	0	0	0	32	F4 on	F0 off
0	0	1	0	0	0	1	1	35	F4 on	F0 on
0	0	1	0	0	1	0	0	36	Reverse	F0 off
0	0	1	0	0	1	1	1	39	Reverse	F0 on
0	0	1	0	1	0	0	0	40	Forward	F0 off
0	0	1	0	1	0	1	1	43	Forward	F0 on

Programming manual controllers

Program manual controller

Command '40' can be used to program the manual controllers. The command byte ('40') is followed by a second byte that can have a value of 16–23. The value '16' selects controller 1, '17' selects controller 2 and so on, up to '23' for controller 8.

If a value in the range of 16–23 is chosen, the address modification is temporary and is no longer valid after a power-down. In order to achieve a permanent address modification, a value in the range of 24–31 must be chosen ('21' for controller 1 etc., up to '31' for controller 8).

Once the control unit has received the second byte, it will send back the controller status. This can lie in the range of '0' (control rotated fully to the left) to '255' (control rotated fully to the right).

The third byte determines the address that must be set for the controller, while the fourth byte indicates the format of the controller and the functions that are already enabled or disabled in the locomotive (or in the PC software). In this way, for example, the control unit knows that if the F3 function is enabled, it must send the 'F0 off' command to the locomotive when the F0 button is pressed on the manual controller.

The bits in the fourth byte have the following meanings:

b8	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**
s/h	o/n	f/r	F4	F3	F2	F1	F0

For example, if the fourth byte is assigned a value of '36' (binary '00100100'), the controller settings are as follows:

s/h = 0 format selection:
1 = format set by software
0 = format set by hardware
o/n = 0 format:
1 = old format
0 = new format
f/r = 0 direction of travel:
1 = forward
0 = reverse
F4 = 0 (F4 off)
F3 = 0 (F3 off)
F2 = 1 (F2 on)
F1 = 0 (F1 off)
F0 = 0 (F0 off)

If the s/h bit is '1', this means that the format (old or new) is not determined by the selection diode in the manual controller, but instead by the o/n bit. If the s/h bit is '0', the format is determined by the selection diode in the manual controller.

After the third (address) byte has been sent to the control unit, the control unit sends back the address set for the manual controller (at the time that the command was sent). Following the fourth byte, the control unit sends back the previously read controller status in the response byte. The response data can be used by the EEDTS Pro software, for example, to restore the original values when control is passed from a software controller to a manual controller.

The eight manual controllers cannot be disabled and will continually send information to the track. You should therefore avoid having more than one controller set to the same address (except for address '0') or having the PC send commands to addresses that are being used by manual controllers.

In order to avoid problems arising from the manual controllers, it is a good idea to first set all manual controllers (or all manual controllers that are not actively in use) to address '0' when starting up your own program.

Read manual controller

Command '41', which is used to read out a manual controller, is closely related to the 'program manual controller' command. However, the 'read manual controller' command only reads data and does not change any controller settings. In this case as well, controller status is returned following the second byte.

The values following the third and fourth bytes are the same as for the 'program manual controller' command, with the understanding that here it does not matter what value is sent by the PC, since the control unit does not do anything with these values.

This completes our presentation of the most important changes in the new control unit protocol. The new microcontroller is available from Readers Services under order number **010088-41** and can simply be fitted on the existing control unit circuit board.

(010088-2)

Serial Ports under Windows

with an example in Delphi

By Paul Goossens

The use of I/O functions in Windows programs is still a grey area for most programmers. In this article, we shed some light on the use of the PC's RS 232 ports in a Windows compatible way.

When you wish to use certain devices within a computer, you'll need routines that control them. With older home computers and PCs, writing such a routine was fairly simple. There was only ever one program running at a time so you didn't have to take into account that other programs could access the same device. A well-written program would also make sure that the device was set back to its default state at the end of the program's execution.

With the introduction of Windows the PC world changed overnight. It became possible to run several programs simultaneously. Although this was already a common practice in the professional computing world, for the home user it appeared to be close to magic!

Those home users who were used to control the hardware directly in their programs were unfortunately confronted with a new problem. What happens when two programs attempt to access the same device at the same time? As long as you could make sure yourself that this situation could never occur, there wasn't a problem. Worse still, in some programming languages the facility to access the hardware directly was left out. In some cases it was still possible to use the processor's I/O instructions via programming tricks.

In this article we will show that the correct programming of the serial interface under Windows is not as difficult as most people imagine. As well as giving your programs that professional look, they should also run without problems even in the most recent versions of Windows.

Operating systems

A short but simple description of an operating system is that it is nothing but a (large) program that drives and controls all components within a computer. Some of these components are the mouse, keyboard, display, memory, sound, and so on.

The task of the operating system is to isolate the user program from the need to know the exact hardware configuration of the computer on which it runs. The user program gives instructions to the operating system, which then takes care of the rest. This makes it possible for a program to run on computers with different hardware configurations, without having to adapt the program. The only requirements are that the same operating system is installed on these computers and that the hardware is up to the task.

Drivers

The actual control of the hardware occurs via so-called drivers. For each

Table 1

Useful functions in the API for controlling the serial interface.

FileCreate	Opens a file
EscapeCommFunction	Control of the DTR and RTS outputs
GetCommStatus	Request the status of the input pins and the receive buffer
ReadFile	Read the receive buffer
WriteFile	Write to the output buffer of the serial port
GetCommState	Request the current state of the serial port
SetCommState	Set the state of the serial port
GetCommMask	Read the event mask. This mask determines which incidents generate a Windows 'event'
SetCommMask	Set the event mask
ClearCommError	Get information regarding the last error in the serial interface

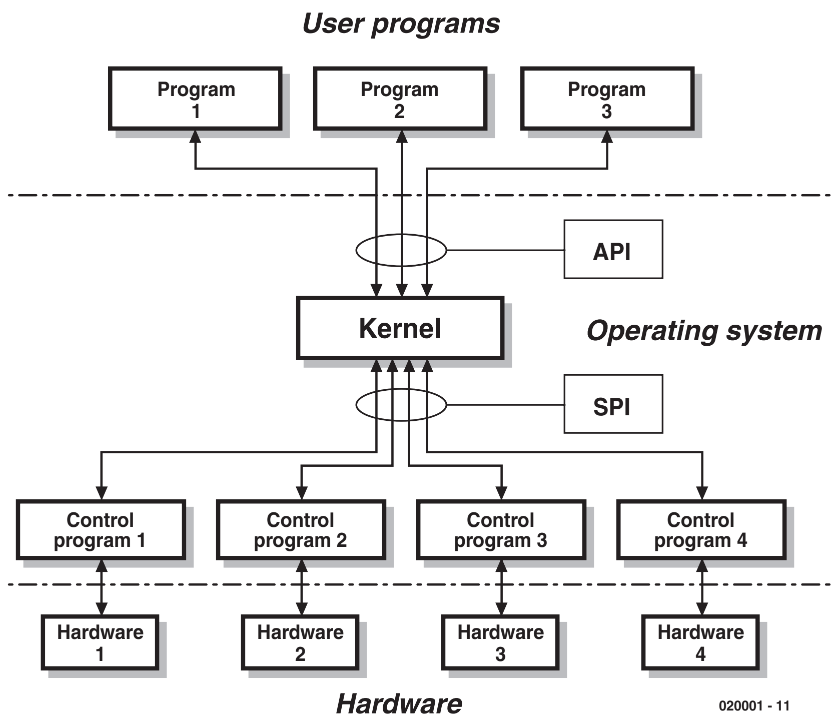


Figure 1. Simplified structure of a computer system.

DLLs

Windows makes extensive use of DLL files. These are libraries that can be used by any program. Most functions of the Windows API are made accessible by these DLLs. When a program makes use of a DLL it is dynamically linked with that program (hence the name: Dynamically Linked Library).
To find out which functions are exported by Windows DLLs, you could use the seemingly unimportant 'quickview.exe' program, which is included with most versions of Windows. This program can be used to open DLLs. Quickview then presents various information about the DLL, including the names of its functions. This way a good overview can be obtained of the capabilities of a DLL.
Quickview can be run by right-clicking with the mouse on a DLL file.

Table 2

The most significant fields in a DCB record.

DCBLength	The length of the DCB.
BaudRate	Speed of the connection in baud.
Binary	Binary connection (has to be TRUE in Windows).
Parity	Parity check (on or off).
fOutxCtsFlow	Enable or disable CTS flow control when sending information. When enabled, the driver stops transmitting from the serial port while the CTS signal is inactive.
fOutxDsrFlow	The same as fOutxXCtsFlow, but for the DSR signal.
fDtrControl	Settings for the DTR signal. This can have the following values[1]: *DTR_CONTROL_DISABLE : DTR becomes low when the port is opened.* *DTR_CONTROL_ENABLE : DTR becomes high when the port is opened.* *DTR_CONTROL_HANDSHAKE : DTR is used for handshaking.*
fDsrSensitivity	When active, the driver will ignore all data received when DSR is low.

[1] The DTR signal is controlled by the EscapeCommFunction function, except when the DTR_CONTROL_HANDSHAKE option is active.

of the components in the computer the operating system requires a specific driver. In this way the operating system can make use of the latest hardware, as long as the manufacturer includes a driver.

APIs

The way in which user programs communicate with the operating system is established by the API (Application Programming Interface). As the name implies, this is an interface between the user programs and the operating system. Compilers usually have libraries and header-files included which take care of the API calls. When this isn't the case, you should refer to the documentation of the operating system. The API is usually included with the SDK (Software Development Kit) of the operating system. These SDKs are usually available as a free download from the operating system provider.

As can be seen in **Figure 1**, a user program only communicates with the API of the operating system. It is of no concern to the programmer how the operating system works internally, as long as the API ensures that the operating system does what the program asks of it. The API then passes the requests on to the kernel for further processing. Where a program requires that the hardware is used, the kernel gets help from the accompanying driver program. This then communicates directly with the hardware.

The communication between the driver and kernel isn't via the usual API (Application Programming Interface), but via the SPI (System Programming Interface).

The serial interface

The serial interface is commonly used in homemade circuits. These circuits usually contain a microprocessor with an on-chip serial port. It is a simple matter to give commands to the circuit via this port, or to request information from it. Even where no use is made of a controller it is still possible to use discrete logic circuits to incorporate serial I/O — an example of this can be seen in the DCI-PLC article in the June 2001 *Elektor Electronics* issue.

Computer devices such as printer ports, serial interfaces, I/O-devices, etc. are treated by Windows (and most other operating systems) as special types of file. This means that a program first has to open this 'file' before being able to use the device. Once such a 'file' has been opened there will be several extra properties and functions that can be applied to it. Those functions that are relevant to the serial interface are listed in

Listing I

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

type
  TForm1 = class(TForm)
    GroupBox1: TGroupBox;
    RadioButton1: TRadioButton;
    RadioButton2: TRadioButton;
    RadioButton3: TRadioButton;
    RadioButton4: TRadioButton;
    GroupBox2: TGroupBox;
    RadioButton5: TRadioButton;
    RadioButton6: TRadioButton;
    RadioButton7: TRadioButton;
    RadioButton8: TRadioButton;
    RadioButton9: TRadioButton;
    Button1: TButton;
    Button2: TButton;
    Label1: TLabel;
    Label2: TLabel;
    CheckBox1: TCheckBox;
    CheckBox2: TCheckBox;
    CheckBox3: TCheckBox;
    CheckBox4: TCheckBox;
    CheckBox5: TCheckBox;
    CheckBox6: TCheckBox;
    Button3: TButton;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    GroupBox3: TGroupBox;
    RadioButton10: TRadioButton;
    RadioButton11: TRadioButton;
    RadioButton12: TRadioButton;
    GroupBox4: TGroupBox;
    RadioButton13: TRadioButton;
    RadioButton14: TRadioButton;
    RadioButton15: TRadioButton;
    Label7: TLabel;
    Edit1: TEdit;
    Label8: TLabel;
    Button4: TButton;
    Edit2: TEdit;
    procedure FormActivate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure RadioButtonComPort(Sender: TObject);
    procedure RadioButtonBaudRate(Sender: TObject);
    procedure CheckBox1Click(Sender: TObject);
    procedure CheckBox2Click(Sender: TObject);
    procedure UpdateClick(Sender: TObject);
    procedure ComboBox1KeyPress(Sender: TObject; var Key: Char);
    procedure RadioButtonParityClick(Sender: TObject);
    procedure RadioButtonStopBitsClick(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Edit2KeyPress(Sender: TObject; var Key: Char);
  private
    { Private declarations }
    hPort       : LongInt;      {handle for the serial port}
    dcbCom      : TDCB;         {record which holds the properties for the}
                                {opened COM-port}
    Open        : Boolean;      {Is COM-port open or not?}
    ComPort     : Integer;      {The COM-port number}
    BaudRate    : LongInt;      {desired Baudrate}
    Parity      : Byte;         {Parity}
```

Table 1.

Before you start programming for the serial port you have to make one important decision. The serial interface can be opened in two very different ways.

The file can be opened in either OVERLAPPED or NON-OVERLAPPED mode. The OVERLAPPED mode lets Windows process any actions in the background, while the main program continues operating. With the NON-OVERLAPPED mode the program has to wait for Windows to complete any serial port actions, before continuing with its own execution.

The advantage of OVERLAPPED is that the program is not unnecessarily interrupted every time the serial port is used. The big disadvantage of this choice is that you don't immediately know if the last command has completed successfully. It is therefore necessary to write an 'event-handler', which is called every time Windows completes a serial command. It is unfortunately fairly easy to make mistakes in this routine, which are difficult to find (such as deadlock and race conditions). In fact, only programmers experienced in parallel programming should use this mode.

The big advantage of NON-OVERLAPPED is that the result of the command is available after it completes, without having to worry if the operation itself has completed. In our example Delphi program we've used NON-OVERLAPPED, partly to avoid making it needlessly complex.

Delphi

As an example, we have written a simple program in Delphi (see **Listing 1**). This program uses the most important API calls that are relevant to the use of the serial port.

The 'Button1Click' routine is called when the user clicks on the OPEN button. This routine opens a file with a name corresponding to the selected COM port (COM1, COM2, etc.).

Opening the file is only half of the work. Next the serial port has to be set up. First the current settings are requested using 'GetCommState'. This Windows routine fills the 'dcb-

Com' record with the current state of the serial port. A list of the most interesting fields of this record is shown in **Table 2**. Next some of the fields in this record are set. Lastly, these settings are made active by calling 'SetCommState'. The serial port is now opened with our required settings.

In the example program it is possible to enter some text and then transmit it with a click of the button. The routine that deals with the transmission of text is 'Button4Click', which is very straighforward. It transmits the text using the Windows routine 'Writefile'.

The program can also display any received serial data and the state of all input pins. This is accessed by pressing the 'Update input' button. The routine used for this is 'UpdateClick'. First of all the 'GetModemStatus' Windows routine is called. This routine sets a variable (called 'State' in our example) with the current state of the input pins. The next few lines update the screen with the state just read.

The second part of this routine takes care of any characters that may have been received. First of all the routine 'ClearCommError' is called. This routine does more than just clear any possible errors. It also fills a TCOMSTAT record with assorted information. The most important of which is the number of characters still present in the input buffer. If this number is greater than zero, the program calls the Windows routine 'ReadFile'. The received characters are then shown on the screen.

The rest of the program is concerned with providing a nice user interface, allowing the user to change the settings of the serial port, choose a different port, etc.

HTML and I/O

HTML and I/O? Wasn't this article supposed to be about programming? The running of programs is certainly also possible from within HTML pages. Scripting languages have been created for just this purpose. The two best known scripting languages are JavaScript and VBScript. We would recommend that you use JavaScript, since most Internet browsers support this. To start with,

```
                                {use only the constant declared in Windows}
                                {NOPARITY EVENPARITY ODDPARITY! }
    StopBits    : Byte;         {Nr of stopbits StopBits}
                                {use only the constant declared in Windows}
                                {ONESTOPBIT, ONE5STOPBITS or TWOSTOPBITS !}
    ReadBuffer  : string;

  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.FormActivate(Sender: TObject);
begin
  ComPort:=1;                    {Set the startup settings}
  RadioButton1.Checked:=true;
  BaudRate:=9600;
  RadioButton6.Checked:=true;
  Parity:=NOPARITY;
  RadioButton10.Checked:=true;
  StopBits:=ONESTOPBIT;
  Radiobutton13.Checked:=true;

  Open:=false;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  hPort:=CreateFile (PChar('COM'+IntToStr(ComPort)),
    GENERIC_READ or GENERIC_WRITE,0,nil,OPEN_EXISTING,
    FILE_ATTRIBUTE_NORMAL,LongInt(0));
  if (hPort = LongInt(INVALID_HANDLE_VALUE)) then
    MessageDlg ('Error opening port COM'+IntToStr(ComPort)+' : '+
    #13+#10+SysErrorMessage(GetLastError), mtError,[mbOk],0);

  if (hPort  LongInt(INVALID_HANDLE_VALUE)) then
  begin
    if GetCommState (hPort,dcbCom) then
    begin
      dcbCom.Baudrate:=BaudRate;
      dcbCom.ByteSize:=8;
      dcbCom.Parity:=Byte(Parity);
      dcbCom.Flags:=0;
      SetCommState (hPort, dcbCom);
    end;
  end;

  if (hPort  LongInt(INVALID_HANDLE_VALUE)) then
  begin
    Open:=true;
    CheckBox1Click(Self);
    CheckBox2Click(Self);
    UpdateClick (Self);
    Button1.Enabled:=false;           { Open-button disabled }
    Button2.Enabled:=true;            { Close-button enabled }
  end;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  CloseHandle (hPort);       { Close the FileHandle  }
  Button2.Enabled:=false;    { Close-button disabled }
  Button1.Enabled:=True;     { Open-button enabled   }
end;

procedure TForm1.RadioButtonComPort(Sender: TObject);
begin
  if (Open=true) then Button2Click(Self);  { Close Handle }
  with Sender as TRadioButton do
  begin
    ComPort:=(Sender as TRadioButton).Tag;
  end;
end;

procedure TForm1.RadioButtonBaudRate(Sender: TObject);
begin
  if (Open=true) then Button2Click(Self); {Close handle }
  with Sender as TRadioButton do
  begin
    BaudRate:=(Sender as TRadioButton).Tag;
  end;
end;

procedure TForm1.CheckBox1Click(Sender: TObject);
var command : integer;
begin
    if CheckBox1.Checked=true then
      command:=SETDTR
    else
      command:=CLRDTR;
  if (EscapeCommFunction (hPort,command)=false) then
      MessageDlg ('Error changing signal : '+
      #13+#10+SysErrorMessage(GetLastError), mtError,[mbOk],0);
end;

procedure TForm1.CheckBox2Click(Sender: TObject);
var command : integer;
begin
    if CheckBox2.Checked=true then
      command:=SETRTS
    else
      command:=CLRRTS;
  if (EscapeCommFunction (hPort,command)=false) then
      MessageDlg ('Error changing signal : '+
      #13+#10+SysErrorMessage(GetLastError), mtError,[mbOk],0);
end;

procedure TForm1.UpdateClick(Sender: TObject);
var State : Cardinal;
    State2 : TCOMSTAT;
    Error  : DWord;
    chRead : DWord;
    avail  : DWord;
begin
  ReadBuffer:='';
  if (GetCommModemStatus (hPort,State)=false) then
      MessageDlg ('Error retreiving ModemStatus : '+
      #13+#10+SysErrorMessage(GetLastError), mtError,[mbOk],0)
  else
  begin
    if ( (state and MS_CTS_ON)0) then
      CheckBox3.Checked:=True
    else
      CheckBox3.Checked:=false;
    if ( (state and MS_DSR_ON)0) then
      CheckBox4.Checked:=True
    else
      CheckBox4.Checked:=false;
    if ( (state and MS_RLSD_ON)0) then
      CheckBox5.Checked:=True
    else
      CheckBox5.Checked:=false;
    if ( (state and MS_RING_ON)0) then
      CheckBox6.Checked:=True
    else
      CheckBox6.Checked:=false;
  end;
  ClearCommError (hPort,Error,@State2);
  if (Error 0) then
      MessageDlg ('Error retreiving CommError : '+
      #13+#10+SysErrorMessage(GetLastError), mtError,[mbOk],0);
  Avail:=State2.cbInQue;
```

these scripting languages only had a limited functionality. After the introduction of ActiveX components by Microsoft it became possible to create nicer applications through the use of JavaScript or VBScript in a web page.

A very interesting ActiveX component is the Microsoft 'Communications Control'. This takes care of the control of a serial port. Since Internet Explorer supports ActiveX components, it becomes possible to control the serial ports from a web page!

To use this component on a web page, the following declaration has to be included:

```
<OBJECT
classid=clsid:648A5600-
2C6E-101B-82B6-
000000000014 id=MSComm1>
</OBJECT>
```

This includes the ActiveX component in the web page. You obviously won't be able to see it on the web page because it isn't a graphical component. The properties of this component are set via a PARAM command or directly in the script.

The settings made with PARAM commands become active when the component is placed on the web page by the browser. These PARAM commands aren't compulsory, however. When the properties aren't set by PARAM commands, the default values will apply. The only problem is that it is not always clear what these default properties are.

Using JavaScript we can read and/or change these properties from within a script. Especially programmers with some experience in OOP programming will appreciate these properties. In the HTML application of **Example 1** both these methods are used.

Other programming languages and environments

The Delphi program described in this article could just as well have in written in C++ Builder, Visual Basic, etc. When writing a program in these languages, which uses the serial port, the same functions can be used as for Delphi.

It is remarkable that in Delphi the

Windows API can be used directly, even though it isn't described in the help pages. So even when your (Windows) programming environment doesn't have a description of these routines in its help pages, there is still a good chance that the Windows API can be used directly, as long as the routines' names are known. It is also possible to use the ActiveX component, which was used in the example of the HTML page.

(020001-1)

Useful links:

www.codeguru.com
www.programmersheaven.com
www.microsoft.com

Useful literature:

'Advanced Windows',
by Jeffrey Richter, ISBN 1572315842.

