MAY 2002
£3.20

ELEKTOR ELECTRONICS

THE ELECTRONICS & COMPUTER MAGAZINE

www.elektor-electronics.co.uk

MOBILE PHONE SNIFFER

who's talking now?



BasicCard for GSM Phones



DMX Dimmer

WorldSpace Radio

EEPROM Eraser

RS232-to-RS485

BASCOM-51

9 770268 451098 05

BasicCard for GSM Phones

discover the secrets about your GSM phone

Design by Patrick Gueulle

gueulle.patrick@laposte.net

A programmed BasicCard is used as the starting point in this article. We show how this 'SIM card emulator' can be adapted to test the most important functions that are available from the normal menu and the service menu of certain GSM phones.



The SIM (Subscriber Identification Module) card is the indispensable key for the use of every GSM phone. It contains both secret codes as well as personal details of the user and is supplied to the customer by the network operator, who usually retains the ownership rights.

A Smartcard using an open Operating System, such as the BasicCard, can be modified into an experimental SIM card, which allows a large number of experiments to be done independently from the network operators.

The 'professional' BasicCard

Since the introduction of the 'compact' BasicCard in 1998, the German company ZeitControl (www.zeitcontrol.de) have developed more powerful versions such as the 'enhanced' and the recently introduced 'professional' version. The ZC 4.1 has reached a new milestone, because this is the first BasicCard that supports the T=0 protocol. All previous versions of the BasicCard would work solely with the T=1 protocol, which is widely used in Germany.

Because the GSM 11.11 specification by ETSI requires the use of the T=0-protocol, it wasn't possible up until recently to use the BasicCard in a GSM phone.

The 'professional' BasicCard contains one of the most powerful ATMEL processors: the AT90SC323C. It has 32 KB of Flash memory for use by the operating system, 32 kB of EEPROM memory, 1 kB of RAM and a cryptographic coprocessor. This makes the chip very suitable for use as a 'Phase 2' SIM card and a 'SIM Toolkit'.

The most remarkable property of this chip is the fact that it has been wholly designed around Flash EEP-

Table 1.
Overview of the standard GSM 11.11 instructions.

```
20h : verify CHV
24h : change CHV
26h : disable CHV
28h : enable CHV
C0h : get response
32h : increase
04h : invalidate
B0h : read binary
B2h : read record
44h : rehabilitate
88h : run GSM algorithm
A2h : seek
A4h : select
FAh : sleep
F2h : status
2Ch : unblock CHV
D6h : update binary
DCh : update record
10h : terminal profile
C2h : envelope
12h : fetch
14h : terminal response
```

ROM technology. The on-chip operating system is loaded into memory during the activation of the card and is no longer mask programmed into the ROM area of the card. This gives the card much greater flexibility in use.

In a conventional SIM card the operating system is stored in ROM and can therefore not be modified.

The EEPROM contains an operating system that complies with the GSM 11.11 specification and (for 'Phase 2+' SIM cards) possibly some SIM Toolkit applications (usually Java applets), which have been added by the network operator or other institutions (such as banks).

The BasicCard has an open operating system, just like Javacards, Multos and other 'Windows for Smartcards'. In this case it is an interpreter that makes it possible to write applications in the ever-popular BASIC language, although this version has been specially adapted for use with these Smartcards.

When we've reached the point where the applications have to be transferred into the EEPROM, this can be done using a powerful, but still free, development tool (www.basiccard.com).

The BASIC program of nearly 450 lines used for this project takes up about 17% of the EEPROM memory, with the remainder being shared between the GSM 11.11 compatible files and a large log file, where the card can keep a record of all external instructions received.

The SIM card emulator

It certainly isn't the intention that the SIM card emulator is used to make telephone calls. Its main use is to make a number of functions accessible, which can usually not be realized with a card from the network operator.

In a real SIM card most of the sensitive information is protected against being overwritten and sometimes even against being read, through the use of administrator codes that are only known to the network operator.

In the scope of this project all confidential information (including the PIN code of the subscriber) is permanently deactivated, leaving us with complete access to the existing data.

If you think that it is easy to copy the information from a valid SIM card to make a usable clone, you would be very wrong. There are two protective measures in our project that make it impossible to use it for fraudulent purposes:

– To start with, the (A3/A8) coding algorithm that guarantees the validity of SIM cards on the GSM network is completely bogus in this card and can't be modified.
– You can try to make a copy of a working card, but the secret code is stored in a part of the card that can absolutely not be read.

One of the niceties of this card is that it provides access to hidden menus in some phones (especially so in Motorola phones), which are otherwise only accessible to authorised technicians.

The card is not suitable to recover lost security codes or to activate functions that are concerned with the cell information of transmitters of the various networks in a particular area.

There is enough EEPROM memory in the card to store all instructions sent to it by a phone or another terminal in an internal file. (Another terminal could be a PC running a program for accessing SIM cards). When this log file is subsequently transferred to the PC, it can be analysed for instructive or diagnostic reasons.

Finally, our card is also 'Proactive SIM' compatible, which means that it (as opposed to simple Phase 1 or 2 SIM cards) not only deals with the instructions sent to it by the phone, but can also initiate actions in the phone, particularly concerning the display and sounder.

The card has a small self-contained program, which only works in Phase 2 compatible phones. After about a minute this returns a hexadecimal string on the display (the 'Terminal Profile'), which after decoding provides a detailed list of the capabilities of the 'SIM Toolkit' of the phone. Next, the card starts to interfere with the workings of the phone (without disrupting the normal operation however), momentarily showing 'BasicCard' on the display once per minute, whilst playing a melody.

The card in detail

The SIM card emulator recognises all the standard instructions that can currently be sent by GSM phones (see **Table 1**) for Phase 1, 2 and 2+ standards. 'Recognises' does not necessarily mean 'processes', especially where instructions are concerned that access personal details. Even when the card ignores an instruction completely, it will return a successful reply to keep the GSM session from stalling.

For example, when you activate the PIN code function, the card will respond with the generic success code (9000h), but then won't request the PIN code. When we attempt to change the PIN code the card will also reply that the operation was carried out successfully, but any PIN code will be accepted as the correct one.

This trick makes it possible to play with the contents of the card without restraint. Before we get to that stage we should explain the structure of GSM-instructions a bit more, so that we can interpret the contents of the log file.

For SIM cards working with the T=0-protocol, all instructions destined for the card begin with a five-byte header: CLA INS P1 P2 LEN.
CLA is the ISO class of the instruction, which for a SIM card always has the value A0h.
INS is an operational code of which some values are shown in **Table 1**.
P1 and P2 are two parameters whose contents depend on the operational code used, but which usually have a default value of 00h.
LEN holds the length of the data block that

follows, or 00h if there is no data block.

There is a distinction between incoming instructions (from phone to card; the data block is sent after LEN) and outgoing instructions (from card to phone; the card is requested to send LEN bytes of data). In nearly all cases the card returns a minimum of two bytes, the Status Code (SW1 and SW2, which are 90h and 00h respectively on a successful completion).

Should the card wish to return data in reply to an incoming instruction, it answers with SW1=9Fh, with SW2 containing the number of bytes that should be requested with the instruction 'GET RESPONSE' (A0 C0 00 00 SW2).