```
  chRead:=0;
  if (avail>20) then avail:=20;
  begin
    setLength (ReadBuffer,avail+1);
    ReadFile (hPort,PChar (ReadBuffer)^,avail,chRead,nil);
    Edit2.Text:=ReadBuffer;
  end;
end;

procedure TForm1.ComboBox1KeyPress(Sender: TObject; var Key: Char);
begin
  Key:=Chr(0);
end;

procedure TForm1.RadioButtonParityClick(Sender: TObject);
begin
  if (Open=true) then Button2Click(Self); {Close handle }
  case (Sender as TRadiobutton).Tag of
  1 : Parity:=NOPARITY;
  2 : Parity:=EVENPARITY;
  3 : Parity:=ODDPARITY;
  else
  end;
end;

procedure TForm1.RadioButtonStopBitsClick(Sender: TObject);
begin
  if (Open=true) then Button2Click(Self); {Close handle }
  case (Sender as TRadioButton).Tag of
  1 : StopBits := ONESTOPBIT;
  2 : StopBits := ONE5STOPBITS;
  3 : StopBits := TWOSTOPBITS;
  end;
end;

procedure TForm1.Button4Click(Sender: TObject);
var Written : DWord;
begin
  Writefile(hPort, PChar (Edit1.Text)^, Length (Edit1.Text), Written, nil);
  if (WrittenLength(Edit1.Text)) then
      MessageDlg ('Error writing : '+#13+#10+SysErrorMessage(GetLastError),
      mtError,[mbOk],0);
end;

procedure TForm1.Edit2KeyPress(Sender: TObject; var Key: Char);
begin
  Key:=#0;
end;

end.
```

Multi-Standard Infrared Receiver

compatible with (almost) all remote control transmitters

This circuit complements the remote controls with which practically all consumer electronics devices are equipped: a receiver which can operate with a wide range of infrared transmitters.

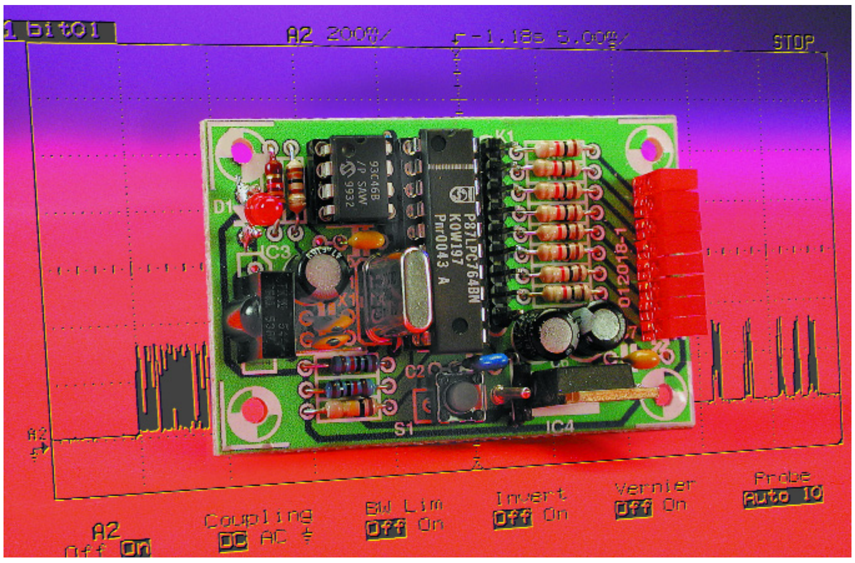
Given the information on remote controls that appeared in the March and April 2001 issues of *Elektor Electronics*, the final remaining step is to build a genuinely universal multi-standard infrared receiver. Using this, a remote controlled system can be constructed using practically any readily-available transmitter, and remote controls that would otherwise be scrapped can be given a new lease of life. Also, the infrequently-used buttons (such as the automatic programming buttons on a TV remote control), can have a new function and be used for example to control the room lighting. Unfortunately, remote controls that transmit using 'flash' mode cannot be used, owing to the way the infrared receiver works internally.

The block diagram of the receiver is shown in **Figure 1**. Any desired key on the remote control can be used to operate the circuit: pressing the button causes the selected output to turn on or off. Built into a multi-way extension lead, up to eight channels could be controlled independently.

Model railway enthusiasts will appreciate the ability to easily control lights or other special functions using infrared light, avoiding messy wiring. You might even already have a suitable transmitter. If required, several receivers can be connected in parallel in order to decode all the functions available from a single transmitter. The number of code combinations is practically unlimited, allowing a vast range of remote control applications.

The circuit

The circuit in **Figure 2** is very simple, and should present no constructional difficulties. It is recommended to fit IC1 and IC2 in sockets.



Infrared reception

Receiver IC3 is a highly integrated device that can decode infrared signals with a set carrier frequency. It includes a highly sensitive infrared receiver which operates using a carrier frequency of 36 kHz. A photodiode with daylight filter, amplifier stages, filter and demodulator are all built in, so that no external circuitry worth mentioning is called for. R1 and C1 form a low-pass filter to eliminate supply-borne interference. Several alternatives for this IC are given in the parts list: be careful to note the pinouts!

LED D1 indicates operation of the circuit, flashing rhythmically on reception of data.

Microcontroller

The microcontroller has already been used in several constructional projects in *Elektor Electronics*, such as the Infrared Code Analyser in the October 2001 issue. It has the following characteristics:

- 4 kbytes ROM
- 128 bytes RAM
- 32 bytes customer code EEPROM
- 2.7 V to 6 V supply voltage
- two 16-bit counter/timers
- internal reset
- internal RC oscillator selectable
- 20 mA driver on all port pins
- maximum of 18 I/O pins, when internal reset and RC oscillator are selected.

Technical characteristics

– supply voltage: 5 V
– capable of learning the following infrared codes: 'Japanese', NEC, RC5, RECS80, SIRCS, Denon and Motorola, as well as 'far east' (as used by Daewoo for example)
– 8 freely programmable outputs
– all data stored in EEPROM
– optimised for 36 kHz carrier frequency
– visual programming confirmation
– visual indication of output states

– 2 analogue comparators
– I^2C interface
– full-duplex UART
– serial in-circuit programming

Using a clock frequency of 6 MHz and the internal divider (with a division ration of six) we have a cycle time of 1 ms. This allows accurate measurement of pulse widths, which is essential when deciding between the various codes. The pulse width is measured using one of the two internal timers and compared against preset threshold values.

EEPROM
The serial EEPROM IC2 has a storage capacity of 1024 bits, organised in a 64 by 16 matrix. All read and write operations take place over a Microwire-compatible interface in blocks of 16 bits. Once written, data are retained for at least 40 years, according to the manufacturer.

Switching stage
The microcontroller has drivers capable of delivering up to 20 mA per output pin, when switching to ground. At a high level only about 1 mA can be delivered, and so a transistor is required to switch the load. Two options are shown in the circuit diagram:

Option 1
The port pin drives the output stage via a BUZ11 MOSFET. With its R_{DSON} of 0.04 W, this transistor can easily handle continuous loads of 5 A. The diode protects the transistor against voltage spikes, if for example an inductive load is used. The diode can be dispensed with in the case of a resistive load. This option can only be used if direct currents are to be switched.

Option 2
In this variation a 12 V relay with a coil resistance of 400 W (that is, with a coil current of 30 mA) is switched via a BC548 transistor. The desired load can be switched using a small power relay. The flywheel diode is essential here.

Power supply
Voltage regulator IC4 produces the supply voltage of 5 V, as required by the receiver IC3. The input voltage to the regulator must be at least 9 V, since the regulator drops about 3 V. In applications where a 5 V supply is already available, IC4 can be dispensed with. Observe that cooling is required if the regulator is used at 1 A, but not at 100 mA or below. An input voltage of 12 V is recommended if 12 V relays are to be used.

The software

The software is based on the Infrared Code Analyser, extended so that the received codes are stored and operate the output switches. Rather than simply sampling the signal and storing the samples, the actual code is read: for example the individual pulse lengths are compared against certain given reference values. This operation is rather less memory-intensive, but has the disadvantage that only known formats can be recognised. 'Learning' remote controls generally operate by simply storing samples of the signal and require a large static memory to store the quantity of data.

The software continuously samples the signal (rather more frequently than shown in **Figure 3**) by polling the logical state of input pin P1.4. This occurs at a rate determined by the software cycle time. During the polling process, a software counter counts the number of samples for which the input remains in each logical state. The count is a digital measure of the pulse length. Short interruptions or spikes in the signal can be ignored by only accepting a logic level when it has been stable for a certain minimum number of samples. If this minimum threshold is not reached, interference is assumed and the current counter value is discarded, counting continuing from the previous value.

As with all things, this procedure has its advantages and its disadvantages. On the one hand, it is easy to program, it is resilient (interference between samples is ignored when delay loops are used) and it is portable to other types of microcontroller, since no hardware resources are used. On the other hand, it places a great burden on the processor: the software is exclusively processing the samples and cannot get on with other tasks while a message is being received. Also, interrupts occurring during the measurement can lead to highly inaccurate readings, depending on the time it takes to service them.

The received codes are stored in hexadecimal form in the microcontroller's internal RAM and in the EEPROM. There is no distinction made between address and com-

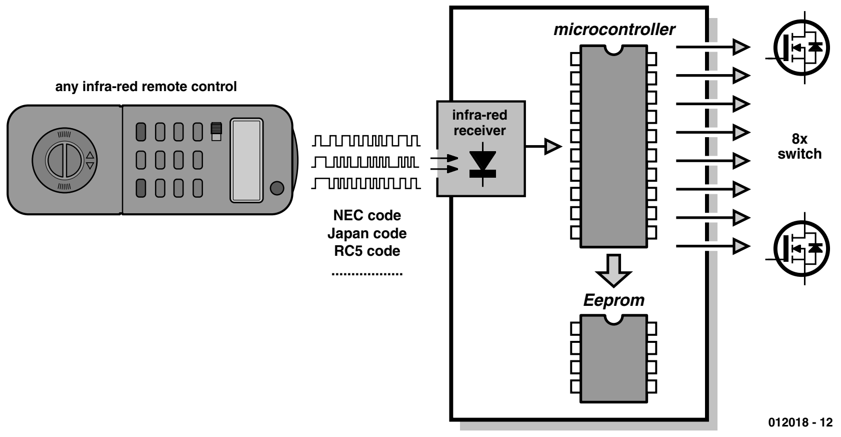


Figure 1. Block diagram of the multi-standard receiver

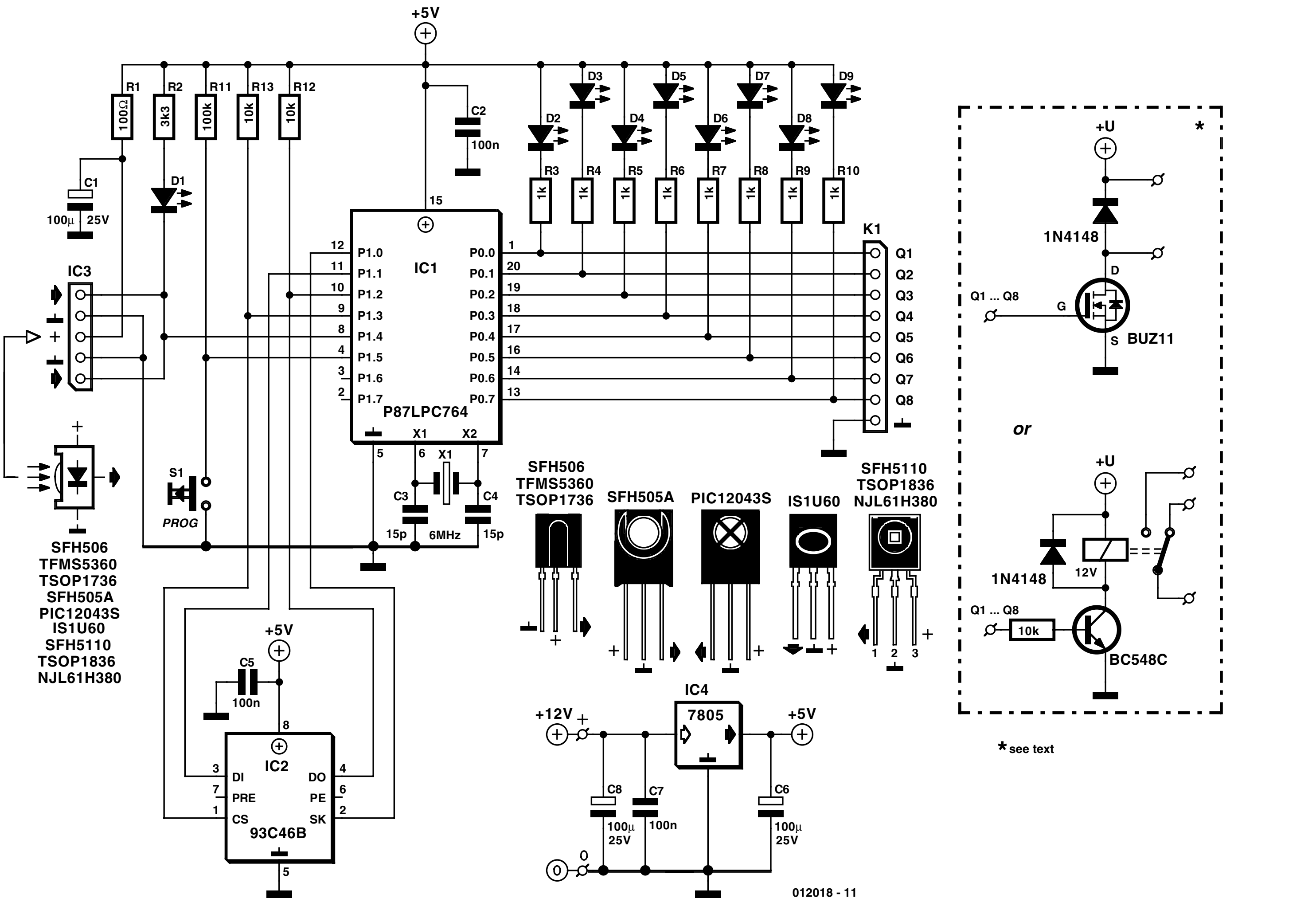


Figure 2. The circuit consists of just an infrared receiver module, a microcontroller, and an EEPROM

mand bits, and so the entire received message is compared with the stored one. 'Toggle' bits, such are used in the RC5 code, which change their state each time a button is pressed, are ignored by the software, since these would otherwise give rise to two different hexadecimal values for the code.

In use

Constructing the receiver circuit on the printed circuit board shown in **Figure 4** should present no problems. The pinouts of several suitable infrared receiver devices are shown.

Before the unit is installed it must be programmed, so that the microcontroller can learn which command is to be used to control which output. Programming is started by briefly pressing button S1. LED D2 immediately begins to flash. Each command now received using one of the codes understood by the microcontroller is stored and assigned to that port pin. This port pin is now programmed in, and the programming process for the next begins automatically: LED D3 starts to flash. This port pin can now be associated with a command in exactly the same way. It therefore does not matter if a different remote control transmitter, employing a different code, is used for each output.

The process continues until all eight outputs have been programmed, after which the LEDs stop flashing. If, at a later date, it is desired to change the command associated with just one channel,

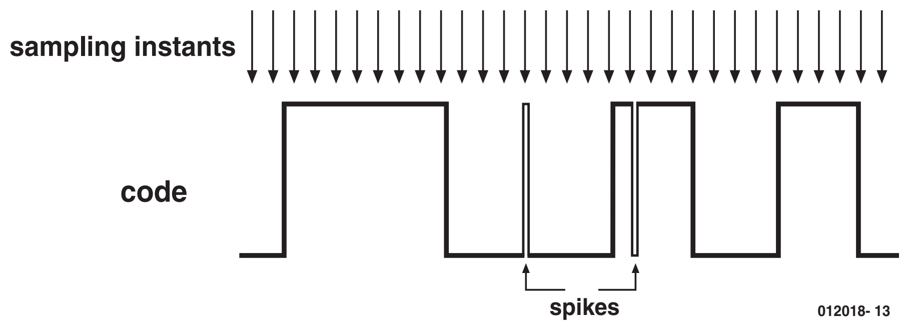


Figure 3. Sampling by polling

COMPONENTS LIST

Resistors:
R1 = 100Ω
R2 = 3kΩ3
R3-R10 = 1kΩ
R11 = 100kΩ
R12,R13 = 10kΩ

Capacitors:
C1,C2,C8 = 100µF 10V
C2,C5,C7 = 100nF ceramic
C3,C4 = 15pF

Semiconductors:
IC1 =P87LPC764BN (Philips), programmed, order code **012018-41**.
IC2 = 93C46 (Microchip 93C46B/P)
IC3 = TSOP1736 (alternatives: SFH5110-36, IS1U60, TFMS5360, PIC26043SM,TSOP1836)
IC4 = 7805
D1 = LED red, high efficiency
D2-D9 = LED, red, rectangular case

Miscellaneous:
X1 = 6 MHz quartz crystal
S1 = pushbutton (make contact)
20-way IC socket
8- way IC socket
K1 = 9-way SIL pinheader
Project software, on disk, order code **012018-11**, or free download from www.elektor-electronics.co.uk

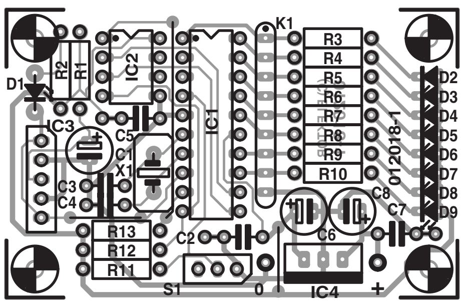
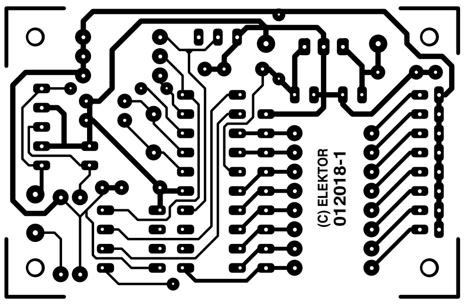



Figure 4. Printed circuit board layout.

the pins for which programming is not required can be skipped with a brief press of the button.

Note that while the LED is flashing, any connected piece of equipment will be switched on and off. All settings are stored permanently in EEPROM IC2, so that when the microcontroller is reset the most recently programmed configuration is still in force.

When a previously-stored code is received, the corresponding output changes state. Since many remote control transmitters send the same message repeatedly when a button is held down, a software monostable is provided to ensure clean operation. Only after a delay of around one second after the last valid message can the state of an output pin be changed again.

Because the software can only interpret the code formats it knows, the user can use the fact that the flashing LED advances as each output is programmed as confirmation that the code has been correctly read. If it is not read, the flashing LED will not advance.

Since we have used an infrared receiver IC3 optimised for a carrier frequency of 36 kHz, we can obtain the greatest range by employing a transmitter that operates on this frequency. If adequate range is not achieved, it is most likely because the transmitter uses a different carrier frequency.

(012018-1)

Internet addresses

EEPROM:
www.fairchildsemi.com/pf/FM/FM93C46.html

Microcontroller:
www.semiconductors.philips.com/pip/p87lpc764bd

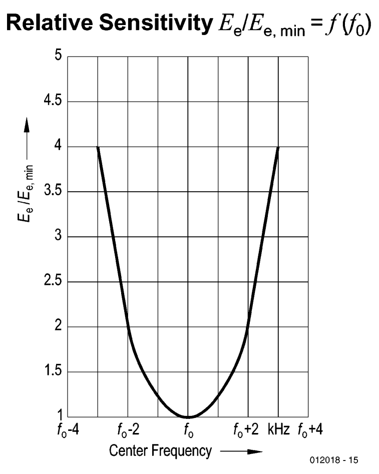
Infrared receiver IC:
www.infineon.com/cmc_upload/0/000/008/562/sfh5110.pdf
www.vishay.com/products/optoelectronics/IRMall.html

Carrier frequency

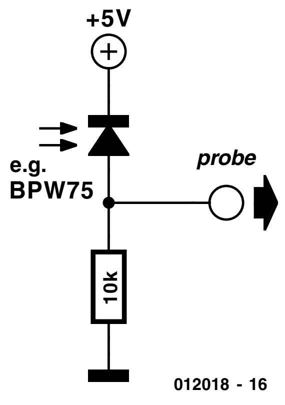
From **Figure A** we can clearly see that even with a transmitter that operates on 34 kHz or 38 kHz, we already have a loss of sensitivity by a factor of 2. If this presents a problem, it may help to use a different infrared receiver. The ICs are available for practically all centre frequencies in 2 kHz steps from 30 kHz to 40 kHz.

A simple way to determine the transmitter carrier frequency is to use an oscilloscope. An ordinary infrared photodiode such as a BPW75 can be connected to 5 V via a resistor as shown in **Figure B**. If a remote control is held close to the photodiode, the signal on the oscilloscope can be analysed and the carrier frequency measured.

A



B

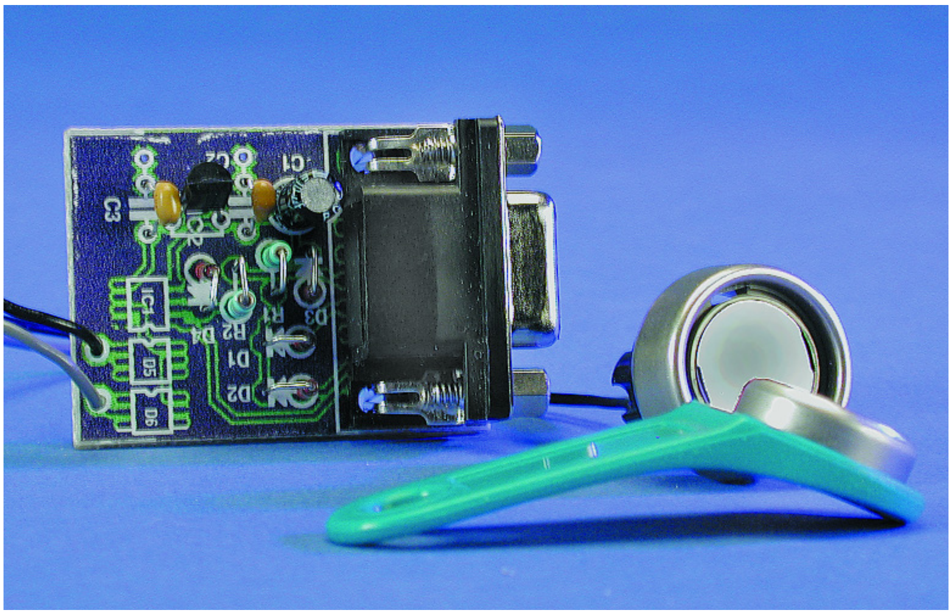


Serial Interface for Dallas 1-Wire Bus

simple to operate from within Windows

By Luc Lemmens

The 1-Wire devices from Dallas Semiconductor are components that via a simple serial connection can be connected to each other, a circuit or a computer. The RS232-interface described here makes the connection to a PC very straightforward. In addition, Dallas supplies the necessary software free of charge.



The 1-Wire series from Dallas already has received attention on earlier occasions in this magazine, such as the E-key project in November 2000 and the 1-Wire Spy in February 2001. In those projects the communication between the 1-wire bus and the PC was controlled by a microcontroller which was specifically programmed for this purpose. Dallas itself also manufactures ICs that deal with interfacing to microcontrollers or PCs. Drivers, evaluation and development software are available for free and may be downloaded from the Dallas web site. This clears most of the obstacles in making a beginning with these devices. This 1-Wire Software Developers Kit (SDK) for Windows Me, 2000, 98, 95, NT 4.00 en NT 3.51 contains programming examples in C, C++, Pascal (Borland Delphi), Microsoft Access and Microsoft Visual Basic. Development tools for 16-bit applications (Windows 3.1 and DOS) are also still available.

New components

The number of different 1-Wire components is quite extensive. In previous issues of *Elektor Electronics* most of the attention was directed towards the so-called iButtons. These are components that are housed in what resembles a sizeable button cell. But there are also components in that range, such as digital potentiometers, temperature sensors, clocks, switches, A/D converters, electronic IDs and memories that are available in a TO-92 package or SMD version. Every device has its own family code (depending on the type of device) and a unique serial number, which enables each device that is connected to the bus to be identified and addressed. **Table 1** shows an overview of the 1-Wire devices that are currently available.

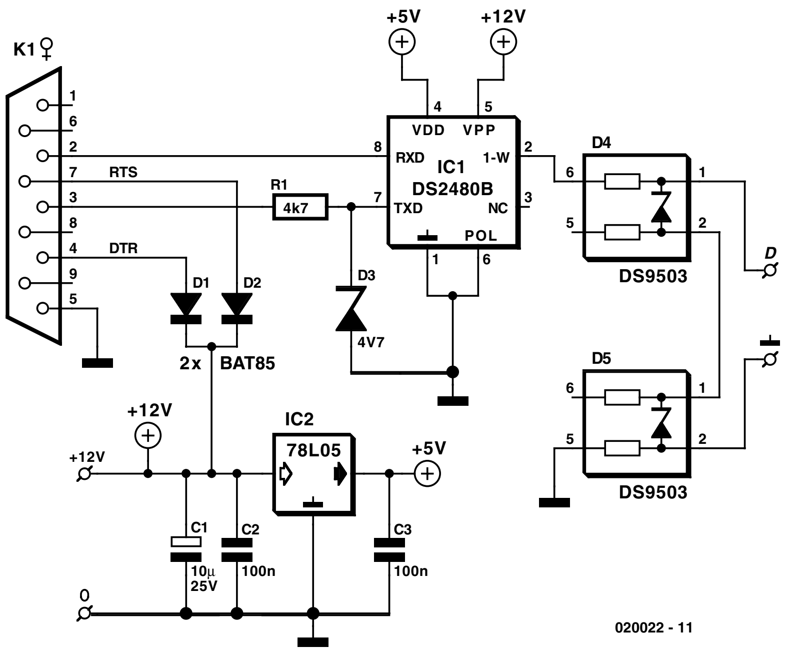


Figure 1. The interface circuit consists almost entirely of two Dallas ICs.

The 1-Wire bus makes it very easy to build a small network that, for example, can measure the temperature in different rooms in a building, or operate switches or light dimmers, etc.

The hardware is very simple: two wires provide power and communications. And Dallas has made available a complete software package on the Internet so that anyone can get started easily. Of course, the 1-Wire bus has to be connected to a PC somehow, but this is not an insurmountable problem, given the interface ICs that have been designed for this bus. In this article we present a serial RS232 interface, a USB interface is to follow soon.

The heart of the interface: the DS2480B

This IC is small with respect to its dimensions (SO-8 package), but large with reference to its capabilities and options. This

1-Wire-Device	Function	Memory
DS1820	Digital Thermometer	16 Bits EEPROM
DS18B20	Programmable Resolution Digital Thermometer	16 Bits EEPROM
DS18S20	High-Precision Digital Thermometer	16 Bits EEPROM
DS1821	Stand-alone Thermostat	2 bytes NV
DS1822	Programmable Resolution Digital Thermometer	No NV
DS2401	Silicon Serial Number	No additional memory
DS2404	EconoRAM Time Chip	4096 bits RAM
DS2404S-C01	Dual-Port Memory Plus Time	4096 bits RAM
DS2405	Addressable Switch	No additional memory
DS2406	Dual Addressable Switch	1024 Bits EPROM
DS2409	MicroLAN Coupler	No additional memory
DS2417	Time Chip with Interrupt	32-bit Real Time Clock Counter
DS2423	1-Wire RAM with Counters	4096 bits RAM
DS2430A	1-Wire EEPROM	256+64 bits EEPROM
DS2433	1-Wire EEPROM	4096 bits EEPROM
DS2434	Thermometer	32 byte EEPROM, 32 bytes SRAM
DS2435	Thermometer/Time-Temperature Histogram	32 bytes EEPROM, 32 bytes SRAM
DS2436	Thermometer, Voltage A/D	32 bytes EEPROM, 8 bytes SRAM
DS2437	Fuel Gauge, Voltage A/D, RealTime Clock, Temperature	40 bytes EEPROM
DS2438	Fuel Gauge, Voltage A/D, Elapsed Time, Temperature	40 bytes EEPROM
DS2450	1-Wire Quad A/D Converter	Status Control Memory Only
DS2480B	1-Wire Line Driver	Status Control Memory Only
DS2490	USB to 1-Wire Bridge	Chip Mode Control and I/O FIFOs
DS2502	Add-only Memory	1024 Bits EPROM
DS2502-UNW	UniqueWare	1024 Bits EPROM
DS2502-E64	IEEE EUI-64 Node Address Chip	256 bits pre-programmed, 768 bits user-programmable
DS2505	Add-only Memory	16,384 Bits EPROM
DS2505-UNW	UniqueWare	16,384 Bits EPROM
DS2506	Add-only Memory	65,536 Bits EPROM
DS2506-UNW	UniqueWare	65,536 Bits EPROM
DS2890	1-Wire Digital Potentiometer	Feature and Control Memory
DS9502	ESD Protection Diode	–
DS9503	ESD Protection Diode with Resistors	–

Table 1. Overview of al 1-Wire devices currently available.

makes it quite a complicated component, but because of the software made available by Dallas we need not concern ourselves too much about the inner workings of this IC. Readers who would like to know exactly what happens inside the DS2480B can study the 30-page datasheet. In this article we make grateful use of the fact that we can treat this IC as a black box. We only need to concern ourselves with the 1-wire devices that are connected to the bus; how they communicate with the PC is of secondary interest.

Hardware

The serial interface between the PC and the 1-Wire bus does not require much, the DS2480B does all the work in co-operation with the driver that is part of the software. The entire schematic is shown in **Figure 1**, and this version is about as complex as possible. For more basic applications a number of parts can even be omitted.

Serial interface

The interface-IC communicates via the RxD and TxD lines of the RS232 port of the PC. As is well-know, according to the standard, a logic one corresponds to –12 V and a logic zero to +12 V, while the serial protocol at the microcontroller is usually +5 V and 0 V respectively. When pin 6 of the IC (POLarity) is connected to ground, the serial signals are inverted inside the DS2480B. The RS232-interface is as simple as passively clamping (R1 and D3) the signal on pin 7 (TxD). The resistor and zener diode ensure that the voltage on this pin is limited to the range of 0 to 4.7 V.

Most (but not all!) modern RS232 ports are quite happy to accept 0 V as a ‘1’ and +5 V as ‘0’. For this reason no additional hardware was added to make this signal conform to the standard (-12 V and +12 V respectively).

1-Wire bus

As the name implies, there is just one ‘wire’ from the DS2480B that deals with the communications on the bus. Most 1-Wire ICs may be powered directly from this bus. In addition, there are devices with a programmable memory (EPROM, Electrically Programmable ROM; note that these are OTP because they do not have windows) that require 12V programming pulses in order to store data. This programming voltage is available at the interface on the VPP-pin (pin 5) of the DS2480B and is automatically applied to the bus when a programming command has been issued.

Dallas Semiconductor explicitly warns users that, when programming, only a single EPROM device must be connected to the interface, with only a short length of wire. In addition, never execute programming commands on the bus if there are devices connected to it that do not contain an EEPROM. This could cause damage to the device or the DS2480B!

D4 and D5 provide ESD-protection (ElectroStatic Discharge) for the interface so that it cannot be damaged by electrostatic discharges from the 1-Wire bus. The DS9503 has a very fast internal diode with a zener voltage of about 7.5 V. The effect of the built-in 5 Ω resistors may be ignored during normal com-

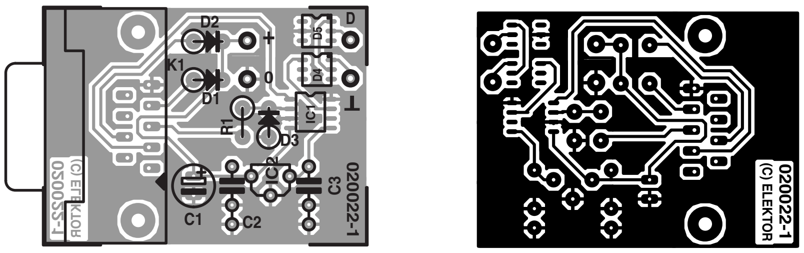


Figure 2. The PCB is not large since the ICs are in SMD packages. Be careful when soldering!

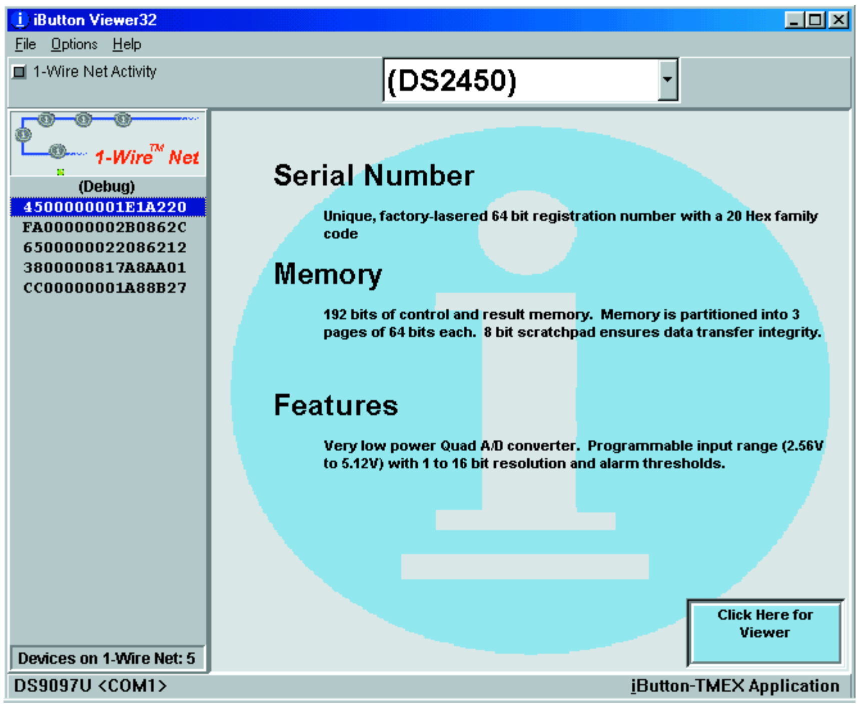
COMPONENTS LIST

Resistors:
R1 = 4kΩ7

Capacitors:
C1 = 10µF 25V radial
C2,C3 = 100nF

Semiconductors:
D1,D2 = BAT85
D3 = zener diode 4.7V 400 mW
D4,D5 = DS9503 (6-pin TSOC)
IC1 = DS2480BS (8-pin SOIC)
IC2 = 78L05

Miscellaneous:
K1 = 9-way sub-D- socket (female), angled pins, PCB mount
PCB, order code **020022-1** (see Readers Services pages)


iButton Viewer32
File Options Help
1-Wire Net Activity
(DS2450)
1-Wire™ Net
(Debug)
4500000001E1A220
FA00000002B0862C
6500000022086212
3800000817A8AA01
CC00000001A88B27

Serial Number
Unique, factory-lasered 64 bit registration number with a 20 Hex family code

Memory
192 bits of control and result memory. Memory is partitioned into 3 pages of 64 bits each. 8 bit scratchpad ensures data transfer integrity.

Features
Very low power Quad A/D converter. Programmable input range (2.56V to 5.12V) with 1 to 16 bit resolution and alarm thresholds.

Click Here for Viewer
Devices on 1-Wire Net: 5
DS9097U <COM1>
iButton-TMEX Application

Figure 3. The main window of the iButton Viewer.

munications but form a high impedance with respect to the zener diode if a discharge occurs on the bus. This ensures that the discharge current flows through the diodes and not pin 2 of the interface IC. If the interface is not going to be used with EPROMs (which require 12 V programming pulses), the normal operating voltage on the bus will not exceed 5 V and D5 can be omitted. Pin 5 of D4 is then directly connected to the ground plane on the PCB and the ground connection of K2 to the anode (pin 2 of D4).

Power supply

The power supply can be obtained from the RS232 port, but that won't be sufficient for the memory devices in the 1-Wire bus that require a 12-V programming voltage. In order to program memories, an external regulated 12-V power supply is connected to K3. Just to be sure: take note of the warnings with respect to the programming of these memories mentioned earlier!

Under normal circumstances, the DTR and RTS can provide the power for the interface. When one of the lines is high (+12 V), there is sufficient current available for IC2 to provide a nicely regulated +5-V supply.

Construction and test

Figure 2 shows the printed circuit board layout that was designed for this interface.

The DS2480B and DS9503 are SMD-components and it is recommended that these parts are fitted first, on the copper side of the PCB. Pay close attention to the marking of pin 1 (a notch). Put a small amount of solder on one corner pad, then place the IC correctly on its footprint and solder this pin first. Proceed to solder the corner pin diagonally opposite. Now carefully check that all the other pins are properly aligned with their corresponding solder pads, before soldering the remaining pins. Excessive solder can be removed with solder wick.

Continue by fitting the resistors and diodes vertically on the component side of the PCB and, last but not least, the voltage regulator IC2, the capacitors and the connectors.

When you are finished, carefully inspect your work. Pay particular attention to possible short-circuits between the pins of the SMD devices.

Before we test the hardware we first download the iButton Viewer and TMEX-drivers from

www.ibutton.com/software/tmex/index.html

Currently version 3.20 is the most recent. The file TM320_32.EXE installs the drivers and the software, and when completed, immediately starts up the iButton Viewer. This program will first go looking for the 1-wire interface.

Connect the circuit with a normal 9-wire serial cable (NOT a null modem cable!) to an available COM port on the PC. The external power supply on K3 is not required and no devices are connected to the 1-Wire bus yet. If everything is correct, our interface will be found after clicking the Auto-Detect button. The setup program will close and the iButton viewer has to be restarted. An error message will appear immediately if the program is unable to communicate with the interface. If this is the case, the fault is almost certainly with the hardware. Once again check your soldering, the orientation of the ICs, electrolytic capacitors and diodes.

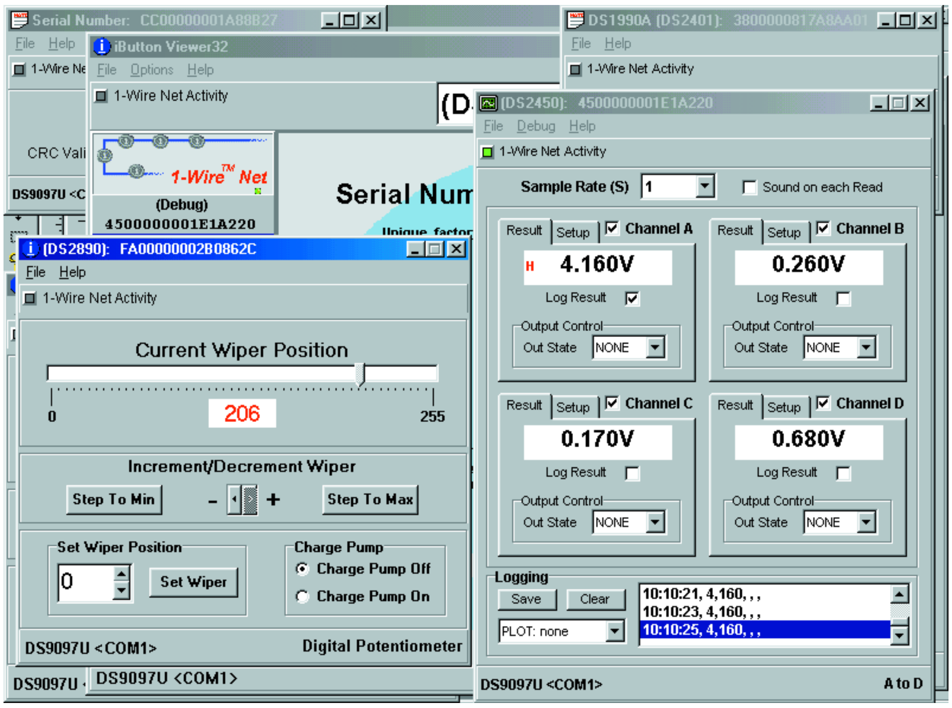
If everything goes as desired, the software will indicate a DS9097U adapter attached to the COM port; this is the interface that is built with a DS2480. At this point the communication between the PC and the interface is working properly. We can now proceed to the 1-wire bus, which is what started all of this in the first place.

Devices on the 1-Wire bus

Now that we have a fully functioning interface it is time to connect one or more devices to the 1-Wire bus. Read the corresponding data sheets thoroughly. Some devices can be directly connected with two wires to K2 of the interface, but others may require external components (a crystal, for example, with clock-IC DS2417) or their own stabilised power supply.

Those of you who don't like programming can get started immediately with the iButton Viewer. This program automatically shows which devices are present on the 1-Wire bus. **Figure 3** shows the main window of the viewer. In the left column are the unique identification numbers of the five devices that have been found on the bus. On the right is a brief description of the device that has been highlighted on the left, in this case a DS2450 (a 4-channel A/D converter).

Clicking the button 'Click Here for Viewer'


Serial Number: CC00000001A88B27
iButton Viewer32
DS1990A (DS2401): 3800000817A8AA01
(DS2450): 4500000001E1A220
(Debug) 4500000001E1A220
(DS2890): FA00000002B0862C
Current Wiper Position: 206 (0 – 255)
Channel A: 4.160V, Log Result ✓, Out State NONE
Channel B: 0.260V, Out State NONE
Channel C: 0.170V, Out State NONE
Channel D: 0.680V, Out State NONE
Sample Rate (S): 1
```
10:10:21, 4,160, , ,
10:10:23, 4,160, , ,
10:10:25, 4,160, , ,
```
DS9097U <COM1> Digital Potentiometer
DS9097U <COM1> A to D

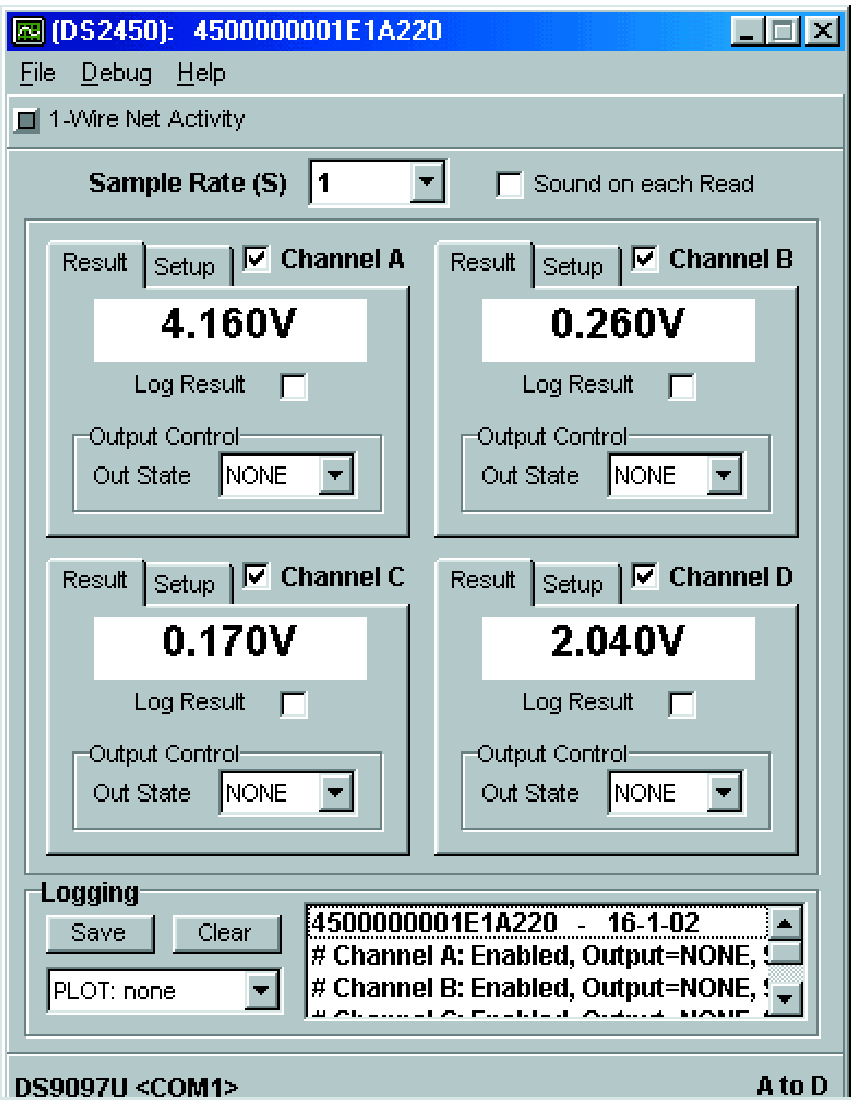
opens a window which displays the four measurement values and various settings for this IC (refer to **Figure 4**). The settings can be altered. For example, the time interval between two measurements can be changed, one or more channels can be turned on or off or the measuring range of a channel can be adjusted, etc.

It is also possible to open multiple views at once, so that several devices are displayed on the screen at the same time. In this way, it is easy to obtain an overview of the network and makes experimenting easy. One way is to interconnect different 1-Wire devices. For example, we can connect a 1-Wire potentiometer to one channel of the A/D-converter and observe the changing measurement result of this channel.

This is all very nice, and very entertaining to play with, and it may even be sufficient for simple applications, but the iButton Viewer does not allow several devices to be interconnected in software. If you would like to do more, such as automatically operate a switch connected to the bus when the measurement result on channel A of the A/D-converter exceeds a certain value, then you will have to program this yourself. The 1-Wire Software Developer's Kit (SDK), available from the Dallas web site, facilitates this.

1-Wire SDK

This software package can be downloaded from the same page as the iButton Viewer. At the moment the most recent version is SDK V4.0 ALPHA. This contains, among other things, the documentation for the TMEX API (application program interface) and descriptions of the functions that can be called from within you own software to communicate with devices that are connected to the 1-Wire bus. It also contains drivers for the serial interface and USB on a PC and programming examples in Pascal, Delphi, C and Visual Basic. There is more than enough information and reading material to get you started with your own programming.


(DS2450): 4500000001E1A220

File Debug Help

1-Wire Net Activity

Sample Rate (S): 1

Sound on each Read

Channel A: 4.160V, Log Result, Output Control, Out State: NONE

Channel B: 0.260V, Log Result, Output Control, Out State: NONE

Channel C: 0.170V, Log Result, Output Control, Out State: NONE

Channel D: 2.040V, Log Result, Output Control, Out State: NONE

Logging

Save Clear

PLOT: none

```
4500000001E1A220 - 16-1-02
# Channel A: Enabled, Output=NONE,
# Channel B: Enabled, Output=NONE,
```

DS9097U <COM1> A to D

Figure 4. This window shows the four measurement results and various settings of the 4-channel A/D converter DS2450.

The 1-Wire bus in practice

Unfortunately, we cannot connect an unlimited number of 1-Wire devices to the interface that has been described in this article. Also, the physical layout of the total network, such as cable lengths and topology of the bus is limited in practice. As long as the interface and the devices connected to it are spread out over a workbench and the bus length is less than 1 meter or so, there will be no problems. But before you randomly start drilling holes and run wires through a house or other building, it is recommended that you first read application note AN148, 'Guidelines for Reliable 1-Wire Networks'. This contains a clear and thorough description of the rules that you have to adhere to, and which steps you can take, to ensure reliable communications across the network. This application note, which also contains additional information about the DS2480B, can be found with the other 1-Wire application notes.

(020022-1)

Internet addresses:

1-Wire SDK:

www.ibutton.com/software/tmex/

1-Wire application notes:

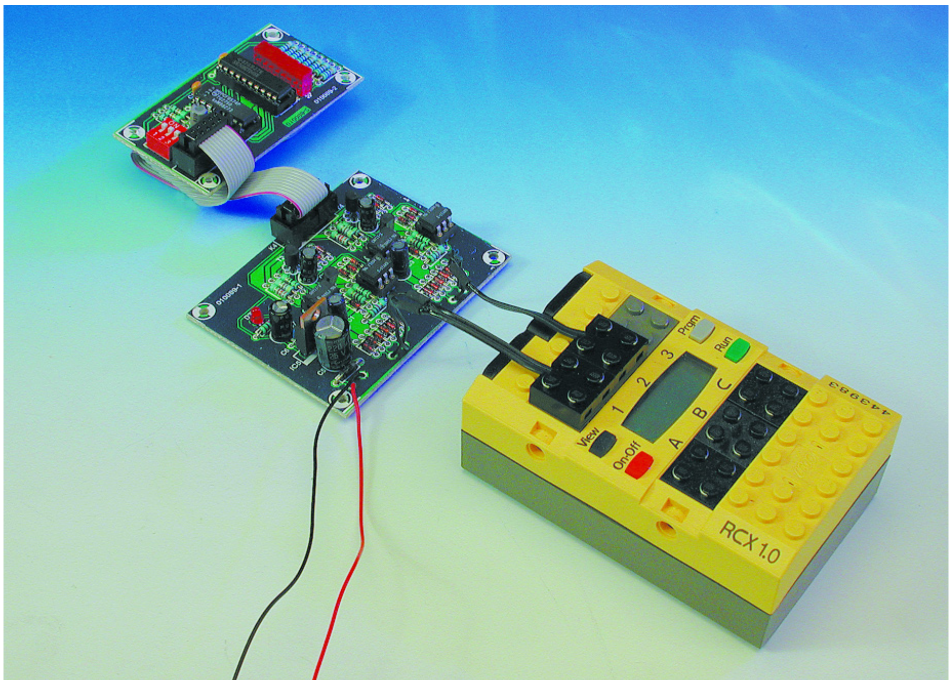
http://dbserv.maxim-ic.com/an_prodline2.cfm?prodline=21

I^{2}C Interface for Lego RCX 'Brick'

makes unprecedented extensions possible!

Design by W. Huiskamp

It has, by now, become well known that the Lego RCX-module is eminently suitable for experimenting with such things as robotics. But once the designs grow beyond simple experimentation it quickly becomes obvious that the number of inputs and outputs that this control module provides, are rather insufficient. That's why *Elektor Electronics* presents the I^{2}C-interface for this Lego brick. An entirely new world is opened up: in principle, no fewer than 128 I^{2}C-devices may be connected to the bus!



The RCX, the 'intelligent' component of the Lego Robotics Invention System, which forms part of the Lego Mindstorms program, is eminently suitable to serve a as the basis of various automation applications. Apart from the programming that is still required, it is very easy to assemble, for example, a small robot together with 'normal' Lego parts. However powerful or modular the design of the RCX-module may be, when more things have to be controlled or measured than are possible with the three inputs and the three outputs, you're out of luck…

This is not the first time that this 'shortcoming' in the design has been noted. In a series of five articles, which commenced in April 2000, we presented the Lego Robotics Invention System. Even then, attempts were made to connect multiple sensors to one input. The Summer Circuits issue of the same year also contained a trick that enabled multiple switches to be connected to one input.

Naturally, the design staff and editors of *Elektor Electronics* where not the only ones to notice that the I/O capabilities of the RCX module are rather limited. Various enthusiasts have invented circuits that,

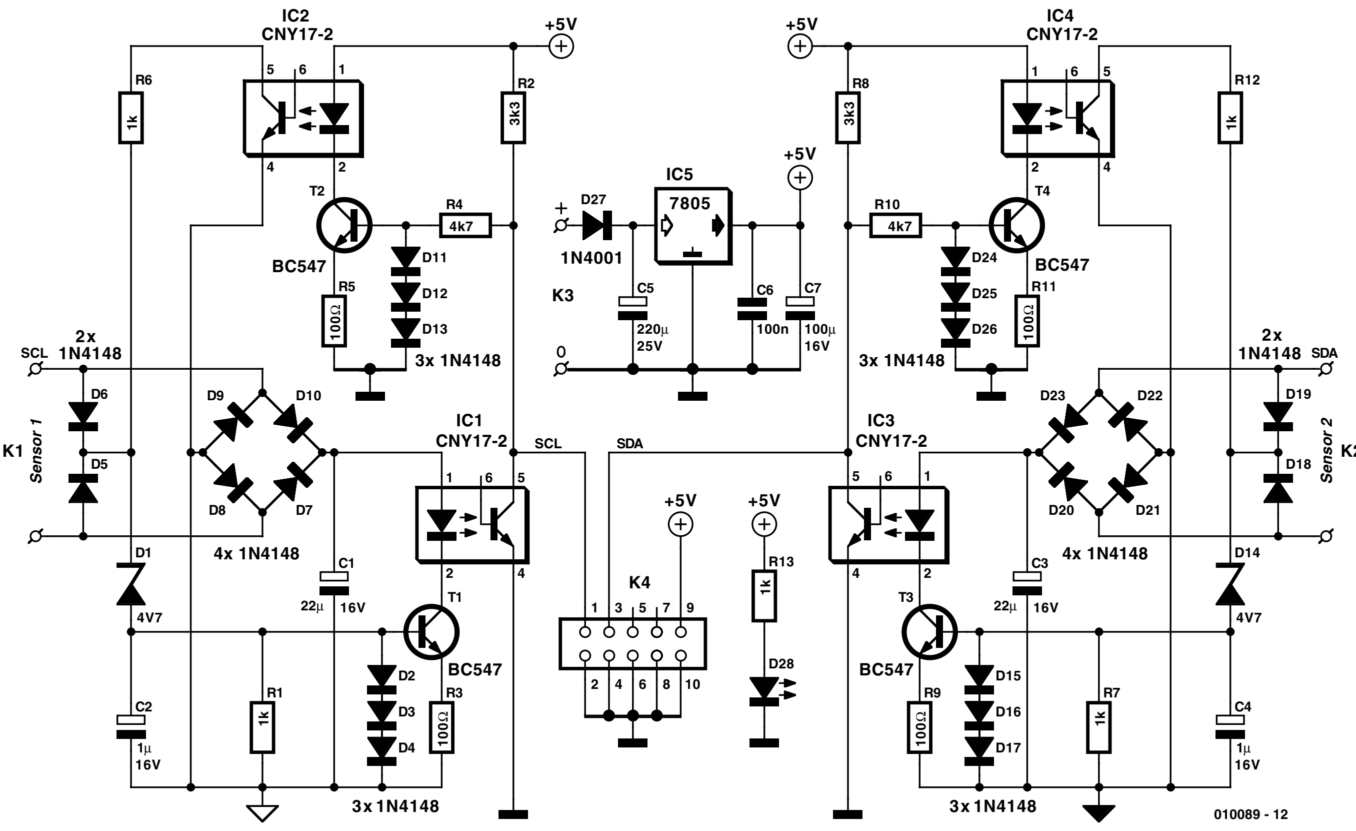


Figure 1. The I²C-interface for the Lego-RCX consists of two identical parts for the SCL- en SDA-signal.

using multiplexing techniques, enable more peripherals to be connected. Some of these are very ingenious indeed, and will certainly be sufficient in certain circumstances. The circuit presented here, however, is probably a fraction smarter still. Because, with this extension, the RCX module can communicate with (theoretically) up to 128 devices on the I²C bus!

Operation of the interface

The Inter-IC bus (I²C) is a 2-wire bus with a bi-directional data line (SDA) and a clock signal (SCL). The bus is subject to a number of rules. Data transfer takes place between the so-called master and one or more slaves. The master is the device that initiates the communications and also generates the clock signal. In addition, the devices are not permitted to pull the connections to the bus active high. A high level is obtained by making that connection high impedance, a resistor with a value of 3k3, connected to the power supply rail, providing the logic High level on the bus. Pulling the Low to produce a zero is permitted. The 3k3-resistor is then pulled to ground.

The RCX module has to be able to generate both ones and zeros on the bus, but also has to be able to read ones and zeros. This can be realised with a single sensor input by utilising a well-known trick. Under software control, a sensor input can be configured as either **active** or **passive**. In passive mode, the measured value (= the voltage at the input) is determined by the internal resistor and the resistance of the attached sensor. The voltage of this potential divider can vary between 0 and 5 V. When the sensor input is set to active mode however, the battery voltage (7 to 9 V) is applied to the sensor connections between measurements. This voltage is periodically, approximately every 3 ms, switched off and the value at the input is determined in the usual way. A measurement takes about 0.1 ms to carry out.

In this way, by switching between active and passive sensors under software control, it becomes possible to generate a signal from the RCX to the I²C bus. In addition, it remains possible to read the signals originating on the bus via the same sensor input.

Circuit

The actual interface for both the SDA and SCL signals is identical. Consequently, the same circuit can be recognised twice in **Figure 1**. We will only discuss the operation for one signal. The circuit is split into two parts because of the optocoupler. This prevents damage to either the RCX module or the attached devices in the event the voltage is too high somewhere (for example, when only one of the two power supplies is connected). This is because the section of the circuit at the I²C side is powered separately. IC5 takes care of this. D27 protects against damage from reverse polarity if the power supply is connected incorrectly. LED D28 is the power supply indicator.

The RCX part of the circuit is powered via D7 to D10. The diodes are connected as a bridge rectifier, so there is no need to consider the polarity when connecting to the RCX module. C1 buffers the power supply during those periods when the measurement takes place. This is necessary because in the

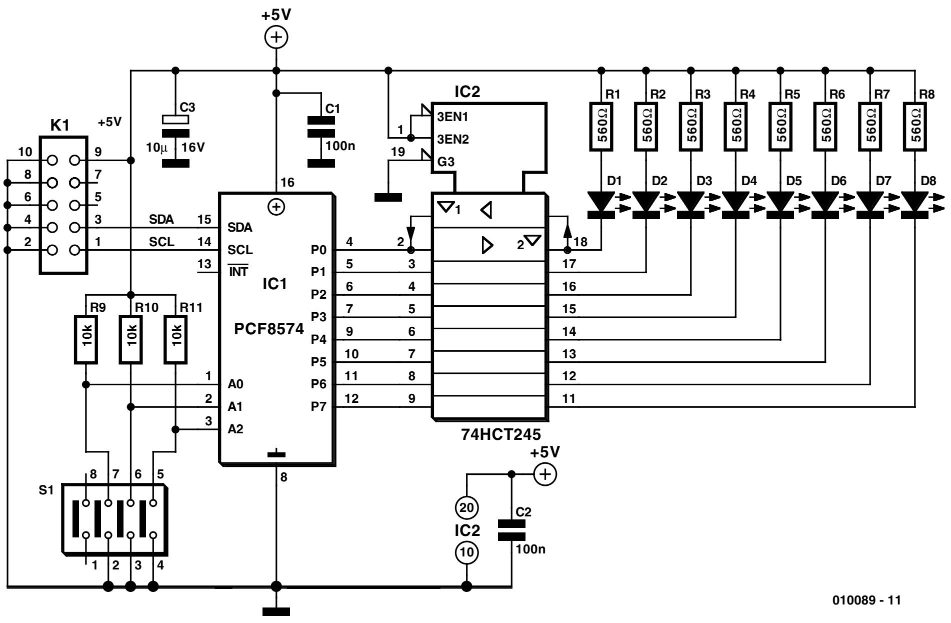


Figure 2. Test circuit with I^{2}C port-expander and driver-IC with eight LEDs.

active mode the battery is periodically disconnected.

The voltage at the sensor connection is also applied to the circuit via D5 and D6 (together they ensure that the polarity is immaterial). The combination of zener D1, and D2, D3 and D4 make it possible to make a distinction between passive and active mode. The zener does not conduct in passive mode. Once the battery voltage is switched on, C2 will charge and cause a voltage drop of up to 1.8 V across D2, D3 and D4. In this case C2 also takes care of riding through the periodic measurements. When this 1.8 V is present, transistor T1 will conduct and a current will flow through the LED in the opto-coupler. R1 makes sure that the capacitor is discharged quickly enough, when the input is switched back to passive mode. This method, with separate power supply and switching circuit, provides an accurately defined ‘1’ or ‘0’ on the bus.

The circuit in the other side of IC1 is very simple and actually consists only of R2. This resistor provides the high-level signal when the line is at high impedance. When the LED is on, the transistor part of the opto-coupler conducts and the output level on the bus is low. The circuit, therefore, acts as an inverter. This is something we have to take into account in the software.

The purpose of R4, D11, D12, D13, T2 and R5, together with IC2 and R6 is to be able to read the signal originating on the bus at the RCX-module. The logic level on the SCL or SDA line is applied via R4 to the base of T2 and the diode string D11, D12 and D13. The diodes limit the voltage at the base of T2 to 1.8 V. When the level on the bus is high, T2 will conduct and the LED in opto-coupler IC2 will be on. The LED obtains its power from the separate power supply for the I^{2}C section.

On the RCX side of IC2, the transistor in the optocoupler ensures that, via R6, the input voltage at the sensor input will be low when the LED is on. D5 and D6 ensure that, once again, the polarity is not important when making the connection.

Your first device

In order to be able to test the interface, it is necessary of course, to connect a device to the bus. We summon the circuit of **Figure 2** for this purpose. At the centre of the circuit is IC1, a so-called I^{2}C port expander. This ‘device’ is programmed via the bus with an 8-bit value that is presented in binary value at the outputs. If you send the number 1, for example, then P0 will go High; the value 17 will cause P4 and P0 to be High. The outputs of IC1 are connected via IC2, an octal driver IC, to eight LEDs. In an actual application these may of course be any other arbitrary indicators or actuators, such as buzzers or (via buffer stages) relays. Also note that the port expander can read an 8-bit value via P0 to P7 as well. For this purpose, the master first has to set all outputs high, individual connections can then be pulled low. A read command will provide the value for each pin. Up to eight different port-expanders may be connected to the bus. This is because the address for the IC has the form 0100xxx0, where xxx may be set with DIP switch S1.

Building

The construction of either circuit should not cause you any difficulties, particularly if you etch your own circuit boards according to the layouts of **Figures 3** and **4**. We would like to remind you of the things to consider during assembly. Take note of the polarity of diodes, capacitors, transistors and ICs. You may decide to use sockets for the optocouplers. As can be seen from the parts list, a low-current LED was selected for D28 on the interface PCB. This is important because the LED is powered directly from the RCX brick. And every little bit helps, of course, when it comes to saving battery power.

PCB pins and box headers provide the connections to the outside world. Standard Lego-wires can be soldered to K1 and K2. An external 9-V battery eliminator is connected to K3. Box header K4 leads the I^{2}C bus to the outside world. Not only the SCL and SDA signals are available (pins 1 and 3 respectively), but also the power supply (pin 9) and ground (pins 2, 4, 6, 8 and 10). By making the power supply available on this connector, the test PCB can be connected without requiring its own power supply, provided the load is within reason. The 8 LEDs shown here are not a problem. The test PCB is fitted with the same 10-way connector and can be connected with a short piece of ribbon cable.

Software

The accompanying software is an essential part of the interface, of course. A number of functions are implemented that make data communications with the I^{2}C devices possible at a higher level. These functions have been written in NQC 2.2 (Not Quite C). RCX firmware ver-

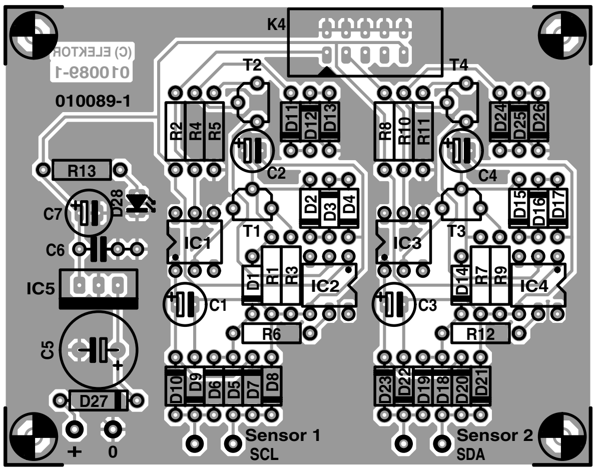
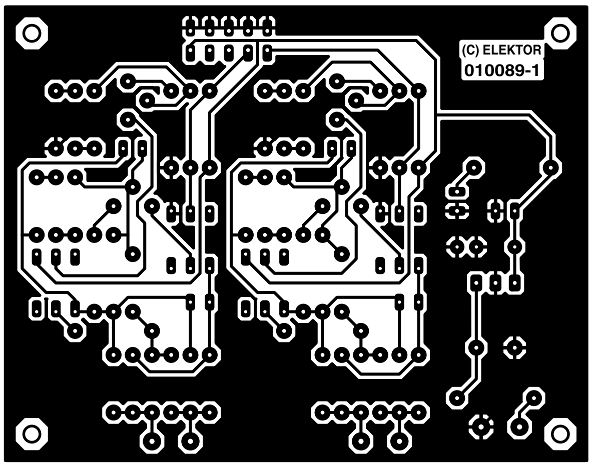



Figure 3. The printed circuit board layout for the I^{2}C-interface.

COMPONENTS LIST
(interface board)

Resistors:
R1,R6,R7,R12,R13 = 1kΩ
R2,R8 =3kΩ3
R3,R5,R9,R11 = 100Ω
R4,R10 = 4kΩ7

Capacitors:
C1,C3 = 22µF 16V radial
C2,C4 = 1µF 16V radial
C5 = 220µF 25V radial
C6 = 100nF
C7 = 1000µF 16V radial

Semiconductors:
D1, D14 = zender diode 4.7V 400mW
D2-D13,D15-D26 = 1N4148
D27 = low-current LED
T1-T4 = BC547
IC1-IC4 =CNY17-2
IC5 = 7805

Miscellaneous:
K1,K2,K3 = 6 PCB solder pins
K4 = 10-way boxheader
Disk, project software, order code 010089-11 (see Readers Services page) or free download from www.elektor-electronics.co.uk

COMPONENTS LIST
(test board)

Resistors:
R1-R8 = 560Ω
R9,R10,R11 = 10kΩ

Capacitors:
C1,C2 = 100nF
C3 = 10µF 16V radial

Semiconductors:
D1-D8 = rectangular LED
IC1 = PCF8574
IC2 = 74HCT245

Miscellaneous:
K1 = 10-way boxheader
S1 = 3-way DIP switch (4-way will also fit)

sion 2.0 has been used to interface with the hardware (previous versions do not work!). If your RCX module is not up to this revision, and you would like to make use of the I^{2}C-functions shown here, you will need to upgrade the module first. This is done as follows: First download the firmware V2.0 (the 'system software') for the RCX module (refer to the link at the end of this article). The new firmware is then downloaded to the RCX. The program BricxCC can be used for this, for example. You first have to set BricxCC to RCX2, nor RCX. The program will ask for this the first time it starts up. Afterwards you can reach this setting by going to Edit → Preferences under the tab Start Up.

Also note that BricxCC is not only used for downloading new firmware, but also provides communication between a PC and the RCX when sending NQC programs. NQC is a compiler, which can translate C-like programs into code for the RCX module.

The functions that provide communications with the bus are best illustrated with the help of two example programs.

First we look at **i2c_test_wr.nqc**, which can be found with the other examples on the floppy disk that belongs with this project

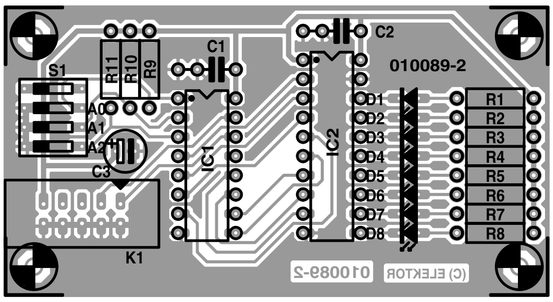
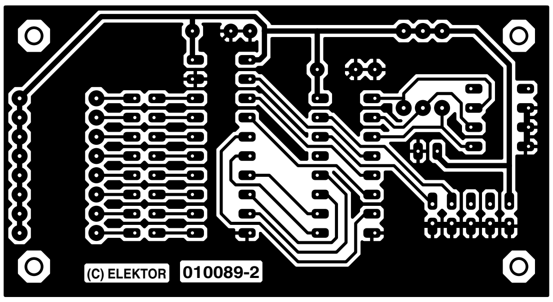



Figure 4. The printed circuit board layout for the test circuit.

(refer to service pages, EPS 010089-11) or on the *Elektor Electronics* web site. The test program increments a counter in steps of one, until a value of 255 is reached. Every new value is sent via the bus to the test PCB. We encounter the following lines of code:

```
i2c_init();
i2c_start();
i2c_data = PCF8574_0;
i2c_send();
i2c_data = test;
i2c_send();
i2c_stop();
```

i2c_start() initiates the I^2C-message. This is followed by sending the address of the device that we're talking to. That is the port expander in this case. After this the actual data is sent, via the variable i2c_data. And finally the communication session is closed.

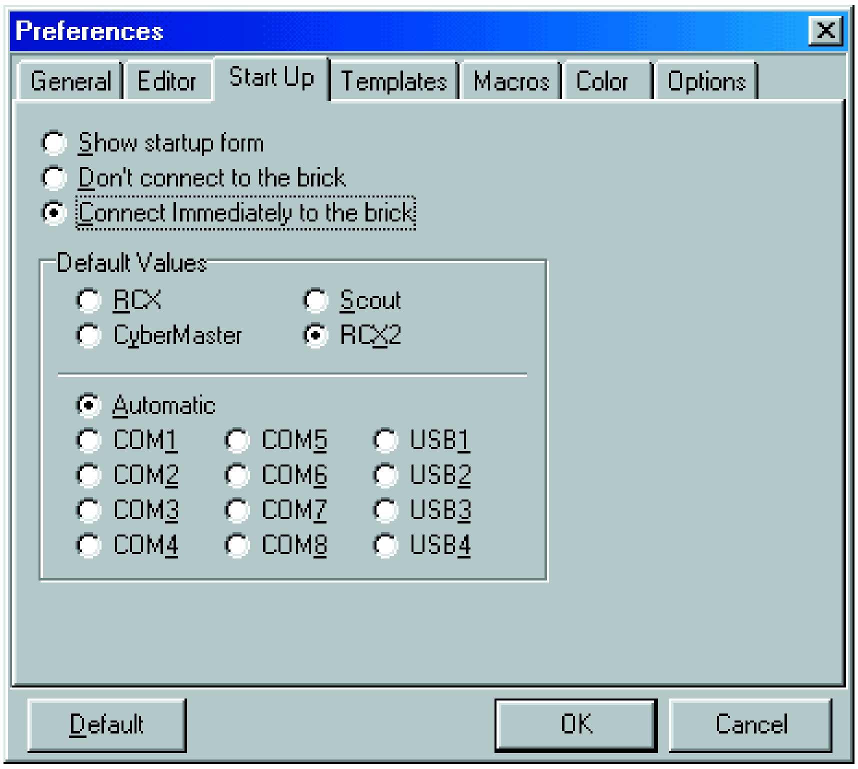
In the example program, this is followed by incrementing the counter and executing the loop once more. This works well, but in principle it is not necessary to stop the communication as long as the same device is addressed. Initialisation, starting, sending of the address and stopping can take place outside the loop in this case. This is certainly a lot faster.

In the second example program, **i2c_test_rd.nqc**, the port-expander is read back. Before this can happen, the communication has to be initialised in the usual manner:

```
i2c_init();
i2c_start();
i2c_data = PCF8574_0;
i2c_send();
```

In order to function as an input, all connections have to be set high. To achieve this, a value of 0xFF (255) is sent to the device:

```
i2c_data = 0xFF;
i2c_send();
```


Preferences

General | Editor | Start Up | Templates | Macros | Color | Options

( ) Show startup form
( ) Don't connect to the brick
(●) Connect Immediately to the brick

Default Values
( ) RCX ( ) Scout
( ) CyberMaster (●) RCX2

(●) Automatic
( ) COM1 ( ) COM5 ( ) USB1
( ) COM2 ( ) COM6 ( ) USB2
( ) COM3 ( ) COM7 ( ) USB3
( ) COM4 ( ) COM8 ( ) USB4

Default | OK | Cancel

Figure 5. After starting BricxCC it is important that you select RCX2.

Now the actual read operation of the port can be carried out:

```
i2c_recv();
test= i2c_data;
```

Finally, the communication is stopped:

```
i2c_stop();
```

The example program is slightly different because it also contains a loop. Also, both programs do not bother with the acknowledge bit i2c_ackn. The master can receive this bit from a slave device, once all the data to the slave has been sent. The reverse is also true, an acknowledge bit can be sent to the slave once all its data has been read. This may be of use in some applications, but it is not necessary here.

It appears that with this software, RCX brick and interface a speed of about 2 complete I^2C messages per second are achievable. This is not particularly fast, and will have to be taken into account for certain applications.

The actual implementation of the user functions is in the file **i2c_master.nqc**. When these functions are called from within a program, this file has to included with the compiler directive #INCLUDE. We consciously avoid a discussion of the low-level functions. Those of you who are nevertheless interested can examine the source code for yourselves. The only thing that we could mention is that the inverting operation of the interface has been compensated for at this level.

In this file can also be indicated which sensor input ultimately represents which signal on the bus. The default is SCL for sensor 1 and SDA for sensor 2. There are also different addresses possible for the PCF8574 (the port expander IC in the test circuit). They are already provided as PCF8574_0 through PCF8574_7 (0x40h through 0x4E).

This definitely does not mean that the software is limited to this device. Other I^2C devices can also be used, such as A/D or D/A converters (PCF8591 for example), LED- and relay-drivers (SAA1064 for example) or even a real-time clock with the Lego RCX module!

(010089-1)

Useful websites

Bricx Command Center 3.3:
http://hometown.aol.com/johnbinder/bricxcc.htm

Not Quite C:
www.cs.uu.nl/people/markov/lego

Official NQC download site:
www.enteract.com/~dbaum/nqc/doc

RCX firmware 2.0:
http://mindstorms.lego.com/sdk2/

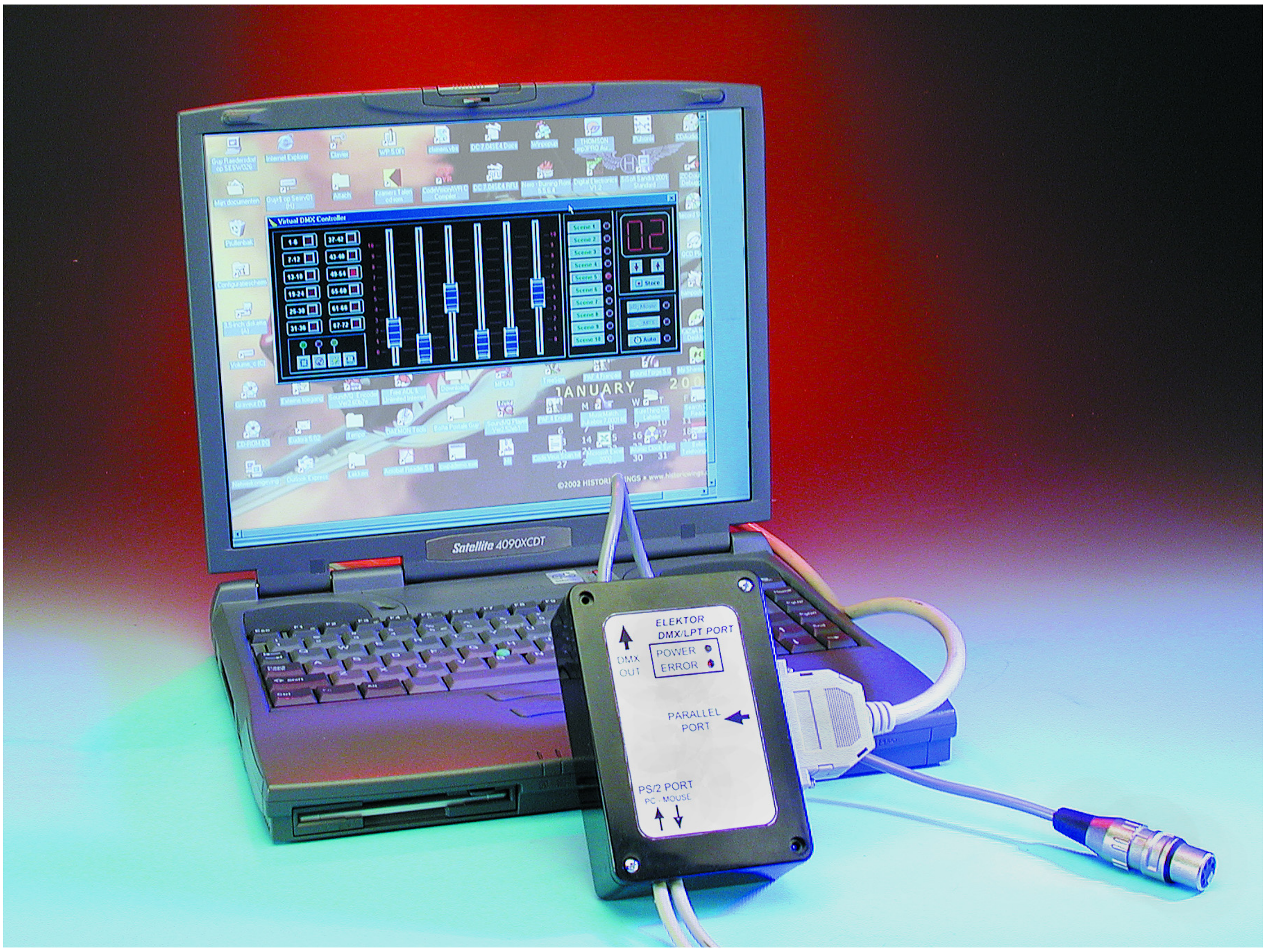
LPT/DMX Interface

480 DMX channels via the parallel port

Design: B. Bouchez

bbouchez@netcourrier.com

The DMX/MIDI interface described in the September 2001 issue is extremely flexible and suitable for nearly all applications, but it is a rather complex circuit. By contrast, the interface described here is the height of simplicity with regard to both construction and programming, even under Windows.



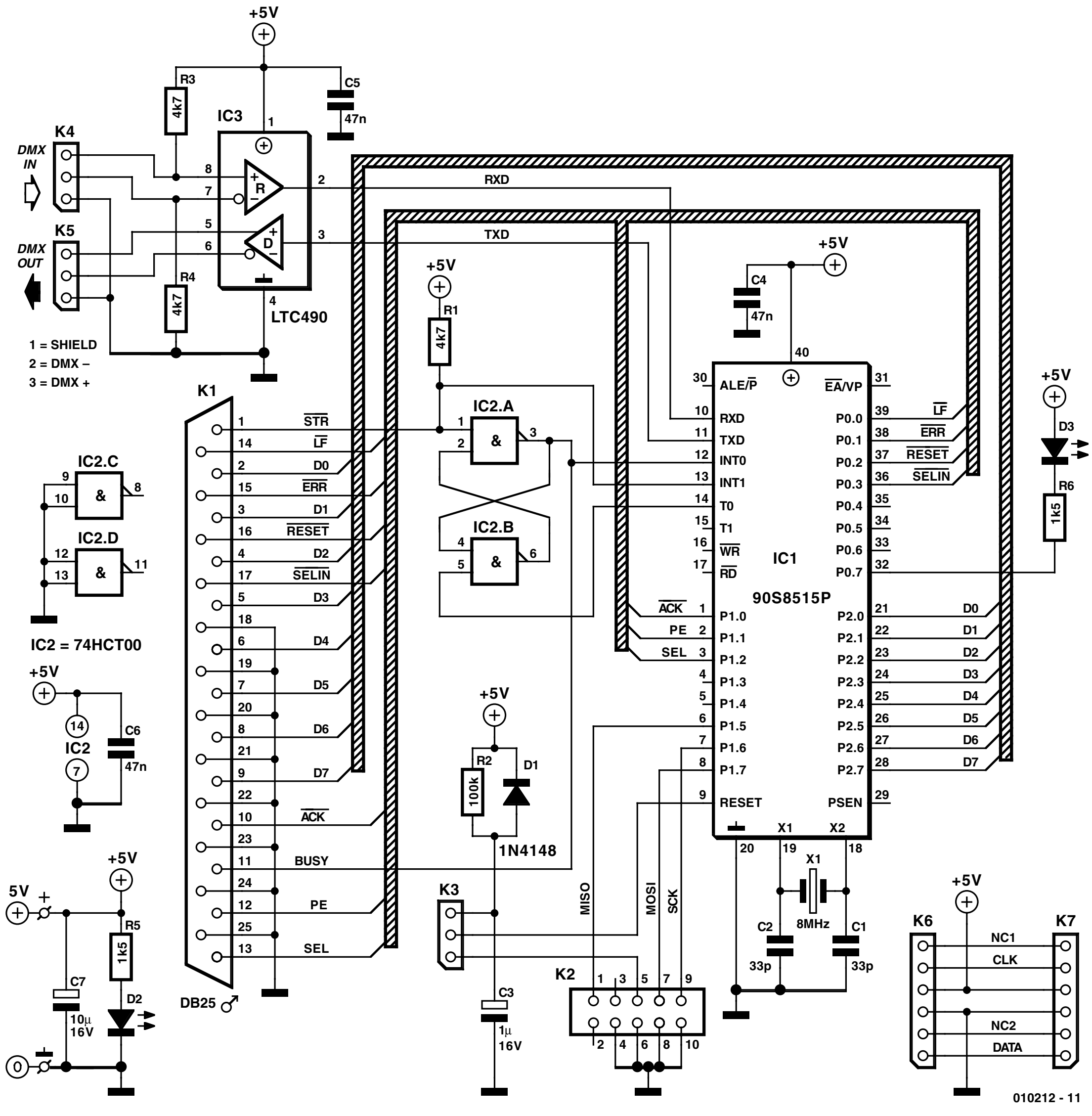


Figure 1. The electronics of the LPT/DMX interface are concentrated in an Atmel AT90S8515 IC.

The DMX protocol has already been the subject of several articles in *Elektor Electronics*. Consequently, here we will describe only the essential points in order to refresh your memory. Readers who are interested in the technical details of the protocol are referred to the article in which it is described ('DMX512 Revealed' in the May 2001 issue) and to various DMX designs that have subsequently appeared in *Elektor Electronics* ('MIDI/DMX Interface' and 'DMX Demultiplexer').

The DMX512 protocol

The protocol was defined in 1986 by the USITT. This American organisation is responsible for the standardisation of all technical matters in the 'show biz' area, such as exchanging digital information between control panels and devices such as dimmers and automated lighting equipment.

The DMX protocol is based on a unidirectional serial link meeting the specifications of the RS485 standard. The commands that are intended to be received by the 'slaves' are sent in the form of a series of bytes, each of which controls one DMX channel. The link works at a speed of 250 kbaud with the format '8 bits, no parity and 2 stop bits'. In order to allow the start of each data frame (which can be as large as 512 bytes plus the start byte) to be determined, communications are interrupted ('Break') for the duration of at least two bits, which allows the receivers to synchronise themselves.

Why this interface?

The DMX signal can be generated by a simple serial port (which, by the way, was also what the developers of this standard had in mind). However, the serial port of a PC is totally unsuitable for this, for two reasons. First, PC ports are designed to meet the RS232 standard (although an RS232/RS485 converter might provide a solution here), and second, the clock rate of serial PC ports makes it impossible to generate a 250-kbaud signal.

To make a long story short, if you want to use DMX in combination with a PC, a separate interface is necessary. The approach that has been taken here is basically very simple.

We just use the parallel port, which is supported by most operating systems (DOS, Windows 3.x/95/98/ME etc. and LINUX). Furthermore, the PC is far from being the only type of computer with a parallel port. It is thus possible to use this interface with machines such as the (former) Atari, Amiga and so on, as long as the proper software is available.

In order to keep this interface as universal as possible and thus avoid limiting its use to modern PCs, it employs only the Centronics protocol instead of any of the PC-specific protocols (Nibble, Reverse, EPP and ECP).

This approach also has a significant advantage when the interface is used with Windows. With this operating system, it is normally necessary to develop drivers in order to be able to use specific interfaces, which is rather complicated. Since this interface uses the Centronics protocol, Windows simply considers it to be a printer and thus installs the standard drivers.

This means that even under Windows, programming the interface amounts to nothing more than 'printing' control commands. This interface can thus be controlled using any desired programming language (Delphi, Visual Basic, Visual C, C++ Builder etc.) that supports printing under Windows.

Furthermore, the designer of the interface has written a special Windows module that allows you to control it without having to know how to program the parallel port under Windows. There is also a special program for testing the interface. Finally, we should note that the interface is supported by the Soft Controller programs (more about this later on).

COMPONENTS LIST

Resistors:
R1,R3,R4 = 4kΩ7
R2 = 100kΩ
R5,R6 = 1kΩ5

Capacitors:
C1, C2 = 33pF
C3 = 1µF 16V radial
C4,C5,C6 = 47nF
C7 = 10µF 16V radial

Semiconductors:
D1 = 1N4148
D2 = LED, red, high efficiency
D3 = LED, yellow, high efficiency
IC1 = AT90S8515-8PC, programmed, order code **010212-41**
IC2 = 74HCT00
IC3 = LTC490 CN8 (Linear Technology)

Miscellaneous:
K1 =DB25 plug (male), PCB mount, angled pins
K2 = 10-way boxheader
K3,K4,K5 = 3-way SIL pinheader
K6 = 6-way mini-DIN socket, PCB mount
K7 = 6-way SIL pinheader
X1 = 8MHz quartz crystal
PCB, order code **010212-1** (see Readers Services page)
Disk, source code and programs, order code **010212-11**

Project software and PCB layout also available as free downloads from www.elektor-electronics.co.uk

The hardware

As can be seen from **Figure 1**, the schematic diagram of the interface is rather simple, since IC1 (an Atmel AT90S8515 RISC microprocessor) looks after practically everything. It contains just about all that we need: RAM, a serial port and EEPROM for the program. A few years ago, around ten chips would have easily been necessary for such a circuit.

The only other semiconductors we need are IC2 (which is used to implement a flip-flop) and IC3 (a Linear Technology LTC490CN8 RS485 transceiver). By the way, the LTC490 may be replaced by its equivalent, the SN75179 (which may be easier to find).

In our first prototypes, the Strobe line of the parallel interface was connected directly to one of the microprocessor interrupt inputs. The width of the pulse on this line varied so much that sometimes things didn't work properly. This is why the flip-flop formed by IC2a/b was added. With this change, even very short pulses are registered.

The pin assignments of K1 match those of the parallel port connector on the PC. The interface can thus be connected to the Centronics connector using a standard cable.

IC3 is a RS485 transceiver. It converts the TTL signals from the microprocessor's serial port to the RS485 signals required by the DMX standard. To keep things simple, we have not provided electrical isolation between the microprocessor and the transceiver. If you are worried about excessive voltages due to earth loops, you should fit Transil protectors between the DMX+ and DMX– lines on K4 and K5 (pins 2 and 3).

You may have already noticed that the DMX interface has a DMX IN input, but this input is not used! It has been included for possible future extensions.

LED D2 indicates whether the interface has power. The job of LED D3, which is driven by the microprocessor, is to report any errors that may occur in the commands send by the computer.

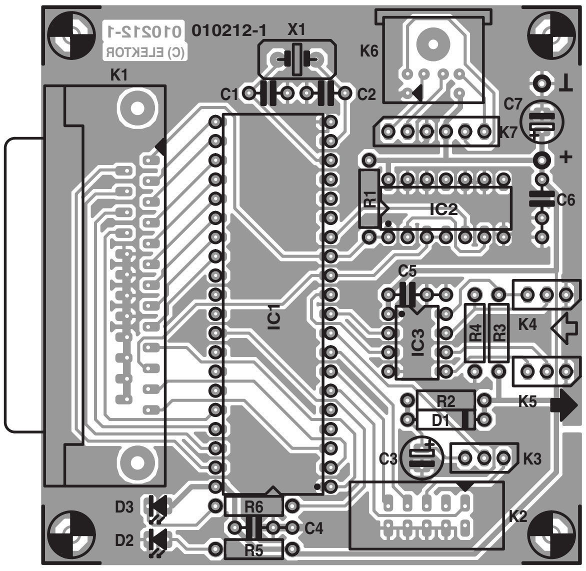
Since this interface can be powered from the PC to which it is connected, the values of R4 and R5 are chosen for use with low-power, high-efficiency LEDs. If you decide to use standard LEDs, a value of 220 or 330 Ω is suitable for these resistors.

Thanks to its low current consumption (less than 10 mA), the interface can be powered directly from the PC to which it is connected. A voltage regulator is thus not necessary.

To obtain power from the PC, we make use of the connection from the PC to the keyboard, which is powered from the 5-V supply. Connectors K6 and K7 are connected to a simple keyboard cable (PS/2 or AT), and the necessary voltage can be tapped off from this cable.

Connector K2 is only used for programming the microcontroller. The Atmel programming cable can be fitted directly to this connector (see the Atmel site at www.atmel.com for the schematic diagram). If you order a pre-programmed microprocessor from Readers Services (order number **010212-41**), K2 may be omitted.

Finally, the triplet R2/C3/D1 generates the reset pulse for the microprocessor. This pulse runs via K3, where you can select the source for the reset (connector K2 or the reset circuit). It should be clear that the K3 jumper should not be fitted if you want to program the microprocessor yourself. During normal operation, the jumper must be present on K3 to provide a link to R2/C3. If you purchase a pre-programmed microprocessor, you can simply replace K3 with a wire bridge fitted between the connections for pins 2 and 3.



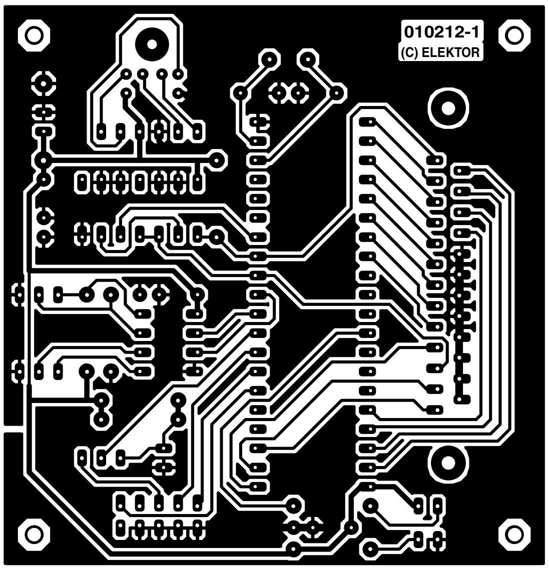
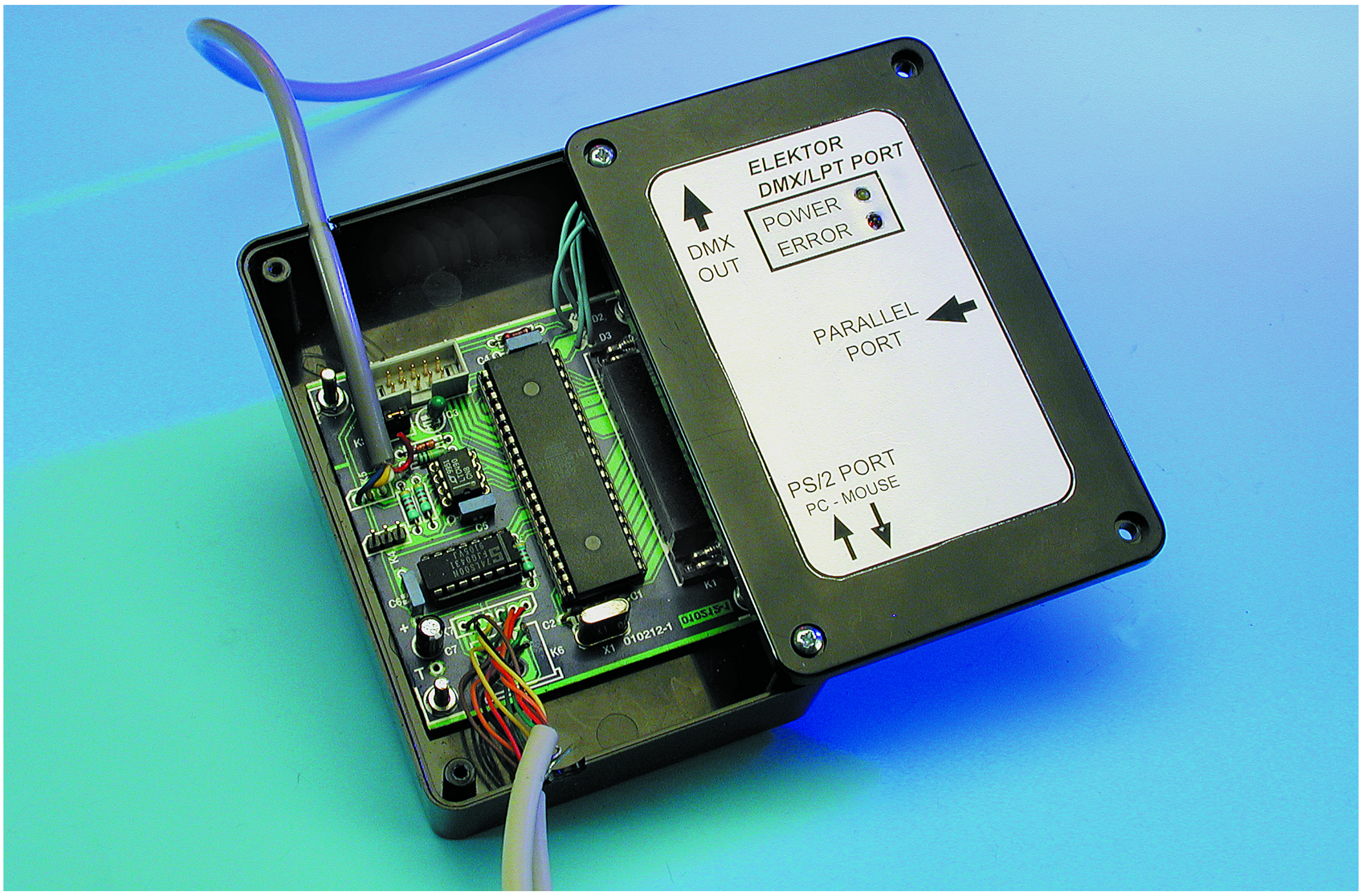


Figure 2. The design of the single-sided printed circuit board is quite simple, so there's not much that can go wrong during assembly.

Here comes the soldering iron

Thanks to the small number of components, assembling the interface is relatively simple. **Figure 2** shows the track and component layouts for the circuit board. In spite of its small dimensions, this is still a single-sided circuit board (the author has made a circuit board that was only half as large by using the PLCC version of the Atmel microprocessor). Use good-quality sockets for the three ICs. If you want to connect K1 of the interface directly to the DB25 connector on a PC, be sure to use a connector without integrated securing screws for K1, since otherwise the connector cannot be



plugged into the PC.

As far as the power supply is concerned, there are various options. The simplest solution is to draw power from the mouse or keyboard cable present on every PC. You can also power the interface from an external supply with a 5-V regulator, but that is more expensive.

If the interface is to be powered from the mouse or keyboard cable, the simplest solution is to take a DIN/DIN extension cable and cut it in half. The wires from the two halves of the cable can then be soldered to the proper pins of connectors K6 and K7. **Figure 3** shows the pin assignments of the PS/2 and DMX cable and chassis connectors.

On the circuit board, K6 is implemented as a PS/2 bus connector. If you fit such a connector to the circuit board, all you have to do is to solder a length of cable with a PS/2 or DIN plug to K7. The original mouse or keyboard plug can then be inserted into the PS/2 bus connector on the circuit board.

Table 1: Interface commands.

Command 'D'	Change a group of values for a series of DMX channels.
syntax:	ESC D hi lo nn dd dd dd...
parameters:	hi: high byte of the first DMX channel to be changed
	lo: low byte of the first DMX channel to be changed
	nn: number of DMX channels to be changed (1–255)
	dd: DMX value for the first channel to be changed
	dd: DMX value for the second channel to be changed
	dd: DMX value for the third channel to be changed
	etc. …
	The number of values (dd) must match the parameter nn.
Command 'd'	Change the value of one DMX channel (number 1–256).
syntax:	ESC d nn dd
parameters:	nn: DMX channel number (0–255 for channels 1–256)
	dd: the value to be sent
Command 'e'	Change the value of one DMX channel (number 257–480).
syntax:	ESC e nn dd
parameters:	nn: DMX channel number (0–224 for channels 257–480)
	dd: the value to be sent
Command 'F'	Set the standard parameters (Start Code = 0 and Break Time = 100 µs).
syntax:	ESC F
Command 'I'	Initialise the interface (also occurs when the power is switched on).
syntax:	ESC I
Command 'S'	Change the Start Code
syntax:	ESC S ss
Command 'Z'	Completely reset of the DMX memory (all channels set to 0).
syntax:	ESC Z
Command 'T'	Change the Break Time setting.
syntax:	ESC T tt
	tt: Break Time setting (in steps of approximately 50 µs; standard value is '2' for 100 µs)

The enclosure

The final subject is fitting the circuit into an enclosure. Given the small dimensions of the circuit board, that should not present any difficulties.

You can fit the circuit board into a separate enclosure that is connected to the parallel port of the PC by a suitable cable (see the photo at the head of the article), or you can fit the circuit board into a small enclosure that can be attached directly to the parallel port connector of the PC.

Bear in mind that the latter solution is not always the most convenient, due to the size of the circuit board. With an office computer, there is usually enough free room at the back, but with a laptop the other connectors are usually so close to the printer connector that your little DMX box will make it more or less impossible to use the other connectors.

The circuit board is not particularly sensitive to external noise, but it is still recommended to use a screened plastic enclosure (i.e., one with an internal graphite coating connected to ground).

The DMX output is implemented in an especially simple manner, namely using a short length of cable fitted with a 3-pin or 5-pin XLR connector. This cable is soldered to K5.

Centronics language

As soon as the circuit is connected, you can start testing and programming.

Thanks to the simple design, it is most likely that everything will work properly from the start. However, a special program has been written to make it easier to test the circuit board. This program is very easy to use, but nevertheless we have summarised the various commands in a separate file on the floppy disk.

Besides providing special functions for testing the interface (such as switching on Error LED D3), this program is also suitable for testing spotlights.

It is obviously our intention to have this interface be used with programs other than just this test aid. If you want to make things easy, you can use the Soft Controller I and II programs developed by the author.

These have already been briefly discussed in the article describing the MIDI/DMX interface, with which they are also compatible. For more information, you can send an e-mail to the author. Still, as already mentioned at the beginning of this article, the main advantage of this circuit is that it can be easily programmed, even under Windows.

While most commercially available DMX interfaces for PCs cannot be used without a lot of control programs and drivers, our interface uses a protocol that is available on every PC: the Centronics protocol.

The interface acts like a printer by emulating the appropriate commands. This means that in order to have the interface execute a command, all you have to do is to instruct the PC to print a series of characters. Printing functionality is included in various flavours of programming languages, such as Delphi, C++ Builder, Visual Basic and the like, which means that the biggest problem is already solved.

If you program under DOS, you can simply use the Basic LPRINT instructions to communicate with the interface.

As already noted, this interface is not limited to use with PCs. Any device with a Centronics port is by definition compatible with the interface.

With regards to programming, here we limit ourselves to describing the method that must be used to send commands to the interface. For readers who program under Windows, we have developed a small DLL library that can be found on the floppy disk (as well as on our Internet site for free downloading). This means that you do not have to trouble yourself with learning how to use the Windows API (Application Programming Interface).

Programming the interface is actually very simple. The interface recognises a certain number of commands, each of which consists of a letter that may be followed one or more parameters, depending on the particular command. To allow the interface to recognise the start of a command, the ESC code (‘27’ in ASCII or ‘1B’ in hexadecimal) is used as an identification code.

As an example, let’s see which bytes you have to send to the ‘printer’ to set DMX channel 18 to a value of 155. For this purpose we use the command ‘d’, followed by the channel number (which can range from 1 to 256, but here we use 0 to 255 to allow it to fit into a single byte), and then followed by the DMX value to be sent.

The bytes to be sent to the parallel port thus appear as follows (the hexadecimal values are shown in parentheses):

```
ESC  (1Bh)
d (64h)
17(11h)
155  (9Bh)
```

Another example: to reinitialise the interface, all you have to do is send the command ‘Z’. The byte sequence is thus:

```
ESC  (1Bh)
Z (5Ah)
```

Knowing this, all you have to do to create your own ‘program of the century’ is to write your own series of bytes and send it to the interface.

Table 1 summarises all the commands recognised by the interface.

(010202-1)

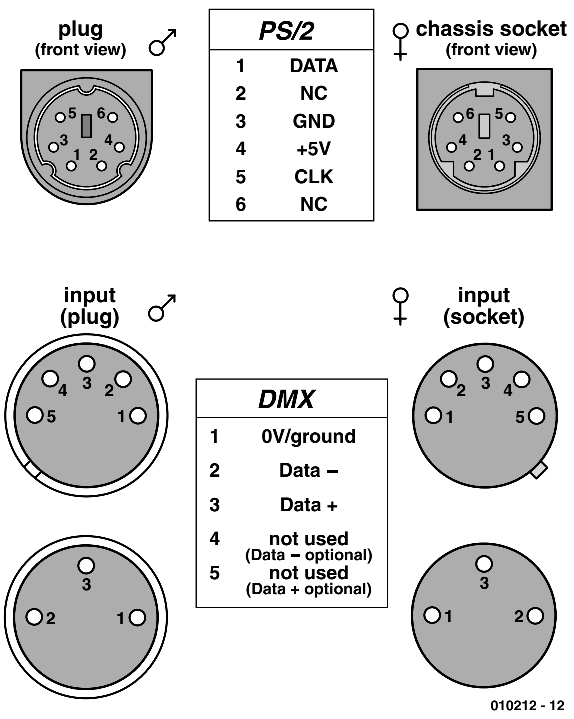


Figure 3. Using this connector information, you can ensure that the interconnecting cable is properly attached to the circuit board.

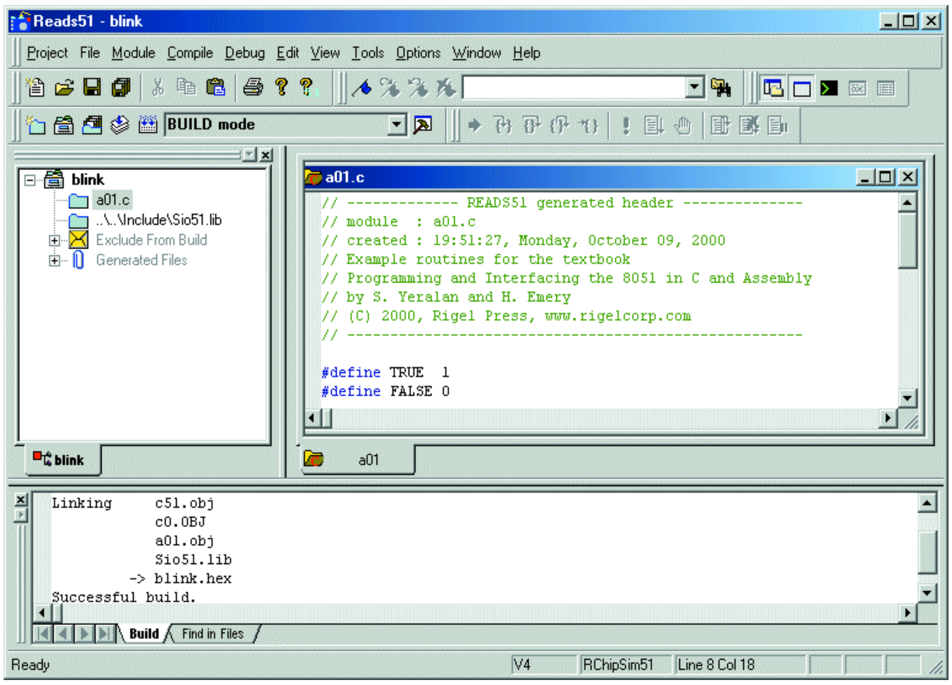
Disk contents (010212-11)

8515def.inc	file containing the definitions of the 8515 registers
defio.inc	file containing the descriptions of the input and output pins
Lptdmx.asm	assembler file for the LPT/DMX interface (in French)
Lptdmx.hex	hex file for the LPT/DMX interface
LPT_DMX_LIB	folder containing the DLL library for controlling the interface under Windows, along with the file ‘readme’ (English)
LPTDMX_TESTER.EXE	test program for the interface
LPTDMX_TESTER_E.DOC	English-language user’s guide for the test program
LPTDMX_TESTER_F.DOC	French-language user’s guide for the test program

Microcontroller Basics Course

part 4: the READS51 C compiler

Anyone who seriously intends to work with microcontrollers must sooner or later use the C programming language. In this final instalment of the Microcontroller Basics course, we use the READS51 C compiler from Rigel.