The T=0-protocol has the property that it does not support the simultaneous transmission of incoming and outgoing instructions, which is possible with the T=1 protocol. This property makes it much simpler to use.

The most often used instruction is undoubtedly 'SELECT'. This is used to navigate through the directory structure, in a similar way to the old MSDOS instruction, 'CD' (Change Directory).

The root directory (address 3F00h) contains the following subdirectories:
7F10h (Telecom)
7F20h (GSM)
7F21h (DCS, a copy of the GSM directory, which is used by 1800 MHz phones).

To change to the 'Telecom' directory from the root directory, the following instruction should be issued:
A0 A4 00 00 02 7F 10.

Table 2 gives an overview of the files that are in the various directories on our card. Some of the files are vital to the operation of a GSM session; the other files can be used for interesting experiments.

Once a certain file has been selected its contents can be processed using read instructions (*read binary, read record...*) or write instructions (*update binary, update record...*).

In order to use the card to its full potential you would obviously have to know the function of all the files on the card. Although the only official description of the GSM specifications (www.etsi.org), and in particular the section on GSM 11.11, has very detailed explanations, we will limit ourselves to just the essential parts.

We will concentrate primarily on the GSM (or DCS) directory, which has the most potential for informative experiments.

The Phase and SST (SIM Service Table) files play an important role in the extent to which the phone returns information regarding the capabilities of the card (in reply to 'Terminal Profile' that is sent by some phones).

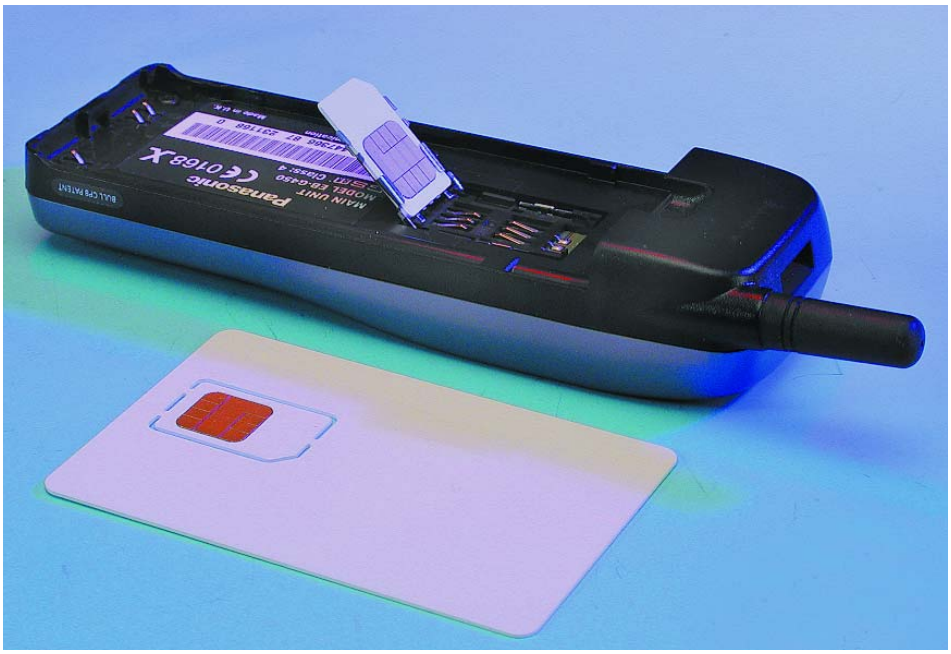
The 'Phase' file consists of a single byte, with a value of 02h for Phase 2 or 03h for Phase 2+, and is usually nonexistent in SIM Phase 1 cards. In contrast, the SST file will be larger for SIM cards offering more capabilities. Every service that is supported by the card is realised via two bits, which indicate whether or not the service in question is present and whether or not it is active (see **Table 3**). Normally, only the operator is able to modify the SST (for example when you subscribe to an extra service). In our case the SST is not write-protected, but you should

Table 2.
The files on our card.

Root directory:
```
2FE2 (ICCID)
```

Telecom directory:
```
6F3A (ADN, Abbreviated Dialling Numbers)
6F3B (FDN, Fixed Dialling Numbers)
6F3C (SMS, Short Messages)
6F40 (MSISDN, Own Numbers)
6F42 (SMSP, Short Messages Service Parameters)
6F43 (SMSS, SMS Status)
6F44 (LND, Last Number Dialed)
```

GSM or DCS directory:
```
6F05 (LP, Language preference)
6F07 (IMSI, International Mobile Subscriber Indentity)
6F20 (Kc, Ciphering Key)
6F30 (PLMN, Preferred PLMNs)
6F31 (HPLMN search period)
6F38 (SST, SIM Service Table)
6F3E (GID1, Group Identifier level 1)
6F3F (GID2, Group Identifier level 2)
6F74 (BCCH, Broadcast Control Channels)
6F78 (ACC, Access Control Class)
6F7B (FPLMN, Forbidden PLMNs)
6F7E (LOCI, Location Information)
6FAD (AD, Administrative Data)
6FAE (Phase)
```



refrain from writing random values into it! Its original contents (DF 30 C3 F3 00 00 00 03) are a mirror image of the capabilities offered by the selected files. Attempts to activate services on the card that it doesn't support are pointless. On the other hand, it **is** possible to deactivate one of the services by turning one of the original '1' bits into a '0'.

In the same way we can temporarily change phase 03h into phase 02h, to study the consequences.

ICCID contains the (not confidential) identification number of the card; IMSI identifies the subscriber via the network operator (the phone number, really...). IMSI also contains the code of the operator, allowing the possibility of 'roaming': When a network comes across an unfamiliar IMSI, it requests permission from the operator of the SIM card for the subscriber to use its services. If granted, the logo of this network will appear on the display. In our case the original network code is 001-01, a value for a fictional network and reserved for our SIM test card.

The AD file is also programmed with a typical test value. The advantage of this is that the card won't be rejected by a phone which has a 'simlock' added by the service provider.

The GID1 and GID2 files have similar roles. Some phones will only accept SIM cards containing a certain value in these, normally write-protected, files.

The contents of LOCI become apparent when you manually try to connect the phone to a network. The attempt is bound to fail because the identification inside the SIM card is fictional. In this file we can find the code of the last operator with which contact was attempted, as well as some details regarding the place of the attempt and the reason for the failure of the connection. At the same time, the code of every network that rejected the SIM card is stored in the file FPLMN, up to a maximum of four. The list can be cleared by setting all bytes in FPLMN to FFh.

Most of the files in the Telecom directory are concerned with personal details of the subscriber, such as lists with phone numbers and SMS messages. The menu of virtually every phone offers the opportunity to modify the contents of these files, but there are also programs for the PC that are much easier to use.

Table 3.
The services in the SIM Service Table.

Byte	Active	Present	Service	GSM
	b2	b1	Service	1: CHV (PIN)1 disable function
1	b4	b3	Service	2: Abbreviated Dialling Numbers (ADN)
	b6	b5	Service	3: Fixed Dialling Numbers (FDN)
	b8	b7	Service	4: Short Message Storage (SMS)
	b2	b1	Service	5: Advice of Charge (AoC)
2	b4	b3	Service	6: Capability Config. Parameters (CCP)
	b6	b5	Service	7: PLMN Selector
	b8	b7	Service	8: Party Subaddress
	b2	b1	Service	9: MSISDN
3	b4	b3	Service	10: Extension 1
	b6	b5	Service	11: Extension 2
	b8	b7	Service	12: SMS Parameters
	b2	b1	Service	13: Last Number Dialled (LND)
4	b4	b3	Service	14: Cell Broadcast Message Identifier
	b6	b5	Service	15: Group Identifier Level 1
	b8	b7	Service	16: Group Identifier Level 2
	b2	b1	Service	17: Service Provider Name
5	b4	b3	Service	18: Service Dialling Numbers (SDN)
	b6	b7	Service	19: Extension 3
	b8	b7	Service	20: RFU
	b2	b1	Service	21: VCGS Group Identifier List (EF VGCS and EF VGCSS)
6	b4	b3	Service	22: VBS Group Identifier List (EF VBS and EF VBSS)
	b6	b5	Service	23: Enhanced Multi-Level Precedence & Pre-emption Service
	b8	b7	Service	24: Automatic Answer for eMLPP
	b2	b1	Service	25: Data download via SMS-CB
7	b4	b3	Service	26: Data download via SMS-PP
	b6	b5	Service	27: Menu selection
	b8	b7	Service	28: Call control
	b2	b1	Service	29: Proactive SIM
8	b4	b3	Service	30: Cell Broadcast Message Identifier Ranges
	b6	b5	Service	31: Barred Dialling Numbers (BDN)
	b8	b7	Service	32: Extension 4
	b2	b1	Service	33: De-personalization Control Keys
9	b4	b3	Service	34: Co-operative Network List
			RFU = Reserved for Future Use	

Utilities

In principle any asynchronous card reader could be used to modify the contents of our cards via the PC's keyboard, using the active low ISO7816 instructions. For example, in order to write 02h to the 'Phase' byte, the following instructions would have to be executed:

```
A0 A4 00 00 02 3F 00
A0 A4 00 00 01 7F 20
A0 A4 00 00 02 6F AE
A0 D6 00 00 01 02
```

There are also programs for the management of SIM cards that are much easier to use, thanks to a built-in editor. One of the best examples of this is SIMSurf Profi, which is included with certain versions of ChipDrive card readers by Towitoko (www.towitoko.de).

These readers are very popular in Europe and can be set up to work in either PC or SC mode (by installing its drivers). They are also

Table 4. Explanation of the 'Terminal Profile' structure.

Byte 1:
```
Bit 1 : Profile download
Bit 2 : SMS-PP data download
Bit 3 : Cell Broadcast data download
Bit 4 : Menu selection
```

Byte 2:
```
Bit 1 : Command result
Bit 2 : Call Control by SIM
```

Byte 3 (Proactive SIM):
```
Bit 1 : Display Text
Bit 2 : Get Inkey
Bit 3 : Get Input
Bit 4 : More Time
Bit 5 : Play Tone
Bit 6 : Poll Interval
Bit 7 : Polling Off
Bit 8 : Refresh
```

Byte 4 (Proactive SIM):
```
Bit 1 : Select Item
Bit 2 : Send Short message
Bit 3 : Send SS
Bit 4 : Send USSD
Bit 5 : Setup Call
Bit 6 : Setup Menu
Bit 7 : Provide Local Information
```

compatible with the utility program UTILPCSC.EXE, which is required for reading the LOG file of our experimental card. The programs UTIL1.EXE and UTIL2.EXE are meant to be used with the CyberMouse card readers (the ACR20S or the ACR30S from ACS), which are included with the BasicCard kits (www.basiccard.com).

The various utilities mentioned here, as well as the TP.IMG file that has to be stored on the ZC4.1 SIM card, can be downloaded from the *Elektor Electronics* website (www.elektor-electronics.co.uk). They can also be ordered from *Elektor Electronics* (on floppy disk, order number **010138-11**).

With a special driver (found at www.acs.com.hk) the program UTILPCSC.EXE can be used. This program, which like the SIM card has been developed in ZCBasic V4, provides four fundamental functions for use with the LOG file: Activate, deactivate, download and clear. In general, the LOG file is activated *just* before insertion of the SIM card in a GSM phone and deactivated immediately after its removal, in order to prevent the overflow of data on the card.

We download the file (which is stored on the hard drive with the name CARD.LOG) and clear the contents of the card to make room for new information. CARD.LOG is an ASCII file that contains hexadecimal values. This can be viewed using any editor or word processor. Every instruction received by the card is put on a new line, while a blank line indicates that the card was reset at that point (for example, by switching the phone off, and then on again).

The information shown below was captured when a Phase 2+ compatible phone was activated. The 'Terminal Profile' instruction at the end of the file is used by the phone to inform the card of its capabilities.

```
A0 A4 00 00 02 7F 20
A0 C0 00 00 16
A0 A4 00 00 02 6F AE
A0 C0 00 00 0F
A0 B0 00 00 01
A0 A4 00 00 02 6F 05
A0 C0 00 00 0F
A0 B0 00 00 03
A0 A4 00 00 02 6F 05
A0 C0 00 00 0F
A0 A4 00 00 02 7F 20
A0 C0 00 00 16
A0 A4 00 00 02 6F AE
A0 C0 00 00 0F
A0 B0 00 00 01
A0 10 00 00 04 0F 03 FF F7
```

The hexadecimal string 0F 03 FF F7 appears on the display of the mobile after the initialisation procedure (those long delays which are often followed by the message 'try again later' when you try to execute a function that requires too many system resources).

A quick glance at the information in **Table 4** (based on the GSM 11.14 specification) tells us that this phone possesses all SIM Toolkit functions, apart from the 'Send USSD' function. Furthermore, we can tell that this device is incompatible with several advanced functions (sending SMS and email) and that this is a particular international prepaid SIM card, the EasyRoam GSM card from Swisscom (www.easy-roam.com). Every phone model will give different results, which can also vary depending on the location or country.

(010138-1)



Editor's note:
Unfortunately the BasicCard used for this project has a relatively high price: nearly 10 US$ per card with a minimum order quantity of ten pieces. For readers who are interested in the card we would suggest that they band together and place a combined order with the supplier.

Preparing the card

Although this project was developed using a BasicCard kit V4.12, you don't need the complete kit to build and use the SIM card emulator.

In practice it is sufficient to have the following items:

– the program BCLOAD.EXE, which is part of the free development kit that can be downloaded from www.basiccard.com.
– a compatible Smartcard reader, such as the CyberMouse that is included with the BasicCard kit, or another PC/SC reader
– a blank BasicCard that can accept our TP.IMG program file (Currently that is a ZC4.1 RSA [28/09/2001])

When the Smartcard reader has been installed according to the manufacturer's instructions, you should open an MS-DOS window and change to the directory that contains the files TP.IMG and BCLOAD.EXE.

One of the following commands should now be typed:

– BCLOAD -D -P1 TP.IMG when using a CyberMouse on COM1,
– BCLOAD -D -P2 TP.IMG when using a CyberMouse on COM2,
– BCLOAD -D -P101 TP.IMG when using a PC/SC compatible reader, regardless of the port that it is connected to.