```
// ------------- READS51 generated header --------------
// module  : a01.c
// created : 19:51:27, Monday, October 09, 2000
// Example routines for the textbook
// Programming and Interfacing the 8051 in C and Assembly
// by S. Yeralan and H. Emery
// (C) 2000, Rigel Press, www.rigelcorp.com
// -----------------------------------------------------

#define TRUE  1
#define FALSE 0
```

```
Linking     c51.obj
            c0.OBJ
            a01.obj
            Sio51.lib
         -> blink.hex
Successful build.
```

Figure 1. READS51 in action.

Up to now we have used assembler and BASIC-52 as programming languages in the Microprocessor Basics Course. Now it's time to work with C, using a compiler that can be selected and downloaded from the following Internet address:

http://www.rigelcorp.com/8051soft.htm

The relevant files to fetch are

SetupReads51.exe and Reads51.pdf.

A C complier translates a source text into pure machine code, in contrast to a Basic interpreter, which only generates intermediate code that must interpreted and executed at run time. C is thus many times faster than Basic.

The C language has been around for a long time and is available for many different systems. Its decisive advantage is that C programs are largely independent of the hardware used. The results of laborious effort can thus be relatively easily ported to other systems. ANSI C was defined as early as 1988 (ANSI stands for 'American National Standards Institute') in order to create a common standard. A smaller version called 'Small C' has been specially developed for microcontroller systems. Although it has certain limitations compared with ANSI C, such as the absence of 'real' variables, it has the advantage that it can be use with very small systems. Various free compilers for Small C can be found on the Internet. For this course we have chosen READS51, since it is particularly suited to novices and has a convenient user interface.

READS51 was specially developed by Rigel for the educational market and is intended to support their microcontroller boards. The company makes this product available to anyone using it for purely private purposes or educational use. Rigel have kindly given *Elektor Electronics* permission to use the compiler for the Microcontroller Basics course. All interested readers should therefore download READS51 from Rigel's Internet site and install it on their systems. By the way, you can also find many other equally interesting help files at this site. All examples for the Microprocessor Basics course have English labels and comments. We simply couldn't

Listing 1. The first sample program.

```
// ———–- READS51 generated header ———–
// module  : a01.c
// created : 19:51:27, Monday, October 09, 2000
// Example routines for the textbook
// Programming and Interfacing the 8051 in C and
Assembly
// by S. Yeralan and H. Emery
// (C) 2000, Rigel Press, www.rigelcorp.com
// ——————————————————————————-

#define TRUE  1
#define FALSE 0

#include <sfr51.h> // P1_0 is defined here
// prototypes
#include <Sio51.h>

main(){
int n;

// —– initialize serial port (9600 Baud) —–
 InitSerialPort0(DEF_SIO_MODE);
                    //DEF_SIO_MODE is defined in
<Sio51.h>
 putc('\n');

// endless loop
 while(TRUE)
  {
   P1_0=0;                 // LED on
   putc('+');
   for(n=0; n<10000; n++); // waste some cycles
   P1_0=1;                 // LED off
   putc('0');
   for(n=0; n<10000; n++); // waste some cycles
   }
}
```

proceed any further without using this international approach.

The best way to get started with READS51 is to use one of the accompanying examples. The project blink can be loaded from Project/Open Project. If you double-click on source text file for the main module, a01.c, the source text will appear in the Editor window (see **Listing 1**).

A C program always has a main function called `main()` that is executed when the program is started. At first glance, the sample program `blink` appears to contain only this function, but in fact some other functions related to the serial interface of the microcontroller are also used. These functions are located in the module `Sio51.h`. They open the serial interface at 9,600 baud (with a crystal frequency of 11.0592 MHz), which is exactly what the *Elektor Electronics* Flash Board needs. Just in case you did not know, the highly successful 89S8252 Flash Board was described in the December 2001 issue of *Elektor Electronics*.

For C beginners, the program notation may at first seem a bit odd, so explanations of some of the details are in order:

```
#define TRUE 1
```
Defines a constant (`TRUE` will be replaced by '1' wherever it appears).

```
#include <sfr51.h>
```
Links in a header file containing definitions.

```
main()
{
...
}
```
Forms the principal function `main`. All the instructions for this function are contained in a block of instructions enclosed by a pair of curly brackets.

```
int n;
```
Declares a variable n of type `integer`, whose allowed range of values is –32768 to +32767. A semicolon (;) terminates the line.

```
InitSerialPort0
     (DEF_SIO_MODE);
```
Calls a function with a passed parameter, in this case a function in module `Sio51.h` that initialises the serial interface.

```
// endless loop
```
A comment, which increases the readability of the program but is not translated with the actual program.

```
while(TRUE)
  {
  ...
  }
```
Forms a loop. In place of `TRUE` for an endless loop, a different condition could be used here to define the condition under which the loop is to be traversed. All instructions that are to be executed in the loop are again enclosed in curly brackets.

```
P1_0=0;
```
An instruction. Here the bit variable `P1_0` is assigned the value '0'.

```
putc('+');
```
Text output via the serial interface. The function `putc` is defined in `Sio51.h`. A text character, which is a variable or constant of the type `char` (`character = text character`, always one byte), is transferred.

```
for(n=0; n<10000; n++);
```
Forms a counting loop, which would be written in Basic as '`For n=1 to 10000: Next n`'. Here the loop does not contain any instructions, as can be seen from the semicolon. A block of instructions enclosed by curly brackets could also be located here.

Even if you haven't yet fully grasped all the subtleties of C programs, it's interesting to see whether this program will run on the

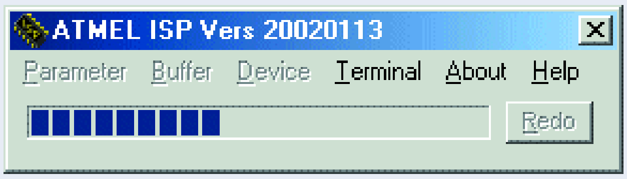
ATMELISP, a new download tool

The simple loader program MicroFlash.exe for downloading programs to the 89S8252 board works only with COM1 or COM2 and does not report back regarding the success of the download, which has led to problems for some users. However, *Elektor Electronics* readers do not sit idle in such situations. Ulrich Bangert (DF6JB) has consequently developed a new and significantly more extensive program named ATMELISP, which allows the Flash memory to be programmed using various types of systems. Besides the Atmel Starter Kit and a proprietary board, the program also supports the *Elektor Electronics* system and the ModuleBus system (EX52-Flash). The new software can be downloaded from the *Elektor Electronics* home page.

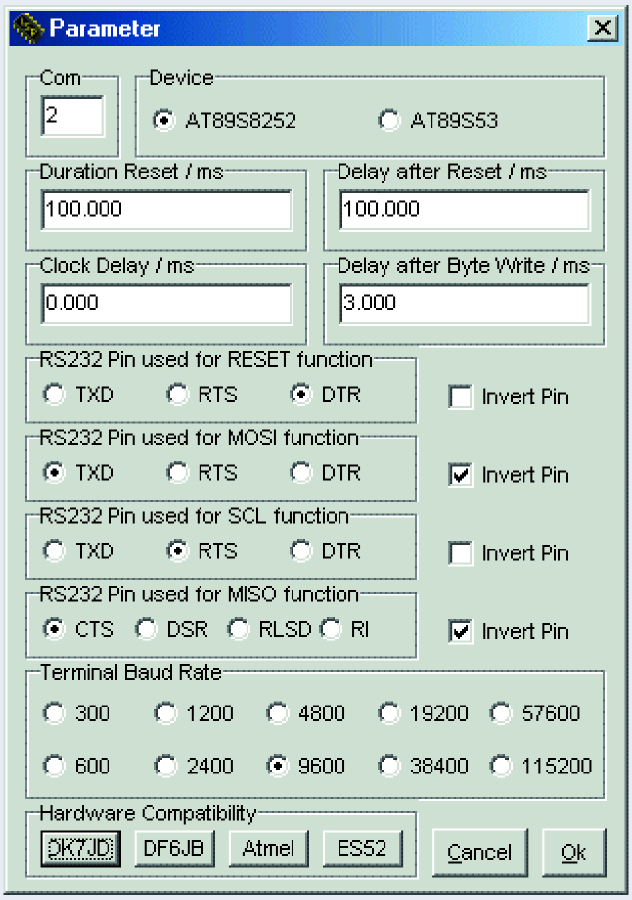
When the zip archive has been unpacked, you will have an .exe program and a comprehensive help file. The start-up screen (**Figure A**) is small and can easily be placed on the monitor next to other applications. Larger windows only appear when program functions are executed. The first thing you must do is to select the serial interface, the connected device and other critical parameters. A click on the button marked 'DK7JD' (which is B. Kainka's amateur radio call sign) configures the proper assignment of the programming lines to the RS232 lines used for the *Elektor Electronics* circuit board. Here you can also see that it is easily possible to use ATMELISP to program any desired circuit board you have developed yourself that uses the same processor, since three lines are simply selected and appropriately assigned. In some cases, it may be necessary to adjust the delay times. Our experience shows that with a relatively slow PC the value of `Clock Delay` must be increased from 0 to 0.01 ms. **Figure B** shows the window for selecting the configuration parameters.

The rest of the procedure can be illustrated using a concrete example. The Flash ROM of the microcontroller is to be loaded with the first sample program from the C compiler. This requires the code to first be read into the buffer. ATMELISP can read files in binary and Intel hex formats. Here the file `Blink.hex` is loaded. It has also proven to be worthwhile to have a quick look at the built-in hex editor after loading the file (**Figure C**), in order to see the content and size of the file.

To program the microcontroller, select `Device/Write Buffer to Code Memory`. Here you must be careful not to confuse the Code Memory with the Data Memory, which is the 2-kbyte EEPROM data region of the microcontroller. Both of these memories can be programmed and read. Besides this, it is possible to load 'lock bits' into the microcontroller in order to prevent the loaded software from being read (**Figure D**). But be careful with the lock bits: if all three bits are set, any further serial programming of the chip is blocked! In this case, it is also no longer possible to erase the entire chip using the program. Only a parallel programming device



A



Com: 2

Device: (●) AT89S8252 ( ) AT89S53

Duration Reset / ms: 100.000

Delay after Reset / ms: 100.000

Clock Delay / ms: 0.000

Delay after Byte Write / ms: 3.000

RS232 Pin used for RESET function: ( ) TXD ( ) RTS (●) DTR — [ ] Invert Pin

RS232 Pin used for MOSI function: (●) TXD ( ) RTS ( ) DTR — [✓] Invert Pin

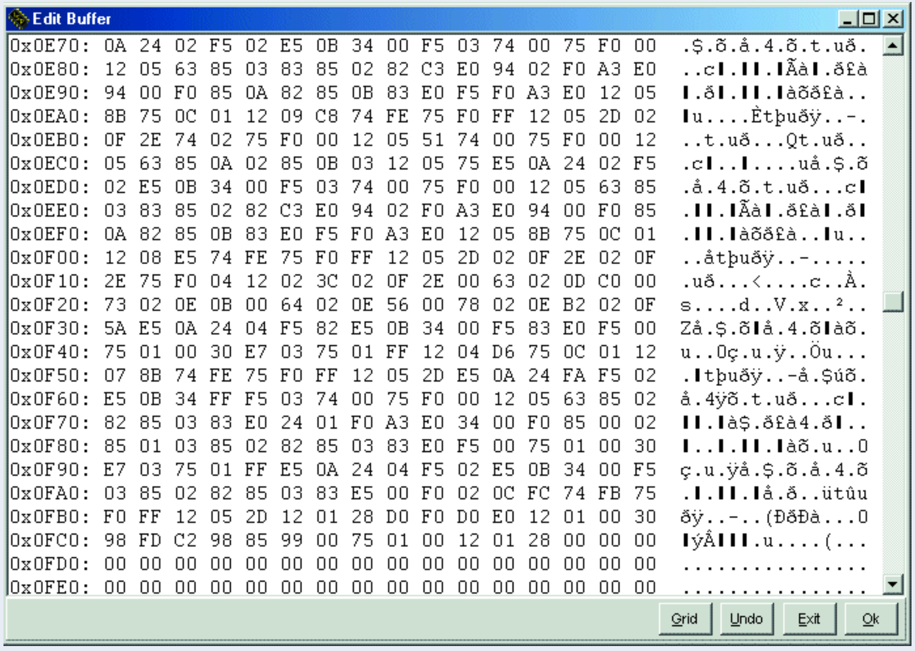
RS232 Pin used for SCL function: ( ) TXD (●) RTS ( ) DTR — [ ] Invert Pin

RS232 Pin used for MISO function: (●) CTS ( ) DSR ( ) RLSD ( ) RI — [✓] Invert Pin

Terminal Baud Rate: ( ) 300 ( ) 1200 ( ) 4800 ( ) 19200 ( ) 57600 ( ) 600 ( ) 2400 (●) 9600 ( ) 38400 ( ) 115200

Hardware Compatibility: DK7JD DF6JB Atmel ES52

B



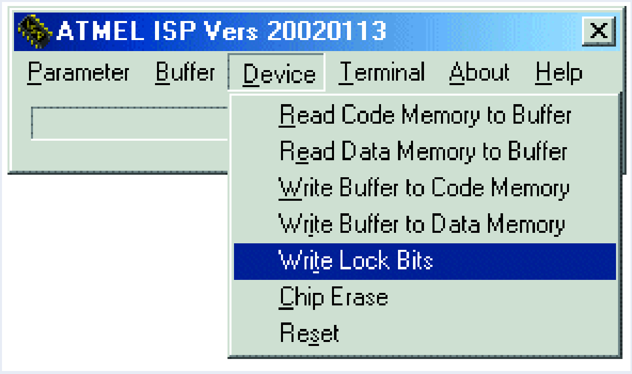
```
0x0E70: 0A 24 02 F5 02 E5 0B 34 00 F5 03 74 00 75 F0 00  .$.õ.å.4.õ.t.uð.
0x0E80: 12 05 63 85 03 83 85 02 82 C3 E0 94 02 F0 A3 E0  ..c▌.▌▌.▌Ãà▌.ð£à
0x0E90: 94 00 F0 85 0A 82 85 0B 83 E0 F5 F0 A3 E0 12 05  ▌.ð▌.▌▌.▌àõð£à..
0x0EA0: 8B 75 0C 01 12 09 C8 74 FE 75 F0 FF 12 05 2D 02  ▌u....Ètþuðÿ..-.
0x0EB0: 0F 2E 74 02 75 F0 00 12 05 51 74 00 75 F0 00 12  ..t.uð...Qt.uð..
0x0EC0: 05 63 85 0A 02 85 0B 03 12 05 75 E5 0A 24 02 F5  .c▌..▌....uå.$.õ
0x0ED0: 02 E5 0B 34 00 F5 03 74 00 75 F0 00 12 05 63 85  .å.4.õ.t.uð...c▌
0x0EE0: 03 83 85 02 82 C3 E0 94 02 F0 A3 E0 94 00 F0 85  .▌▌.▌Ãà▌.ð£à▌.ð▌
0x0EF0: 0A 82 85 0B 83 E0 F5 F0 A3 E0 12 05 8B 75 0C 01  .▌▌.▌àõð£à..▌u..
0x0F00: 12 08 E5 74 FE 75 F0 FF 12 05 2D 02 0F 2E 02 0F  ..åtþuðÿ..-.....
0x0F10: 2E 75 F0 04 12 02 3C 02 0F 2E 00 63 02 0D C0 00  .uð...<....c..À.
0x0F20: 73 02 0E 0B 00 64 02 0E 56 00 78 02 0E B2 02 0F  s....d..V.x..²..
0x0F30: 5A E5 0A 24 04 F5 82 E5 0B 34 00 F5 83 E0 F5 00  Zå.$.õ▌å.4.õ▌àõ.
0x0F40: 75 01 00 30 E7 03 75 01 FF 12 04 D6 75 0C 01 12  u..0ç.u.ÿ..Öu...
0x0F50: 07 8B 74 FE 75 F0 FF 12 05 2D E5 0A 24 FA F5 02  .▌tþuðÿ..-å.$úõ.
0x0F60: E5 0B 34 FF F5 03 74 00 75 F0 00 12 05 63 85 02  å.4ÿõ.t.uð...c▌.
0x0F70: 82 85 03 83 E0 24 01 F0 A3 E0 34 00 F0 85 00 02  ▌▌.▌à$.ð£à4.ð▌..
0x0F80: 85 01 03 85 02 82 85 03 83 E0 F5 00 75 01 00 30  ▌..▌.▌▌.▌àõ.u..0
0x0F90: E7 03 75 01 FF E5 0A 24 04 F5 02 E5 0B 34 00 F5  ç.u.ÿå.$.õ.å.4.õ
0x0FA0: 03 85 02 82 85 03 83 E5 00 F0 02 0C FC 74 FB 75  .▌.▌▌.▌å.ð..ütûu
0x0FB0: F0 FF 12 05 2D 12 01 28 D0 F0 D0 E0 12 01 00 30  ðÿ..-..(ÐðÐà...0
0x0FC0: 98 FD C2 98 85 99 00 75 01 00 12 01 28 00 00 00  ▌ýÂ▌▌▌.u....(...
0x0FD0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x0FE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

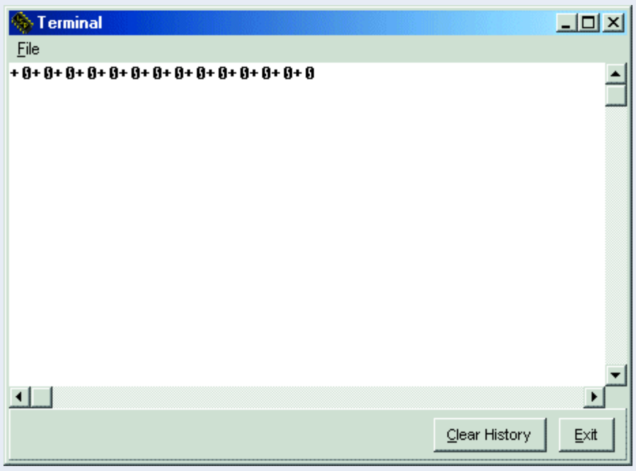
C

can get you out of this trap.

ATMELISP also includes a terminal function (**Figure E**)

D


ATMEL ISP Vers 20020113
Parameter
Buffer
Device
Terminal
About
Help
Read Code Memory to Buffer
Read Data Memory to Buffer
Write Buffer to Code Memory
Write Buffer to Data Memory
Write Lock Bits
Chip Erase
Reset



```
+0+0+0+0+0+0+0+0+0+0+0+0+0+0
```

E

that can be used to view the outputs from the first sample program. This requires the interface cable to be connected to the other interface socket on the board.

Flash Board. Before doing so, you must first compile the program, which means having it translated into machine language. The program can be translated using `Compile/build` or by simply pressing F9. The process is relatively complicated, since the individual object modules must first be translated, following which they are linked together to form a complete program. The final result is a file in Intel hex format named `Blink.hex`. It is located in the project directory `\work\blink` and can also be seen under 'Generated files' as .hex.

The project's Intel hex file can now be simply transferred to the Flash Board using the MicroFlash program. When downloading the program into the Flash memory of the microcontroller, you will notice that in spite of the simplicity of the source text, the translated version is relatively large (4 kbytes). That is primarily due to the module `C51.obj`, which was linked in. This module contains all the functions provided by READS51, including ones that are not actually necessary here.

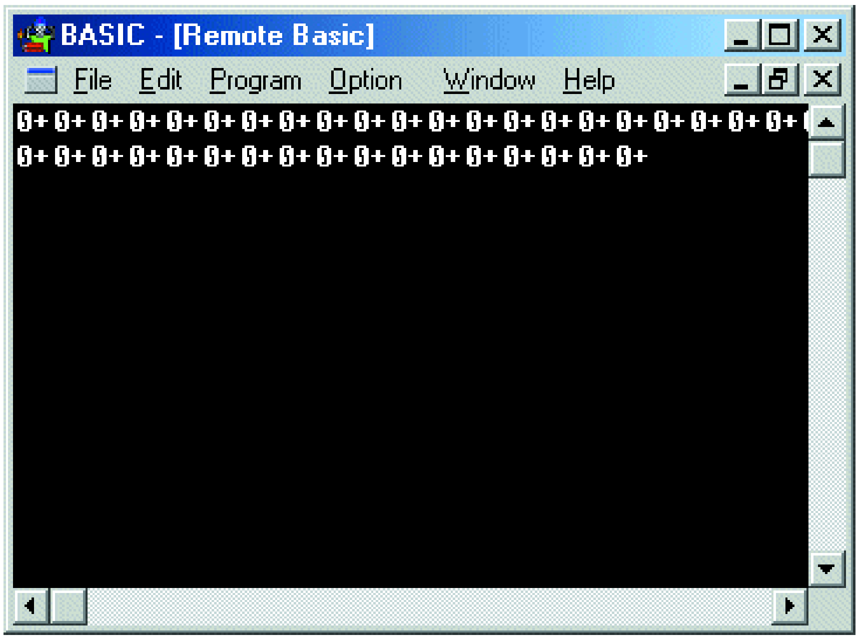
After the program has been successfully downloaded, it's time for testing. Connect a LED and a series resistor between P1.0 and V_{CC} and indeed, it blinks! Of course, people who prefer traditional electronics might comment that the same result could have been achieved more easily using two transistors, but this program does more. It also initialises the serial interface, which now can be used. In order to see that transfers are possible, we need a terminal emulator program.

The simplest approach is to use the Basic terminal program from the course, but the communications parameters must be correct. The C program uses 9600 baud, while `Basic.exe` normally uses 19,200 baud. However, it is easy to change the transfer rate used by this program. Just open the file `Basic.ini` with a text editor and add the line '`Baud=9600`' (see **Listing 2**).