When the reader has been recognised a request appears on the screen to insert a blank card (**SCreendump A**). Should the file turn out to be incompatible with the card, the program BCLOAD will issue a warning (**Screendump B**), otherwise the progress is indicated on the screen by a series of addresses as they are programmed. Wait until the programming has completed before removing the card (**Screendump C**).

The card should now be initialised with the 'Clear LOG file' function of our program. The first time round this can take quite some time (**Screendump D**)!

Three different programs are available:
– UTIL1.EXE for use with a CyberMouse on COM1
– UTIL2.EXE for use with a CyberMouse on COM2
– UTILPCSC.EXE for use with a PC/SC reader

The only thing left to do is to cut the SIM card to size, after which it can be inserted into a GSM phone.

Figure 1 shows all dimensions, in case that the BasicCard has not been pre-punched (this depends on the version). If enough care is taken during the removal of the chip, it will be possible to re-insert it back into the card. It can then be kept in place with some sticky tape. The card can then be put back into the reader for the extraction of the next set of information provided by the phone.

For that matter, there are adapters for sale that take SIM cards and change them to Smartcard dimensions, allowing them to be easily inserted into the card reader on the PC.

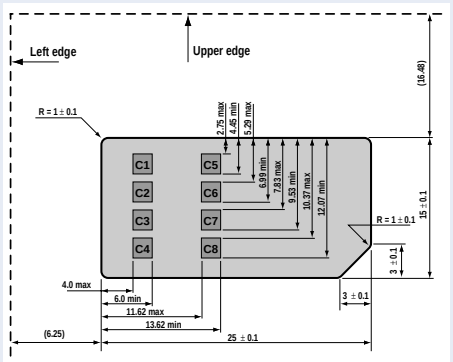
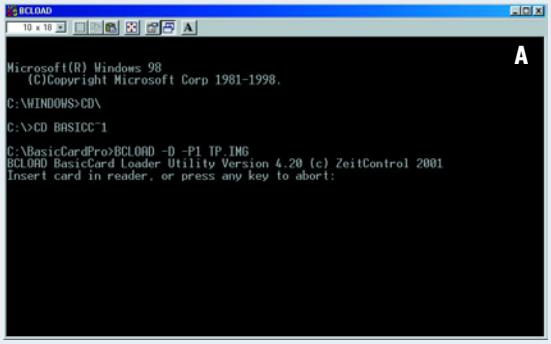
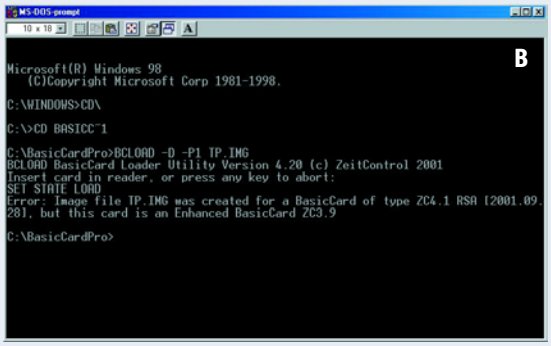
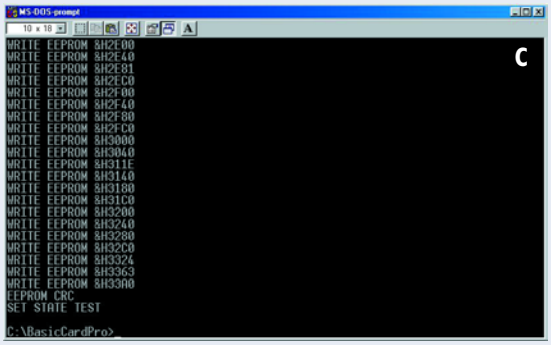


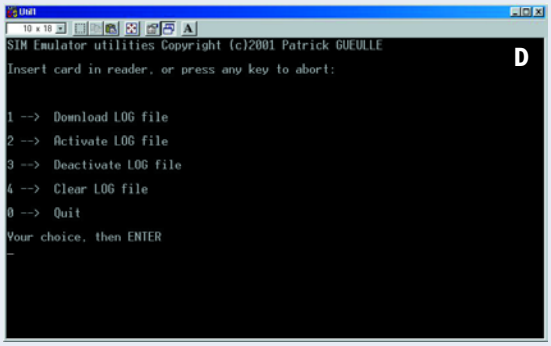
Figure 1. Dimensions of a SIM card.

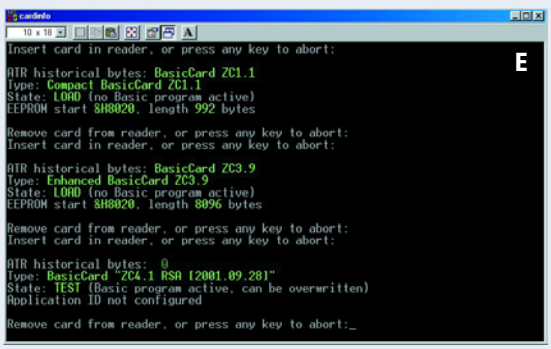
The programs that have been installed in the BasicCard directory as part of the Development Environment offer many other possibilities. For example, we can obtain further details of the cards that are inserted into the reader (**Screendump E**). We hope that you will enjoy experimenting with this powerful tool.











BASCOM-51

a new BASIC compiler for the Elektor 89S8252 Flash Board

By B. Kainka

Till now, the BASIC-52 interpreter was the only choice if you want to run BASIC on the extremely popular Elektor Flash Microcontroller Board (described in the December 2001 issue). Fans of the BASIC programming language will like to hear that there is now a powerful alternative called BASCOM-51.

BASCOM was originally developed by Mark Alberts and his Dutch company MCS-Electronics (www.mcselec.com) for the AVR series of microcontrollers from Atmel. Later, a spin-off called BASCOM-LT was marketed specially for the smaller controller type 89C2051. This eventually led to BASCOM-51 being developed and sold for the beefed up derivates of the 8051. The MCS-Electronics website has available a free download of the BASCOM-51 demo version. This version allows programs with a size of up to 2 kBytes to be developed. This may not seem a lot but don't be fooled because the compiler is very effective at generating code! The free demo version was tested in combination with the *Elektor Electronics* 89S8252 Flash Microcontroller Board (December 2001) with excellent results.

The compiler is marked by a large number of important settings. Under Options/Compiler/Output you can select the desired output format. Here, Hexfile is the format to go for because it can be read by a program like MicroFlash or AtmelISP. From there, executable code may be copied into the controller. The Debug file is essential for the built-in Simulator. The Error file is used to draw your attention to errors that occurred during the compilation process.

Another important setting is found under Options/Compiler/Output/Misc. which is where you can select the Register file, in this case, the file `89s8252.dat`. Once this has been selected, the compiler will recognize all special function registers (SFRs) inside the controller.

A first test may be performed using a small program for fast bit outputting via the ports. The first thing you'll notice is the absence of line numbers, which is evidence of structured BASIC being implemented. Instead of `GOTO` and line numbers, instructions like `Do` and `While` are employed. In this way, BASCOM follows a trend in that higher-level programming languages seem to develop increasing 'common ground'.

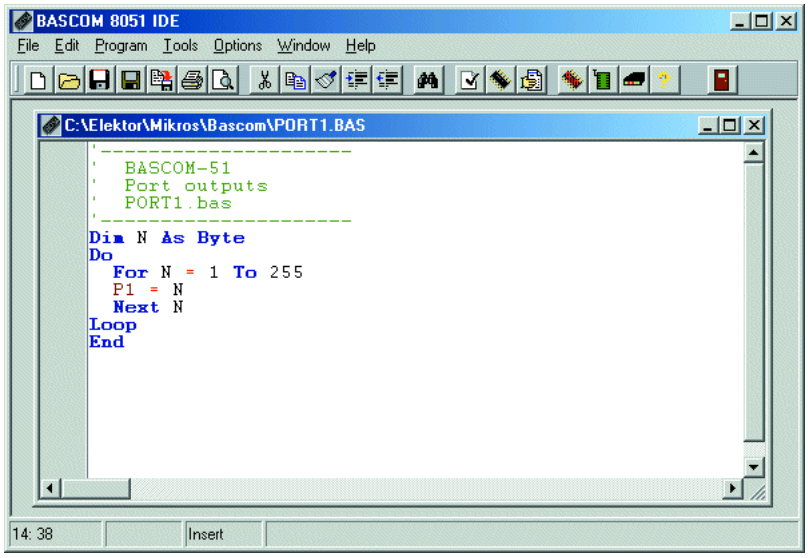


Figure 1. The programming environment.

The second pleasant thing to notice is that all variables have to be declared with their desired type. This allows you, for instance, to work with individual bytes and so approach the results offered by an assembler. Moreover, you can explicitly state whether or not variables are allowed in the internal RAM. BASCOM-51 can make do without external memory, which is not self-evident for a compiler.

```
'------------
'  BASCOM-51
'  Port outputs
'  PORT1.bas
'------------
Dim N As Byte
Do
  For N = 1 To 255
  P1 = N
  Next N
Loop
End
```

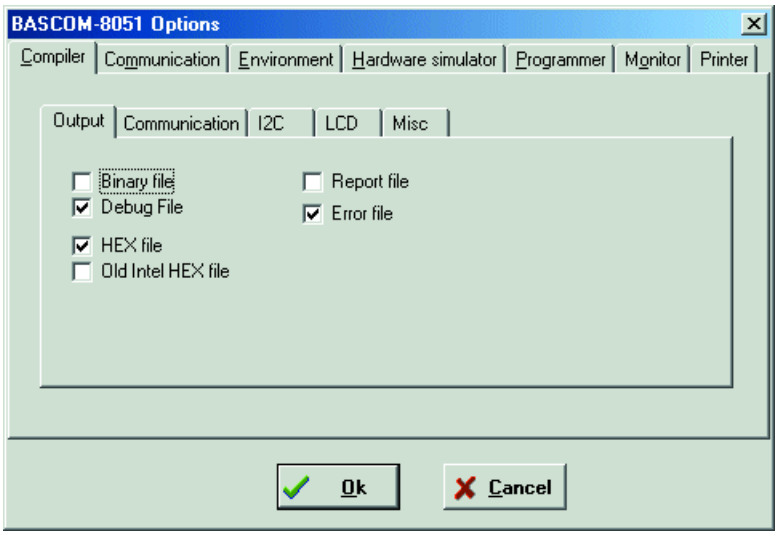


Figure 2. Compiler settings.

As with any new higher programming language, the main question is: 'so how fast is the code'? The answer is given by an oscilloscope hooked up to port line P1.1. The result: only 30 µs for each new output action. Admittedly, assembler is about 10 times faster at just over 3 µs, but then the READS-51 C Compiler requires 125 µs for the same task, not even mentioning BASIC-52, which takes no less than 2,500 µs. This makes BASCOM-51 a clear winner in its class of compilers and interpreters. Compared with BASIC-52, there's another advantage in that the code generated by BASCOM-51 can be made self-starting without problems.

The new compiler offers different serial interfaces. The commands `PRINT` and `INPUT` convey data via the UART inside the microcontroller. The baudrate may be found among the settings under Options/Compiler/Communication.

```
'------------
'  BASCOM-51
'  Port inputs and outputs
'  RS232.bas
'------------
Dim N As Byte

While 1 = 1
  Input N
  P1 = N
  N = P1
  Print N
Wend
```

Besides the hardware UART, you have a software UART so that a number of interfaces are available, also when using the Elektor 89S8252 Flash Microcontroller Board. The next program listing demonstrates the use of the software UART. For a first test, the same pins are used as with the processor's on-chip UART. For the software UART, individual bytes are processed using the commands `GET` and `PUT`.

```
'------------
' BASCOM-51
' Port inputs and outputs
' RS232_2.bas software uart
'------------
Dim N As Byte

Open "com3.1:9600" For Output
As #1
Open "com3.0:9600" For Input As #2

While 1 = 1
  Get #2 , N
  P1 = N
  N = P1
  Put #1 , N
Wend

Close #1
Close #2
```

BASCOM-51 not only processes byte-ports, but also individual bits. The following example program divides the signal frequency applied to port P1.0 and outputs it frequency-halved on port P1.1. This is done by toggling the P1.1 bit on every falling edge of the input signal. In this way, the operation of the program may be compared to that of a binary counter. The maximum input frequency will be about 50 Hz.

```
'------------
'  BASCOM-51
'  Port input freq. divider
'  Toggle.bas
'------------

While 1 = 1
  While P1.0 = 1
  Wend
  P1.1 = Not P1.1
  While P1.0 = 0
  Wend
Wend
```

If necessary, chunks of assembler code may be inserted. The example below demonstrates how fast pulses spaced 10 ms apart are generated on port P1. The longer delay is of course generated in BASIC, the time-critical parts, in assembler.

```
'------------
' BASCOM-51
' Ass. example Assem.bas
' mixed assembler and Basic
'------------
Print "pulses"

While 1 = 1
$asm
   mov A , #00
   mov R1,#100
L1:
   mov P1,A
   inc a
   djnz R1,L1
```

Table 1. Compiler/Assembler comparisons

language	Memory req.	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
BASCOM-51	< 2 K ROM	30 µs	yes

```
$end Asm
Waitms 10
Wend
End
```

BASCOM-51 has far more to offer than can be described in this short article. For example, a number of different interfaces for special peripherals are supported: LC displays in 4-bit and 8-bit mode, I²C devices, the 1-Wire bus, and the software interface discussed above. Many tasks are considerably simplified in this way. We would therefore suggest that every user of 8051 or AVR microcontrollers takes a good look at BASCOM-51.