Listing 2. Content of the modified .ini file for the Basic terminal.

```
[AHBASIC]
COM=2
Baud=9600
```

The terminal program will now show what the C program sends, which is a series of '0' and '+' characters that alternate with each change of state of the switched output P1.0 (see **Figure 2**). This is because the program calls the function `putc` to output individual text characters.



```
0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+
0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+
```

Figure 2. Text outputs in the Editor window.

Fast port outputs

Now that we've seen the first example, it's time to write our own C program. Let's start with a program that simply generates fast port outputs, so that we can make some comparisons with the programming languages we used before.

The first thing to do is to create a new project. A new project name must be entered under `Project/New Project`. Here we choose the name `Output`. READS51 now creates a new directory with the name `Work\Output`. Next, we open a new module under `Module/Create Module` and give it the name `Output.` The port output program is shown in **Listing 3**.

The compiler has to know which project it is supposed to translate. This is accomplished by running `Project/Set Project Active` once. If you forget to do this, the last project that was processed will be translated. The newly generated program code `Output.hex` can now be transferred to the program memory of the processor using MicroFlash. All that we need to verify this function is an oscilloscope or a set of headphones. The highest-frequency signal will be found on P1.0. It has a frequency of 4 kHz; the period is 250 μs. The program needs 125 μs for each new port output.

In contrast to Basic, C allows different types of variables. The first sample program uses a variable n of type `int`, which means an integer variable with a value range of –32768 to +32767, while the second example uses a variable of type `unsigned char`, which corresponds to a byte. However, experiments have shown that this does not result in any significant difference in execution speed.

The effectiveness of the individual programming languages can now easily be compared (see **Table 1**). The most important criteria are the amount of memory taken up by the program, its speed and the ability to implement a stand-alone program for the Flash microcontroller. Although C programs use the system RAM, the complete program is located in the Flash ROM alone. This is why a C program starts up again when the voltage is switched on, in contrast to a BASIC-52 program.

A frequency divider in C

In looking for a somewhat more complex task, we remembered the divide-by-20 frequency divider we already wrote in BASIC-52. A direct comparison of the two programs can help us recognise differences in the structure and notation. This program has been given

Listing 3. A program for fast port outputs.

```
// ———–– READS51 generated header ———––
// module  : C:\Rigel\Reads51\Work\Output\Output.c
// created : 12:33:17, Friday, November 09, 2001
// ———————————————————————————–

#define TRUE  1
#define FALSE 0

#include <sfr51.h> // P1 is defined here

main(){
unsigned char n;
// endless loop
 while(TRUE)
  {
   for(n=0; n<256; n++)
    {
     P1=n;
    }
   }
}
```

Listing 4. Divide-by-20 frequency divider.

```
// ———–– READS51 generated header ———––
// module  : C:\Rigel\Reads51\Work\Count\count.c
// created : 18:26:23, Monday, November 12, 2001
// ———————————————————————————–

#include <sfr51.h>

void pulse(void){
  while(P1_0);
  while(!(P1_0));
}


main(){
  int n;
  n=0;
  while(1)
    {
     while (n<10)
     {
       pulse();
       n=n+1;
     }
     P1_1=1;
     while (n<20)
     {
       pulse();
       n=n+1;
     }
     P1_1=0;
     n=0;
    }
}
```

Table 1. Comparison of the three programming languages

Language	Memory	Loop time	Autostart
Basic-52	8 K ROM, RAM	2500 µs	Only with EEPROM
READS51 C	>4 K ROM, RAM	125 µs	yes
Assembler	< 1 K ROM	3 µs	yes

the name `Count` (see **Listing 4**).

The listing shows a rather decisive advantage of C as a programming language: the programmer is forced to use a structured style. This makes the program easier to read. Here we have the function `pulse`, which suspends the progress of the program while waiting for the next positive edge on P1.0. When using a function, it is common to pass in a value and receive another value in return. However, the function `pulse` does not return any parameter (`void` = empty), and no parameter is passed to it. C does not make a distinction between functions and procedures, as is customary in Pascal and Delphi; it has only functions.

The bit variable `P1_0` yields either '1' or '0'. As long as the condition following `while` is true (= 1), a loop is executed. The second loop contains the actual condition in negated form, which is expressed by the exclamation mark '!' (`!(P1_0)`). The second loop is thus exited when the input level changes from '0' to '1', which means when a positive edge is detected. The main routine calls the function `pulse` at two places: once for n = 0, 1, …, 9 and again for n = 10, 11, …, 19.

Here again the critical question is, what is the highest input frequency that can be applied without any counting errors? For this test, we used a function generator connected to P1.0 and an oscilloscope connected to P1.1. The measurement yielded an upper frequency limit of 3 kHz. To refresh your memory, BASIC-52 only managed a frequency of 50 Hz, while up to 100 kHz is possible using assembler.

(010208-5)

Literature:

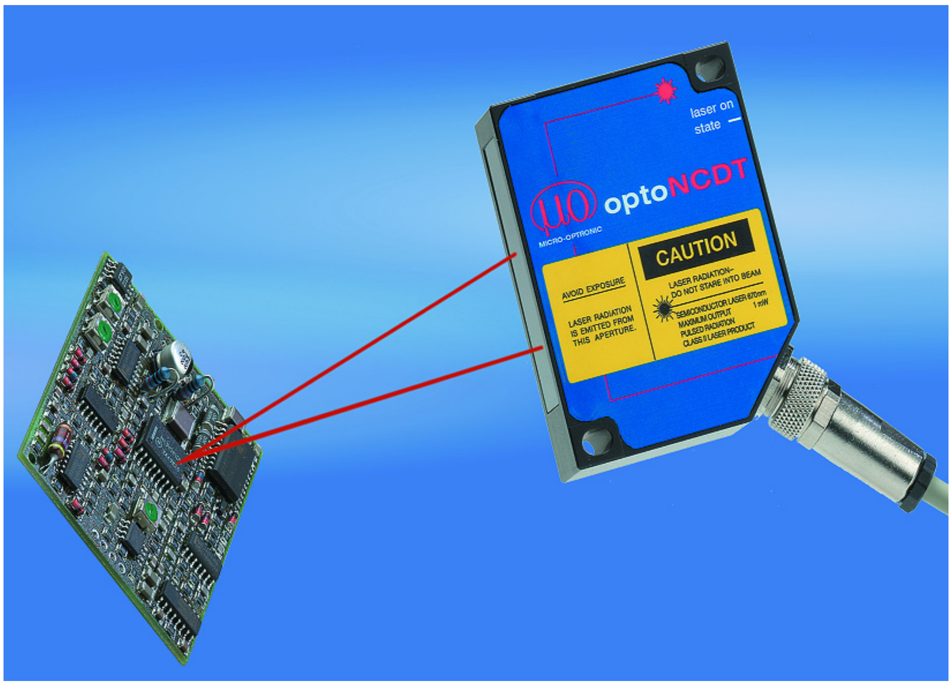
Sencer Yeralan/Helen Emery
Programming and Interfacing The 8051 Microcontroller in C and Assembly
Rigel Press 2000

Contactless Distance Measurement

Sensors and principles

By J. Häuser

Contactless distance measurement is becoming more and more important in analysing vibration, alignment, position or bending. This article explains the most popular measurement methods and discusses their applicability.



Contactless distance measurement finds many practical applications: if the motion of an object would be damped or otherwise affected by coupled mass or the force exerted by a measurement device; when sensitive surfaces must not be damaged; or when rapid motion must be tracked.

Such testing and measurement tasks are commonplace in research and development, automation, quality control and machine control applications. For all these tasks a large number of companies offer numerous models of sensor employing various measurement principles. Three methods of contactless distance measurement have become dominant in the last few years: the eddy current principle, the capacitive principle and the optical triangulation principle.

The eddy current principle

The eddy current principle has special status in the class of inductive measurement methods. The effect is based on the dissipation of energy stored in a resonant circuit when eddy currents are induced in a nearby electrically conductive metallic object.

If, as shown in **Figure 1**, a metal plate is brought near to a coil fed with a high frequency alternating current, the electromagnetic field will induce eddy currents in the plate. By Lenz's law, the field due to the eddy current opposes the inducing field. The consequent energy loss changes the effective inductance of the sensor coil. Hence the amplitude of oscillations in the sensor coil changes as a function of the distance from the measured object (the metal plate). This principle, also known as the 'eddy loss principle', requires an oscillator with stable amplitude and frequency, usually operated in the range 1 MHz to 2 MHz. An air-cored (rather than ferrite-cored) coil is used.

The capacitive principle

The capacitive contactless distance measurement principle is based on the theory of the ideal plate capacitor (**Figure 2**). Changing the distance between the two plates leads to a change in its capacitance. In the measurement system the sensor and the measured object form the two plates. An alternating current of constant frequency is made to flow, and the amplitude of the alternating voltage across the two plates is proportional to the distance between the sensor and the measured object. At the same time an adjustable offset voltage is generated in the control electronics. After demodulation the two voltages are passed to a differential amplifier to generate the output as an analogue signal. By measuring the reactance X_C of the plate capacitor we obtain, without further linearisation, a directly proportional relationship. In practice, the sensor is constructed as a guard-ring capacitor and linearity is almost perfect, independent of the conductivity of the metal in the measured object.

Capacitive sensors can also be used with insulating materials. Extra circuitry is required to obtain a linear output signal with such objects, and a stable dielectric constant is required to obtain a reliable characteristic.

The optical triangulation principle

This principle, in which a pulsed laser beam is reflected off the surface of the measured

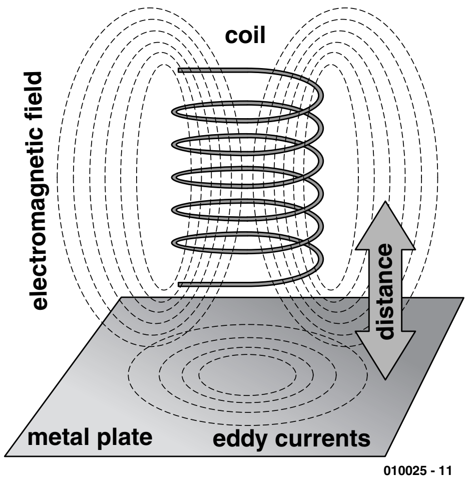
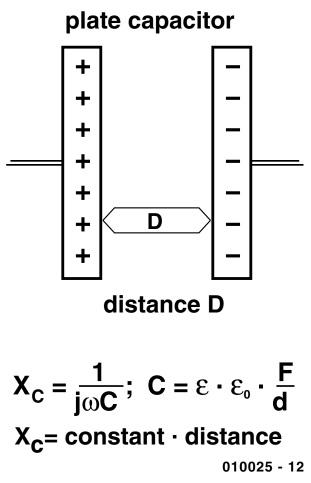


Figure 1. The eddy current principle.



$$X_C = \frac{1}{j\omega C};\ C = \varepsilon \cdot \varepsilon_0 \cdot \frac{F}{d}$$

$$X_C = \text{constant} \cdot \text{distance}$$

Figure 2. The capacitive principle.

Table I: Comparison of contactless measurement principles

Eddy current principle

Advantages	Limitations
– Works with all electrically conductive metals whether ferromagnetic or not	– Output signal and linearity dependent on electrical and magnetic properties of materials used
– Small sensor	– Individual linearisation and calibration required
– Insensitive to dirt, dust, moisture, oil, dielectric substances in measurement gap	– High oscillator frequency limits sensor cable length to 12 to 18 m.
– Usable in electromagnetically sensitive applications	– Sensor diameter (and measurement patch diameter) increase as maximum range is increased.
– Wide operating temperature range	
– High accuracy	

Capacitive principle

Advantages	Limitations
– Independent of metal used in measured object: sensitivity and linearity remain the same.	– Sensitive to dielectric materials in the measurement gap, and hence only usable in clean and dry environments.
– High temperature stability, as conductivity changes due to temperature have no effect.	– Sensor cable must be short, owing to effect of cable capacitance on tuning of resonant circuit.
– Also usable with non-conductive measured objects.	– Sensor diameter (and measurement patch diameter) increase as maximum range is increased.

Optical triangulation principle

Advantages	Limitations
– Small measurement patch diameter.	– Limited usefulness with smooth surfaces (mirrors, glass, CDs, polished metals) or surfaces with low reflectivity (matt black surfaces)
– Sensor can be far from measured object.	– Limited usefulness with transparent or partially transparent surfaces (glass, ceramics, plastics such as Teflon).
– Large measurement range possible.	– Space through which beam passes must be unobstructed and free of dust.
– Independent of materials used.	

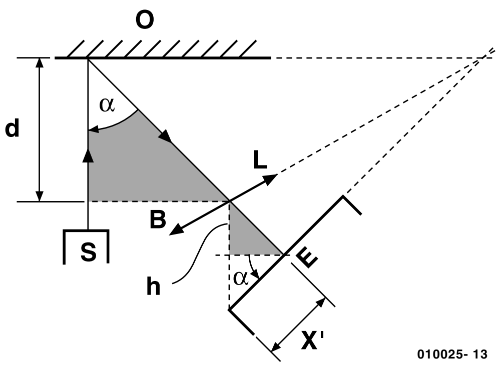


Figure 3. The optical triangulation principle.

object, is based on similar triangles, see **Figure 3**. The first triangle is the 'object triangle', between object and lens, and the second is the 'image triangle', between lens and detector. The detector is either a lateral effect diode or a CCD row sensor. A lens is required in the path of the diffuse rays reflected from the object to allow the angles of incidence and reflection to be different. This is the only way to resolve depth. Further, the detector must be set at a known angle. Here the Scheimpflug rule should be observed, which states that optimal focussing is obtained when the object, lens and detector planes intersect at a single point. Hence we have the following relationship:

$$\Delta d = \frac{B \cdot h}{\cos(\alpha)} \cdot \frac{1}{\Delta x}$$

Manufacturers convert the physical distance measured by the sensor into a standardised voltage range (for example 0 V to 10 V or 0 (4) mA to 20 mA). Alternatively, the value is digitised and transmitted over an RS232 or RS485 interface to a PC. With the aid of a suitable card for the PC (see 'PCI-Hosted Measurement Cards', *Elektor Electronics*, October 2000), and suitable software, an enormous range of test and measurement problems can be solved.

(010025-1)

Literature

Technical note T01d produced by:

Micro-Epsilon Messtechnik
GmbH & Co. KG,
Postfach 1254
D-94493 Ortenburg
Germany

Further information is available from:

MJH Software
Mr. J. Häuser
Am Stadion 28
D-07629 Hermsdorf
Germany
Tel./fax: +49 36601 82313
E-mail: jhaeuser@tridelta-hermsdorf.de

Atmel Micro Programmer

September 2001,
(010005-1)
When a ‘narrow-DIL’ ZIF socket is used in position K3, pin 10 is not connected to ground. This is easily corrected by soldering a short wire to pin 10 of the ‘wide’ socket.
The first version of the software failed to program the last byte. An updated version is available from our website (Free Downloads page).

Digital Benchtop Power Supply

November & December 2001
(000166-1/2)
The 10-µF electrolytics in this circuit appear with different voltage ratings in the parts list and the schematic. The following is offered as guidance: C3, C13-17 and C20 should have a minimum working voltage of 16 volts. C19 has to be rated at 35 V or higher. As usual, higher voltage ratings are always possible.

Miniature PCM Model Control

October & November 2001,
010205-1/2
In the receiver circuit, p-channel FET T4 is used in a wrong configuration, which causes an anti-parallel connection of D2 and the internal protection diode of the FET. Consequently, the motor is short-circuited. The problem may be solved by using an n-channel FET for T4 (for example, an SUP75N03), and exchanging the source and drain connections on the board. The source is then connected to the positive supply voltage. This modification is easy to implement on the PCB. Instead of the p-channel FET mentioned in the parts list, simply use an n-channel FET like the SUP75N03 and mount it the other way around as compared with the PCB overlay indication.

Remote Process Control using a Mobile Phone

January & March 2002,
010087-1/2
The numbers 0-15 in the second column in Table 1 (Part1) indicate the pins of the configuration list for the SMS chip. The port pins of ports 1 and 2 are simply numbered 0-15, allowing the user to select which individual pins become inputs or outputs.
The printed circuit board layout (Figure 1 on page 21) has a short-circuit in the supply line near K8. At the top side of the board, one of the two pads near the edge is connected to ground. The four short connections to ground have to be removed. Ready-made PCBs obtained through our Readers Services have been repaired.

IR Code Analyser

October 2001, 010029-1
The schematic and the parts list need to be corrected to read: R1 = 100Ω; R2 = 220Ω.
If the analyser fails to respond to IR signals, IC2 is probably unable to pull its output sufficiently low. Not all IR receiver ICs suggested for this circuit are capable of sinking enough current. In some cases, it may be necessary to increase the value of pull-up resistor R3 to 2.2 kΩ or 2.7 kΩ, although that may cause the LED to light less brightly.

The content of this note is based on information received from manufacturers in the electrical and electronics industries or their representatives and does not imply practical experience by Elektor Electronics or its consultants.

BASH® IC Amplifier

an entirely new concept

By utilising a special processor and a variable step-down converter, STMicroelectronics in partnership with Indigo Manufacturing have succeeded in developing an integrated amplifier with remarkably high efficiency, without sacrificing audio quality.

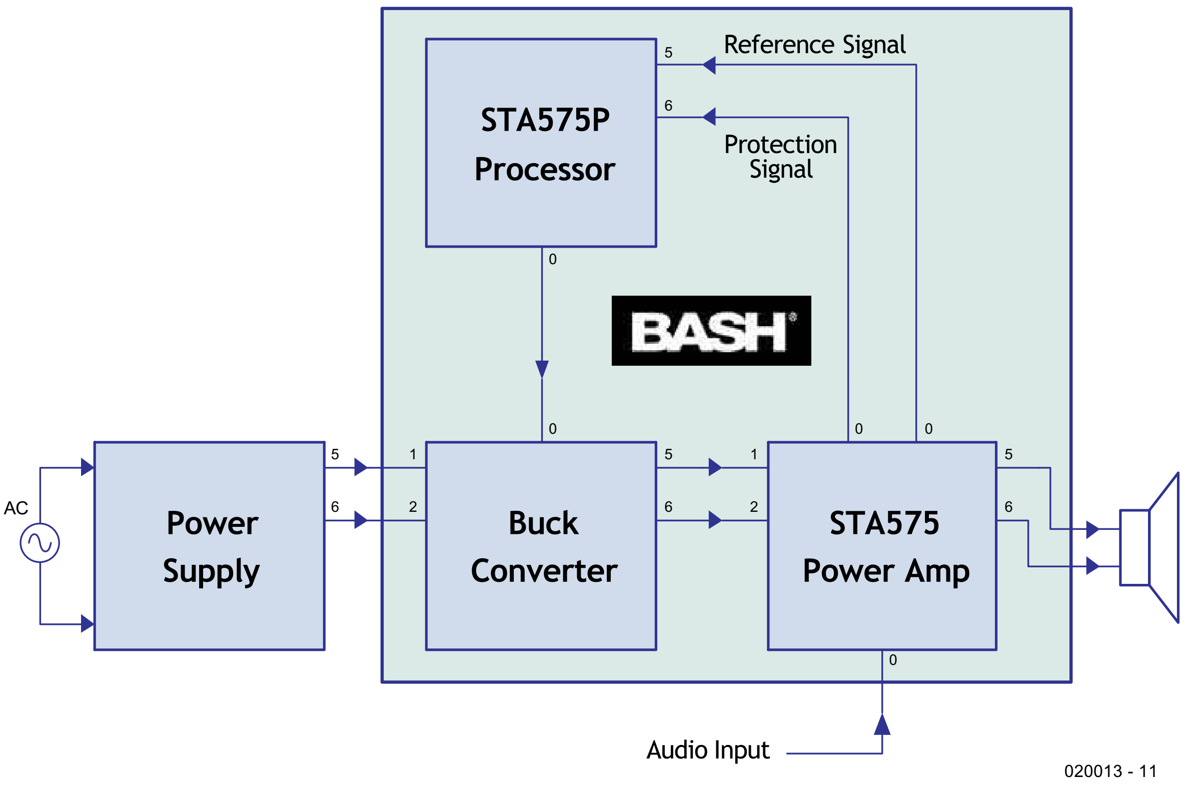


Figure 1. Block diagram of the BASH principle.

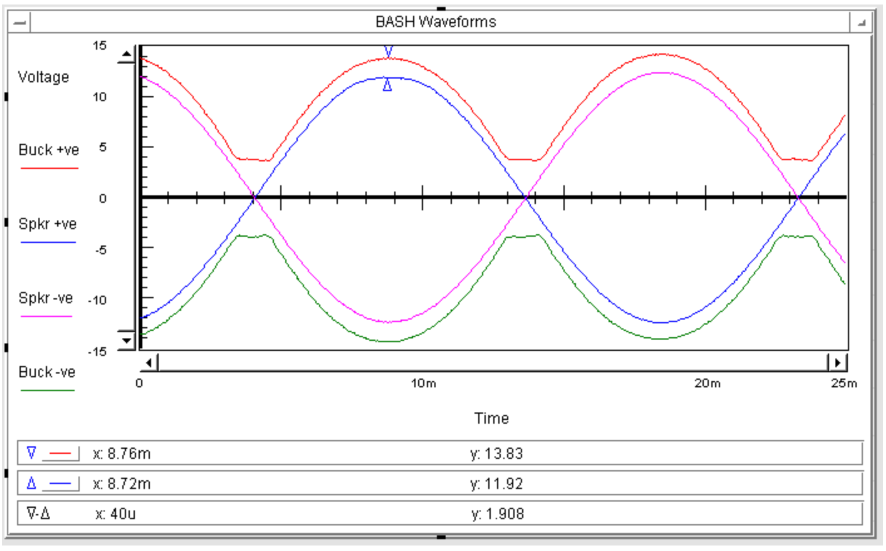
Audio enthusiasts are likely to have widely varying opinions on many different subjects, but they are also likely to agree on one thing. If they had the choice, they would prefer an amplifier that delivers perfect quality, is able to deliver quite a bit of power, and is physically as small as possible.

This combination of characteristics is difficult to obtain in practice. Quality and power can be simultaneously realised. But a quality amplifier demands a sizeable power supply and a generous quiescent current in a Class AB design, for example (not to mention Class-A). This all adds up to considerable heat generation that has to be disposed of using an adequately sized heatsink. Such an implementation is not particularly compact.

Power and compactness are two characteristics that can be realised at the same time relatively easily. But this is only possible with frugal quiescent currents (Class B for example), and the subsequent trade-off of reduced quality. This is not an ideal compromise either.

The middle ground is taken by the so-called Class D design which is based on pulsewidth modulation. The inherent disadvantage of Class D is the extensive filtering required at the output of the power switching stage, as well as the strong interference caused by the switching itself. The total power loss is much reduced. It is an excellent system, but here, the weak point is that this switching leads to interference, which adversely influences the characteristics of the amplifier.

It will have become obvious from this discussion that the ideal combination of power, quality and efficiency has not been achieved yet. The engineers at ST have revisited all these topics once more in order to develop a new concept for the company Indigo Manufacturing. And they have indeed succeeded in realising a design with a significant improvement in efficiency without compromising the quality.



∇	x: 8.76m	y: 13.83
Δ	x: 8.72m	y: 11.92
∇-Δ	x: 40u	y: 1.908

020013 - 12

Figure 2. With BASH, the voltage drop across the power stage transistors remains constant, independent of the output amplitude.

BASH

The only way to improve the efficiency is to, one way or another, vary the power supply voltage depending on the output amplitude of the amplifier. This way, unnecessary dissipation can be avoided. The difference here is that the designers were looking for a way to achieve this without switching, because this would inevitably lead to distortion.

Figure 1 shows the block diagram of the design they arrived at. We can see that besides the usual power supply and power amplifier, two additional blocks have been added: a processor and a buck (step-down) converter.
A reference signal is derived from the power amplifier stage (the 'BASH signal') and applied to the processor. The processor analyses the signal and accurately calculates the demands this signal places on the power supply. This is the basis for generating the gate-drive signal for the step-down converter.

The converter, connected in series with the unregulated power supply, translates this gate signal into an appropriate supply voltage for the power amplifier stage. By continually adjusting the power supply voltage, depending on the amplitude of the amplified audio signal, a constant voltage drop has been achieved across the output transistors, completely independent of the output amplitude. This effect is shown in **Figure 2**.