(021003-1)

BASCOM distributors in the UK

Kanda Systems Ltd.
Units 17/18
Glanyrafon Enterprise Park
Aberystwyth
Ceredigion SY23 3JQ.
Tel. (+44) (0)1970 621030
Email: *sales@kanda.com*
Website: *www.kanda.com*

Quasar Electronics
Mr. Simon Neil
Unit 14 Sunningdale
Bishops Stortford
Herts CM23 2PA.
Tel. (+44) (0)1279 306504
Fax: (+44) (0)870 7064222
Email: *simon@quasarelectronics.com*
Website: *www.quasarelectronics.com*

A list of worldwide distributors for BASCOM-51 may be found on the 'Resellers' page at *www.mcselec.com*

Squelch Add-On

for VHF Airband Receiver

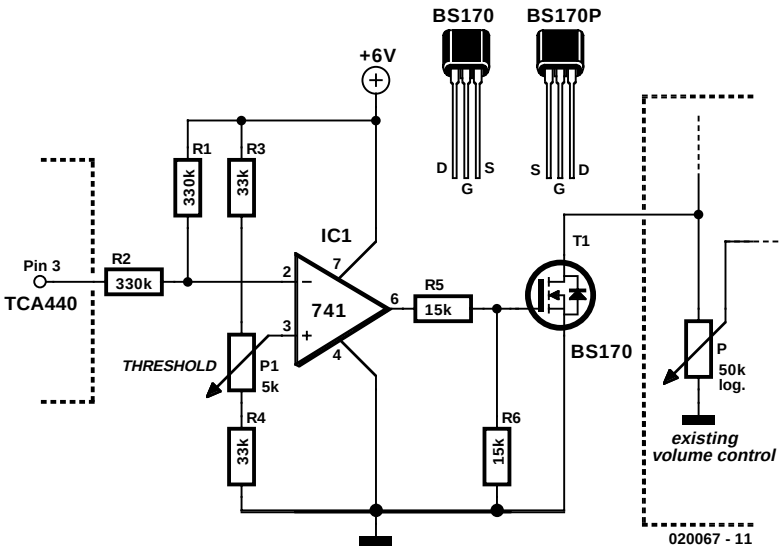
Design by G. Baars

Recent feedback from our readers indicated that the VHF Airband Receiver described in our March 2002 issue has received a warm welcome. However, some of you have expressed some concern about the absence of noise suppression under no-signal conditions. Well, here is the circuit you need.

Silence, please!

Squelch is a system used to prevent weak, unintelligible signals and random noise (produced by the demodulator and receiver RF and IF amplifiers) from being heard by a radio operator while still allowing intelligible signals to be heard normally. This is accomplished by the use of an adjustable threshold below which any received signals are ignored. Only signals whose strength is above the squelch level are allowed to reach the receiver's audio circuitry, with the result that only satisfactory signals are received, and that no annoying hiss is heard when no signal is being picked up.

The word 'squelch' literally means 'to silence' but there is also an obvious link to the second meaning of the verb, i.e., 'make a sucking sound as of treading in thick mud'. You will understand the second meaning once you've heard the so-called squelch trailer, which is short burst of receiver noise that's produced just before a squelch circuit closes. This trailer is more conspicuous with FM than AM receivers because of the higher output noise level produced by the demodulator.



How it works

The circuit shown here is an optional extension to the VHF Airband Receiver. All signals and supply voltages are taken from the receiver.

Opamp IC1 acts as a simple comparator whose output remains high as long as the AGC level measured at pin 3 of the TCA440 (in the receiver) is below a certain threshold set with pot P1. FET T1 then effectively short-circuits the audio signal to ground and we hear nothing.

When the AGC level exceeds the threshold (indicating reception of an intelligible signal), the '741 output drops low and the audio signal is allowed to reach the volume pot in the receiver.

The +6 V supply voltage for the squelch add-on is obtained from the receiver. The connection between the drain of the BS170 and the 'top' of the volume pot should be made in screened cable.

(020067-1)

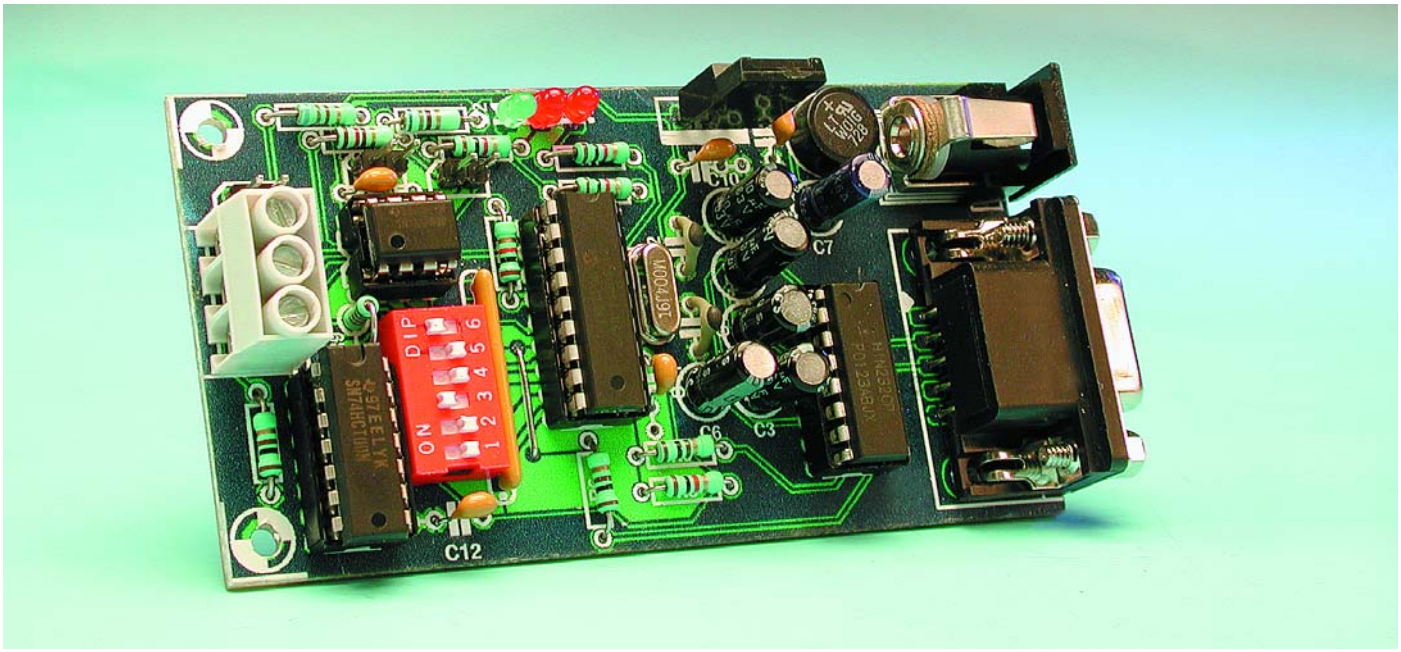
RS232-to-RS485 Half-Duplex Adapter

advantages of balanced lines

Design by E. Eugeni

bitlab@tin.it

This project has been designed to interface an EIA-232 channel to an EIA-485 half-duplex bus. We should do so because EIA-485 offers more features than 232, especially where we apply a long distance wired communication involving more than two devices.



The EIA-232 standard, formerly known as RS-232, was introduced about thirty years ago, well before the first application of desktop sized personal computers. The RS-232 standard mainly covers the communication between DTEs and DCEs, where DTEs are **Data Circuit Terminating Equipments**, and DCEs are **Data Communication Equipments**. A typical DTE could be just a video terminal, a printer, or even a personal computer — it can operate either as a source or destination of data, hence it is the equipment where the communication is **terminated**. A DCE, on the other hand, is a device that accepts and transfers data with the sole purpose of allowing a communication between two DTEs. A typical DCE could be a modem, or even a simple cable that links two DTEs. That's right, two DTEs, not more, because the EIA-232 standard does not allow more than one transmitter and one receiver per cable. EIA-485 lifts this restriction, since it allows up to 32 transmitters and receivers to be linked to the same electrical line, where 32 transmitters should be taken to mean that all share the same electrical path, but only one shall be active at a given time. To allow this mode of operation, EIA-485 drivers have an Enable pin to switch on the active circuit, but only when the data transfer has to take place. EIA-485 receivers do not need an Enable

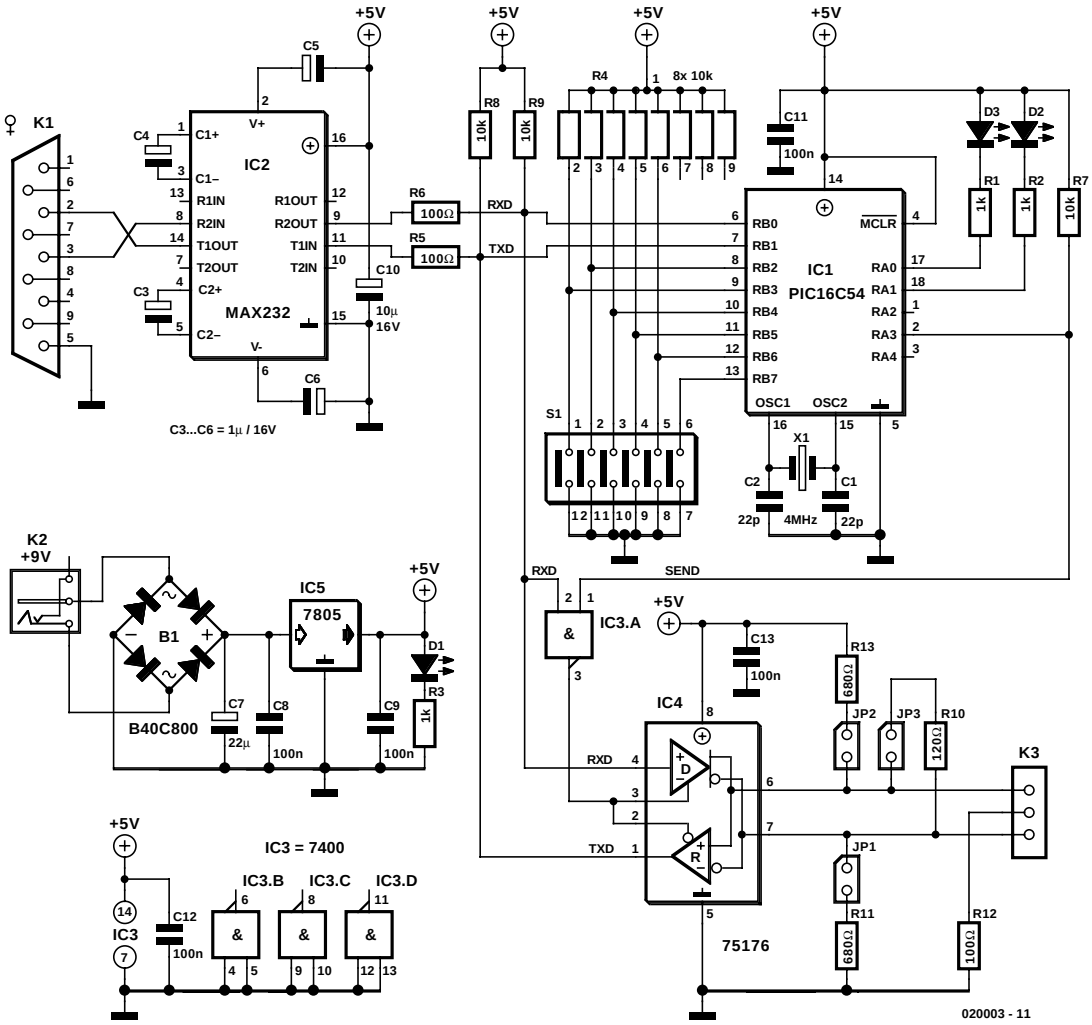


Figure 1. Circuit diagram of the EIA-232 to EIA-485 adapter. At the heart of the circuit we find a PIC microcontroller.

pin, since they can operate in an 'always on' fashion without disturbing the line. Nowadays, EIA-232 circuits can be found in many electronic devices capable of handling some kind of data communication through a cable. The best-known device is probably the trusty and ubiquitous personal computer (PC).

EIA-485 circuits were originally developed to serve industrial applications, but are gaining popularity elsewhere thanks to simple **adapters**, like the one described in this article. It can be built as a do it yourself project with minimal effort and expense.

The adapter function

An adapter may be likened to a simultaneous interpreter, who listens to one party and talks to another, where the parties speak different languages. A half-duplex adapter circuit has everything to accept and transform data from either side of a communications channel. However, the direction of the transfer has to be known beforehand and this is determined on the basis of the first signal which comes in. If the first signal originates from the EIA-232 side, the circuit promptly switches the 485 bus into transmission mode. If the first signal is from the EIA-485 side, no action is required because an EIA-232 channel can operate simultaneously both in transmission and reception mode. But what is, in practice, the real difference between the EIA-232 and EIA-485 communication standards?

Two standards side by side

The main difference between an EIA-232 channel and an EIA-485 bus is the number of electrical connections. EIA-232 employs at least three wires, one of which is the common ground, while EIA-485 requires only two conductors, and does not need a common reference wire. Moreover, EIA-232 employs one conductor to send, and the other to receive, while EIA-485 handles both directions, technically referred as 'A' and 'B', as a single bidirectional path. EIA-232 and EIA-485 data consists of a series of pulses or square waves, but there is a big difference in the voltage levels and polarity. This is brought about by the EIA-232 standard employing **asymmetrical** (unbalanced or single-sided) voltages to carry the zeros and ones of digital information, where asymmetrical means 'referred to a common ground'. A voltage level of between 3 and 12 volts below the ground level should be recognized as a logic High, while a signal of the same swing, but above the ground level, shall be recognized as a logic Low. The EIA-485 standard, by contrast, operates with **symmetrical** (balanced) signals, where symmetrical should be taken to mean 'not referred to a common ground'. In detail, data on a 485 line has to be measured only between 'A' and 'B', without any

Table Ia.

DIP switch settings (word length)

1 = on; 0 = off.

Number of bits	S1-1	S1-2
10	0	0
11	0	1
12	1	0
13	1	1

Table Ib.

DIP switch settings (EIA-232 baudrate)

1 = on; 0 = off.