The result is that unnecessary dissipation has been avoided even though a linear Class AB design is used as the power stage. The efficiency thus obtained is around 85% and is comparable to a Class D amplifier, but the amount of interference is considerably lower. As a consequence, the designers claim that the distortion of a BASH amplifier can be as

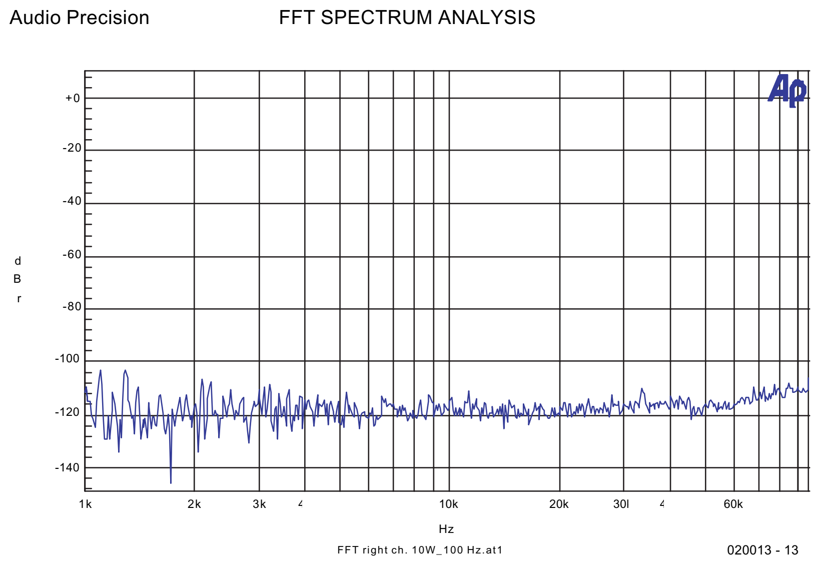


Figure 3. In contrast to a Class D amplifier, no switching artefacts are visible in the output spectrum of the BASH design.

low as that of any other arbitrary Class AB amplifier.

The graph of Figure 3 shows the results of an FFT spectrum analysis of the BASH output signal. It is obvious that this spectrum contains no recognisable switching artefacts.

Another advantage of this BASH system that must not go unmentioned, is the fact that in a multiple channel installation it is possible to have only one power supply and one step-down converter, if that is desired.

Amplifier Modules

Three different BASH amplifier modules have been announced: two stereo versions with output powers of 2 × 50 W en 2 × 75 W (the STA550 and STA575 respectively) and one mono bridge amplifier that can deliver 150W (the STA5150). They are all three identical in design, so for the example we will limit ourselves to the description of the STA575. **Figure 4** shows the block diagram for the circuitry inside the IC amplifier.

Most blocks are powered by means of a fixed positive and negative power supply voltage. Only the power stages are supplied by two external voltages that follow the audio signal.

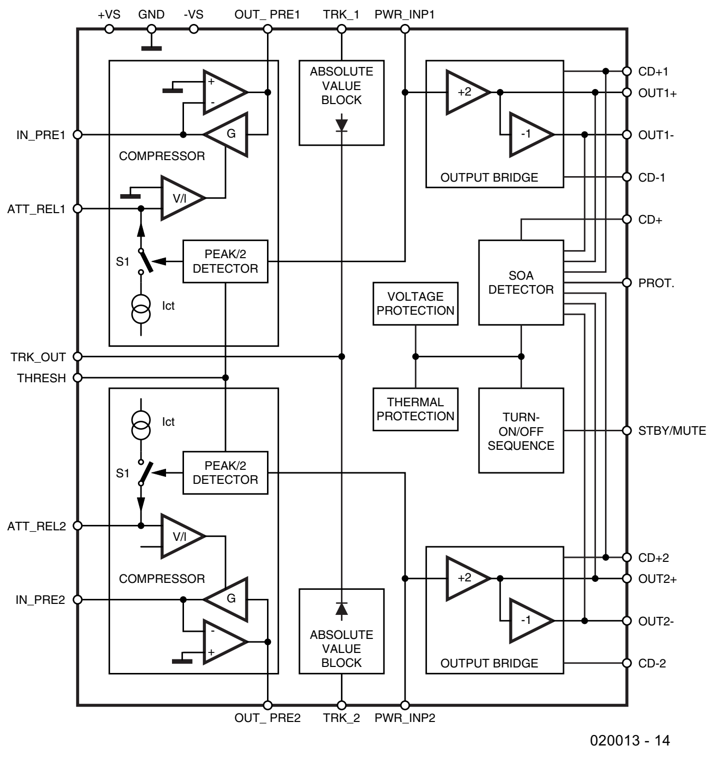
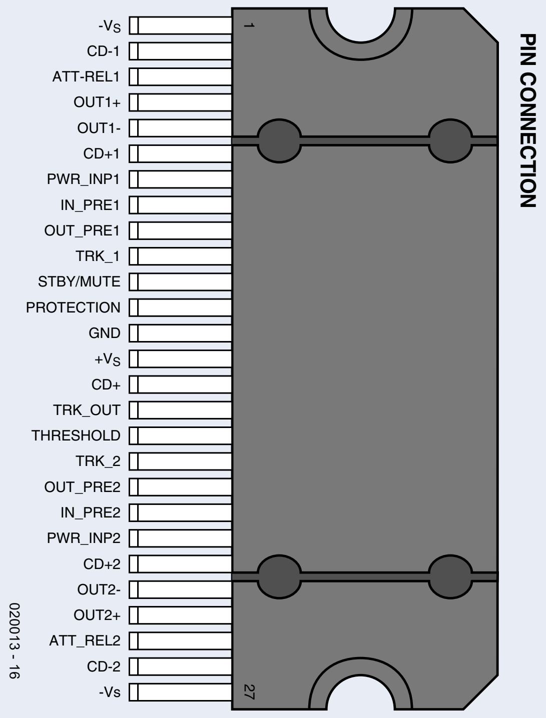


Figure 4. Block diagram for the internal circuitry of the amplifier module, in this case the STA575 which is able to deliver 2 × 75 W.

Table 1. Connections to the STA550/575

Pin-number	Name	Description
1	-Vs	negative bias power supply
2	CD-1	channel 1, variable negative power supply
3	Att Rel1	attack/release channel 1
4	Out1+	channel 1, loudspeaker output positive
5	Out1-	channel 1, loudspeaker output negative
6	CD+1	channel 1, positive power supply
7	Pwr Inp1	channel 1, power stage input
8	In pre1	channel 1, pre-amp input (virtual ground)
9	Out pre1	channel 1, pre-amp output
10	Trk 1	channel 1, absolute-value-block input
11	Stby/mute	standby/mute-control voltage
12	Protection	protection signal for STABP01 digital processor
13	Gnd	analogue ground
14	+Vs	positive bias power supply
15	CD+	variable positive power supply
16	Trk out	reference output for STABP01 digital processor
17	Threshold	compressor threshold input
18	Trk 2	channel 2, absolute-value-block input
19	Out pre2	channel 2, pre-amp output
20	In pre2	channel 2, pre-amp input (virtual ground)
21	Pwr Inp2	channel 2, power stage input
22	CD+2	channel 2, positive power supply
23	Out2-	channel 2, loudspeaker output negative
24	Out2+	channel 2, loudspeaker output positive
25	Att Rel2	attack/release channel 2
26	CD-2	channel 2, variable negative power supply
27	-Vs	negative bias power supply



Each channel has a compressor circuit with a special transfer function. Its purpose is to avoid any dynamic limitations that may be present in the BASH-system. In order to make the system as general-purpose as possible, the attack and release times of the threshold levels are externally programmable. The tracking signal for the external digital step-down converter is generated by the 'absolute value block', which rectifies the output from the compressor. The outputs from these stages are decoupled with a diode to simplify the summing of these signals in a multiple channel amplifier.

The power stages are fitted with a special output pin in order to make AC decoupling possible. This is done so that any DC offset at the compressor output can be eliminated. The power stage has a voltage gain of 4 (+12 dB).

An ingenious circuit functions as the power sensor for the transistors in the power stage and, together with the external converter, provides the intended reduction of the power supply voltage. In addition, there is also a current limit circuit and a temperature sensor to protect the IC itself. Moreover, with an external voltage on the STBY/MUTE pin, both amplifiers can be placed in mute mode, which guarantees on- and off-switching without spurious noises.

The power amplifier ICs are housed in a 27-pin, so-called, Flexiwatt27 package; **Table 1** shows the pin-out for both stereo versions. The mono version is practically pin compatible, only in this case pins 18 through 21 and 25 are not connected.

Digital processor

The block 'processor' in **Figure 1** is implemented with the specially designed STABP01. This is a digital processor with the specific purpose of driving the digital step-down converter. So a BASH amplifier can only be realised with this particular processor.

Figure 5 shows a simplified block diagram for the internals of the STABP01. The manufacturer is still reluctant to divulge too many details. The details of its operation are limited to the information that the processor determines the characteristics of the audio signal and because of this determines the driving signal for the step-down converter. Exactly how this is achieved is not mentioned in any of the documentation. There are a few recognisable details in **Figure 5**, of course, but not the entire approach. An additional comment states that the converter changes the pulse signal into a corresponding power supply voltage for the power amplifier stage, but we already knew that. We also knew already that the processor forms an essential part of the feedback system, where the power supply level of the power amplifier is continually adjusted, based on the instantaneous value of the audio signal. Detailed information, as stated before, is not available yet. Hopefully they will soon reveal some more practical information and provide a detailed application example, because if the BASH system delivers what it promises, it is interesting enough to be thoroughly examined in detail.

(020013-1)

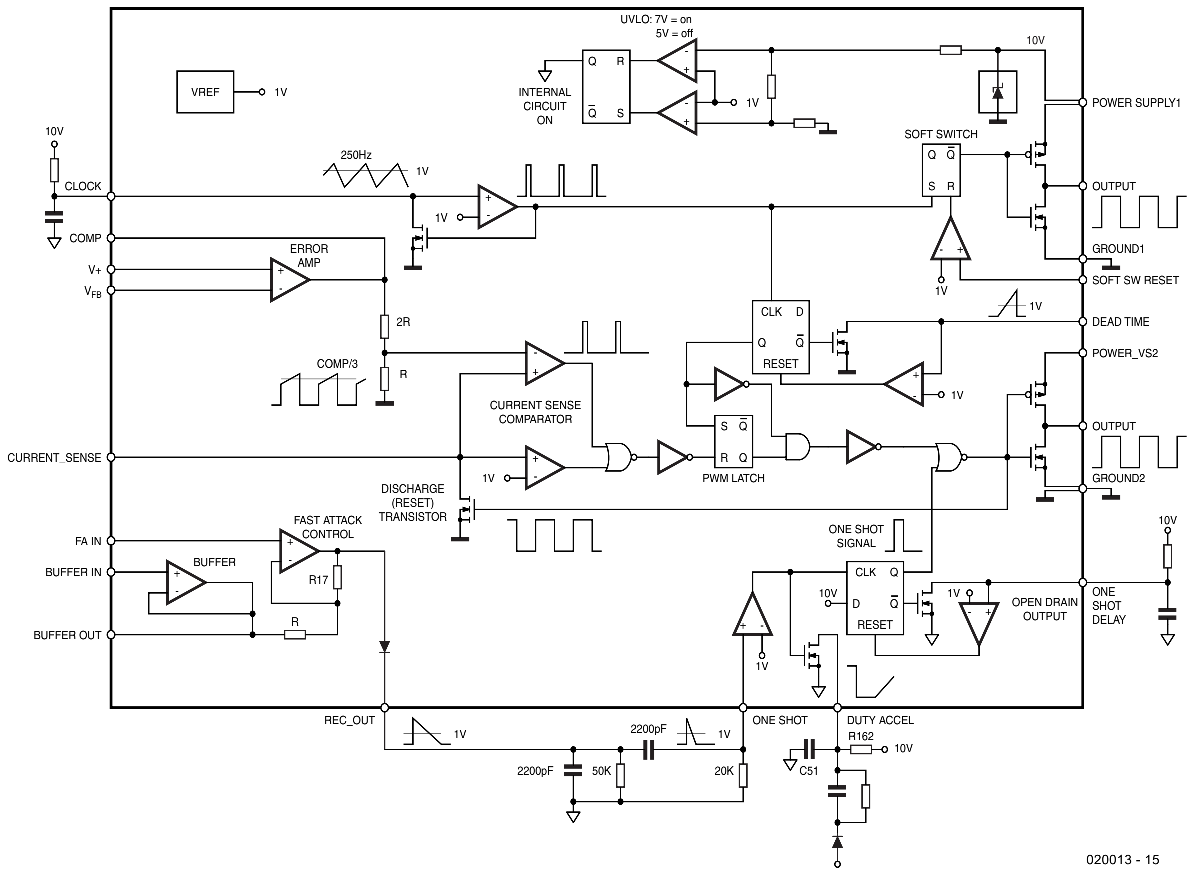


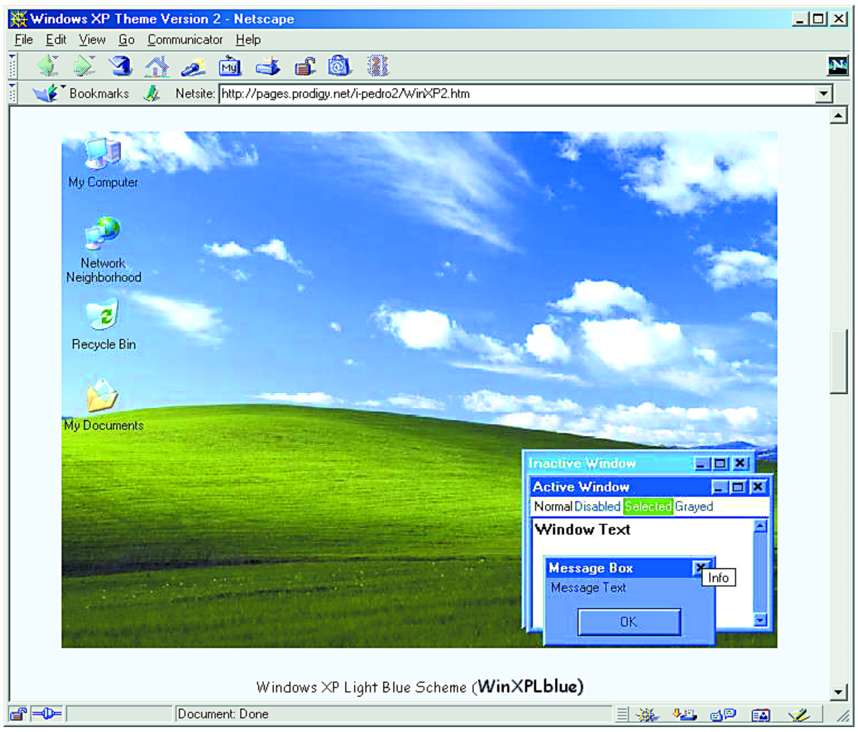
Figure 5. Block diagram of the digital BASH processor STABP01.

Mimic Windows XP

an ersatz version of the latest OS

By Harry Baggen

The latest operating system from Microsoft, called XP, contains a number of significant improvement over its predecessors. At a price, however, because an upgrade from 95/98/ME to XP is pretty expensive. If you are not convinced of all the technical tweaks in XP, but still like its appearance, there plenty of opportunity for 'home improvement' using the XP wallpapers and icons that can be found on the Internet.


Windows XP Theme Version 2 - Netscape
File Edit View Go Communicator Help
Bookmarks
Netsite: http://pages.prodigy.net/i-pedro2/WinXP2.htm
My Computer
Network Neighborhood
Recycle Bin
My Documents
Inactive Window
Active Window
Normal Disabled Selected Grayed
Window Text
Message Box
Message Text
OK
Info
Windows XP Light Blue Scheme (WinXPLblue)
Document: Done

Clearly, the XP operating system is a much more stable basis for Windows programs that any of its predecessors. This is the result of a lot of hard work 'under the bonnet'. Also, significant improvements have been achieved as compared with the 9x and ME releases. However, if you're new to XP, the first thing you will notice is its desktop and icons rather that any of its improved features. Clearly, the Windows programmers have give en a lot of attention to 'cosmetics' this time. It alls looks a bit more 'pop art' with rounded icons, pastel colours and coloured backgrounds.

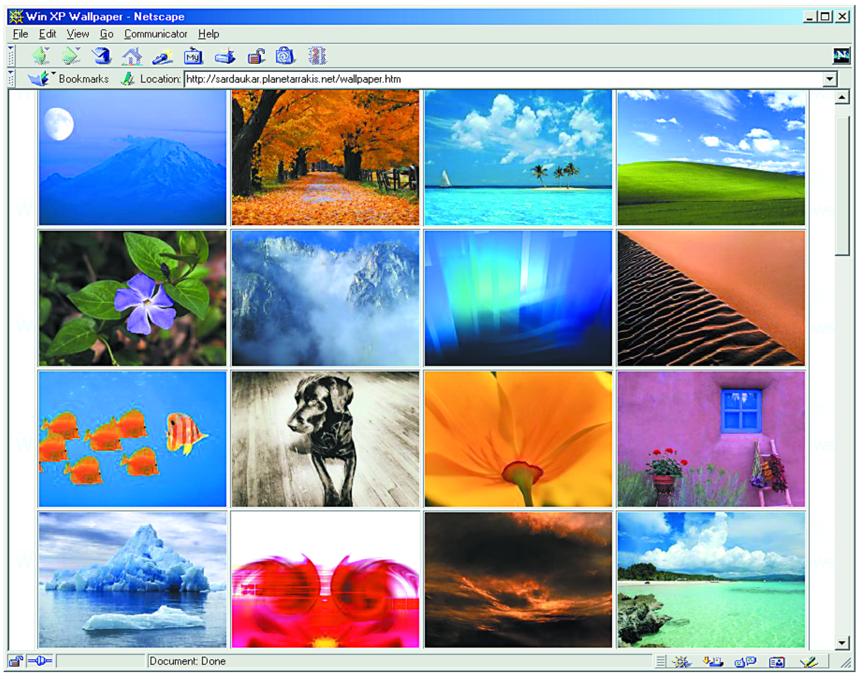
The new 'look' seems to have been received well by many Windows users, and computer addicts will no doubt wonder if and how the XP appearance can be applied to Windows 95, 98 or ME. All without actually buying an XP upgrade, of course, because that to many is nothing short of being robbed of hard earned money. Unfortunately, XP offers a number of desktop options that are not found in older versions of Windows. Unless, of course, we call in the help of a fine selection of software tools! More about these further on.

To start with, the simplest method. We locate a set of cursors and icons that look like those of XP. Next we add to these an XP wallpaper. If desired, the blend can be completed with an XP-ish screensaver and we're in business.

Although collections of XP icons float around on the Internet, it is far more useful to have a program available that adapts everything in one go. **XP Icons** [1] is a shareware program that replaces your existing icons by XP-like equivalents (there are more than 90). Furthermore, this little program allows you to apply a transparent background to the text part of the icons. This will prove useful if you like changing the desktop appearance a lot.

A slightly less extensive alternative is **XP Icon Raider [2]** which does roughly the same but comes free of charge.

For a suitable background you may want to visit one of the countless wallpaper or XP websites. We came across two that offer a really fine selection of XP backgrounds and associated photographs: **Windows**


Win XP Wallpaper - Netscape
File Edit View Go Communicator Help
Bookmarks Location: http://sardaukar.planetarrakis.net/wallpaper.htm
Document: Done

eXperience [3] and **Win XP Wallpaper** [4]. A background is simply copied into the Windows folder, whereupon it may be selected by right-clicking on Desktop - Properties - Background. The function 'Stretch' may be used to make the background design cover the entire screen.

Thanks to the Themes in Windows 98 and ME, icons, mouse pointers, backgrounds and sounds may be replaced in one go. Some programs that employ this function may be found on the Internet. An attractive set of XP themes has been designed by **Peter Wilcox** [5]. This set is available for downloading from Peter's own website or one of the many freeware and shareware sites. After the installation process, you'll find three XP themes in the Theme overview, and you may choose the one you like best.

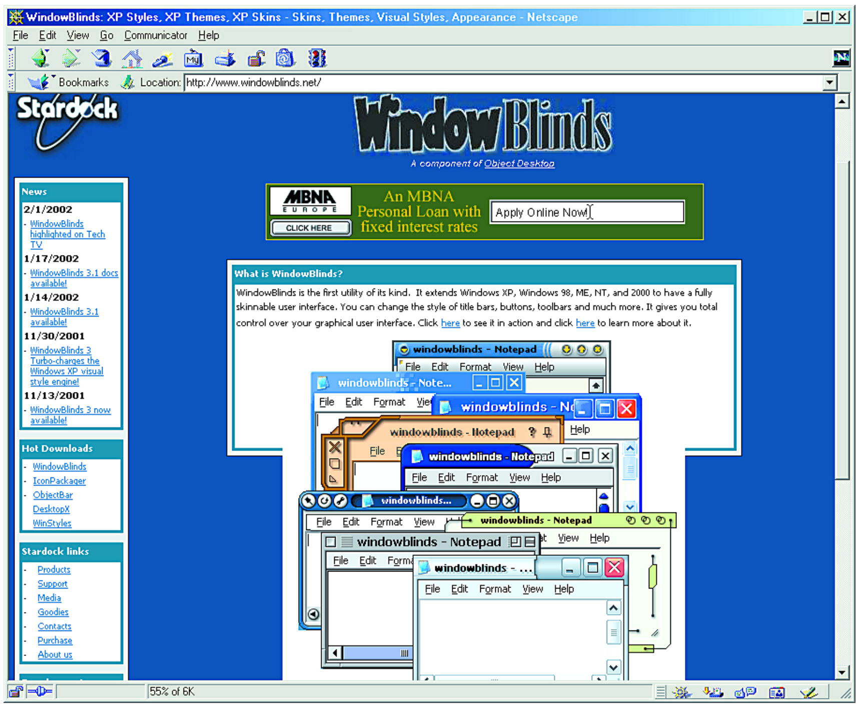
Of course, we should not forget to install a matching screensaver. Without doubt, the finest is **Serenescreen** [6]. The fish swimming around in the aquarium are so lifelike you'll have to suppress a tendency to start cleaning the inside of your CRT every week. The full program costs just $20 but the demo version (with a limited number of fish) is too good to miss.

Unfortunately, all of the above 'ersatz' does not result in a perfect imitation of the XP desktop. Where, you may ask, are the extended Start menu, the restyled Taskbar or the windows with rounded corners? To get all this, you'll need a bigger gun like **Windowblinds** [7] from the Stardock company. This program allows you to restyle almost all of Window's appearance. It is even possible to transform your PC into a Macintosh computer, complete with function buttons moved to the appropriate locations in the windows (just as with a real Mac). Windowblinds is not a low-cost program. Fortunately, a demo version is available which is certainly worth trying. Although Windowblinds comes with two so-called 'skins', many more of these are available on the net — just have a look at **Skinz** and **WinCustomize** [8].

The same company also supplies a dedicated program, Objectbar [9] for the Start Menu.

After all this surfing and downloading, your version of Windows will look totally different and computer illiterates and colleagues may even think you've the very latest OS running on your PC. Without admirers around, you may like the new look of your desktop so much you're won over to buying the real thing after all.

(025026-1)


WindowBlinds: XP Styles, XP Themes, XP Skins - Skins, Themes, Visual Styles, Appearance - Netscape
File Edit View Go Communicator Help
Bookmarks Location: http://www.windowblinds.net/
Stardock
WindowBlinds
A component of Object Desktop
MBNA EUROPE
CLICK HERE
An MBNA Personal Loan with fixed interest rates
Apply Online Now!
Hot Downloads
WindowBlinds
IconPackager
ObjectBar
DesktopX
WinStyles
Stardock links
Products
Support
Media
Goodies
Contacts
Purchase
About us
windowblinds - Notepad
File Edit Format View Help
55% of 6K

News

2/1/2002
- WindowBlinds highlighted on Tech TV

1/17/2002
- WindowBlinds 3.1 docs available!

1/14/2002
- WindowBlinds 3.1 available!

11/30/2001
- WindowBlinds 3 Turbo-charges the Windows XP visual style engine!

11/13/2001
- WindowBlinds 3 now available!

What is WindowBlinds?

WindowBlinds is the first utility of its kind. It extends Windows XP, Windows 98, ME, NT, and 2000 to have a fully skinnable user interface. You can change the style of title bars, buttons, toolbars and much more. It gives you total control over your graphical user interface. Click here to see it in action and click here to learn more about it.

Internet addresses

[1] XP Icons:
http://camtech2000.net/Pages/XPIcons.html

[2] XP Icon Raider:
www.skylarkutilities.com/program.pcs?xp-icon-raider

[3] Windows eXperience:
www.isenhower.com/XP/wallpaper.htm

[4] Win XP Wallpaper:
http://sardaukar.planetarrakis.net/wallpaper.htm

[5] Windows XP Theme 2 van Peter Wilcox:
http://pages.prodigy.net/i-pedro2/

[6] SereneScreen:
www.serenescreen.com

[7] Windowblinds:
www.stardock.com/products/windowblinds
www.windowblinds.net/

[8] Skins voor Windowblinds:
www.skinz.org
www.wincustomize.com

[9] ObjectBar:
www.stardock.com/products/objectbar/download.htm