Baudrate	S1-3	S1-4	S1-5	S1-6
19,200	0	0	0	0
57,600	0	0	0	1
300	0	0	1	0
19,200	0	0	1	1
4,800	0	1	0	0
110	0	1	0	1
300	0	1	1	0
19,200	0	1	1	1
57,600	1	0	0	0
115,200	1	0	0	1
600	1	0	1	0
19,200	1	0	1	1
2,400	1	1	0	0
19,200	1	1	0	1
38,400	1	1	1	0
19,200	1	1	1	1

reference to the ground. When 'B' is positive relative to 'A', a logic High is carried. Conversely, A>B means a logic Low. In theory, both signal lines 'A' and 'B' could be at any reasonable potential below or above ground, because the information is carried only by the voltage difference measured between them. In practice, when linking two EIA-485 devices, a third wire is added to create a common ground line. This is not required for the data transfer proper, just useful to maximize noise immunity, since the voltage level we use are in the range 0.2 V to 6 V. As with the '232 standard, where any voltage smaller than ±3 V is treated as undefined, with '485 we consider undefined a voltage level in the range of ±200 mV.

Circuit description

The schematic of the EIA232-to-485 adapter is given in **Figure 1**. There are four functional blocks: the EIA-232 interface around IC2 and some passive components; the EIA-485 bus interface based upon IC4; the control logic driven by microcontroller IC1; and finally the power supply regulator built around IC5. There are also three LEDs, D1, D2 and D3, and 6-way DIP switch block, S1, which is directly connected to some microcontroller port lines and serves to set up the adapter's operating parameters.

The EIA-232 device is hooked up to connector K1, observing the usual connection of pin 2 for sent data, pin 3 for received data, and pin 5 for signal ground. IC2, a MAX232, provides all the functions to convert the voltage levels of the EIA-232 standard (±3 to ±12V) to the usual TTL range of 0 to +5 V. For incoming data, IC2 provides clamping of the negative pulse levels, followed by a voltage reduction of the positive swing. Functionally, the MAX232 is an inverter.

For outbound data, where a +5 V swing must be converted to a bipolar voltage, the same MAX232 acts as a charge pump booster using external capacitors C3, C4, C5 and C6. The internal voltage step-up process takes place at very high frequency, and will inevitably generate some spurious components on the supply rails. To avoid disturbing the other ICs, especially the microcontroller, decoupling capacitor C10 is connected to pin 16 of the MAX232. With the same objective in mind, low-value series resistors R5 and R6 are inserted in the TTL in/out lines.

At the other side of the circuit, IC4 operates as half-duplex interface to the EIA-485 world (technically, this is referred to as a 'bus') which is connected to K3. IC4 does not use any voltage booster, since the signal levels we require for the external interface are unipolar and not referred to ground. The 75176 IC combines a balanced driver (with its inputs on pin 4 and outputs on pins 6 and 7), and a balanced receiver (with its inputs on pins 6 and 7 and output on pin 1). To create a complete EIA-485 interface we also need two pins for the power supply and a pair of Enable terminals to decide when to listen and when to talk, since in a half-duplex system these functions can not be performed simultaneously. To switch the EIA-485 bus to transmission mode, pin 3 is pulled high, while receive mode is selected by a Low on pin 2. By linking the pins together, data may be sent by a single High level, and received by imposing a single Low. In the real circuit, we drive both Enable lines from the output of IC3A, a NAND gate which combines a signal from pin RA3 of the PIC controller, and the (inverted) RxD signal from the EIA-232 channel. Using this simple and smart arrangement, the transmission mode is switched on in good synchronism, since the received '232 signal acts directly on the '485 driver enable terminal, together with the microcontroller. Actually, the intervention of IC1 is delayed because its senses the arrival of a Low level from RB0, executes some instructions, and then responds by pulling low RA3. The Low level from RB0 is used as the start bit in the '232 data stream, and as a trigger to the internal counting loop of the processor. This produces the following events: initially, the incoming '232 start

COMPONENTS LIST

Resistors:
R1,R2,R3 = 1kΩ
R4 = 10kΩ 8-way or 6-way SIL array
R5,R6 = 100Ω
R7,R8,R9 = 10kΩ
R10 = 120Ω
R11,R13 = 680Ω
R12 = 100Ω

Capacitors:
C1,C2 = 22pF
C3-C6 = 1µF 16V radial
C7 = 22µF 16V radial
C8,C9,C11,C12,C13 = 100nF
C10 = 10µF 16V radial

Semiconductors:
D1,D2,D3 = LED, red, low current
IC1 = PIC16C54-HS/P, programmed, order code **020003-41**
IC2 = MAX232CP
IC3 = 74HCT00
IC4 = 75176BP
IC5 = 7805
B1 = bridge rectifier B40C800, round case (40V piv, 800mA)

Miscellaneous:
JP1,JP2,JP3 = jumper
K1 = 9-way sub-D socket (female), angled pins, PCB mount
K2 = mains adaptor socket, PCB mount
K3 = 3-way PCB terminal block, lead pitch 5mm
S1 = 6-way DIP-switch
X1 = 4MHz quartz crystal

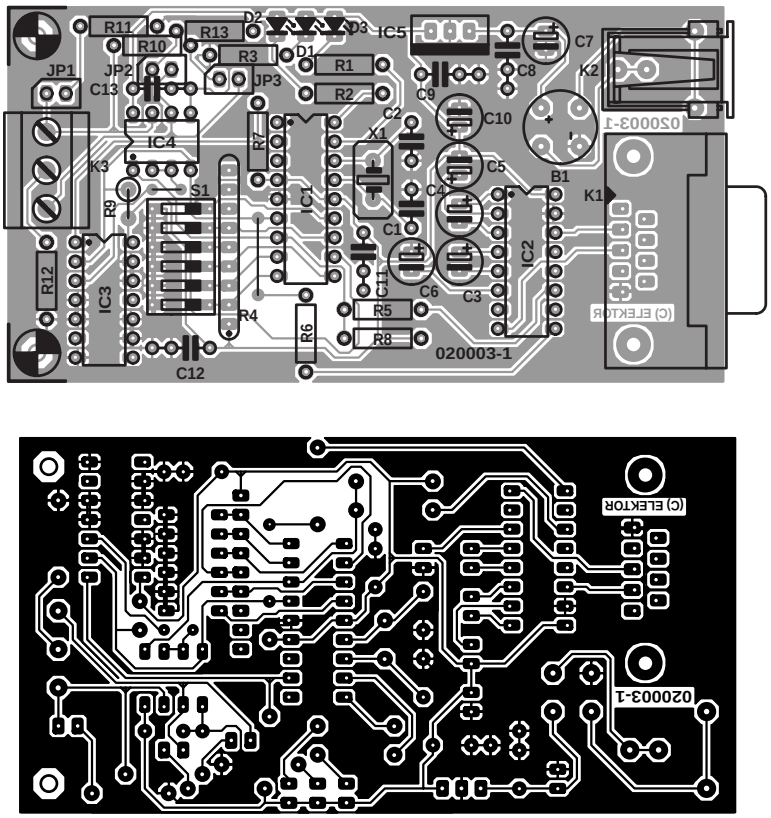


Figure 2. Copper track layout and component mounting plan of the adapter (board not available ready-made).

bit will force IC4 into transmission mode, thanks to the direct link to pin 2 of IC3A. After a few microseconds, the processor will sense the start bit at RB0 and will promptly confirm the transmission mode through RA3 and pin 1 of IC3A. The '485 bus driver will remain in transmission mode for a while, even when the '232 data stream returns to High level and repeatedly toggles to follow the information. The '485 driver will be reset to receive mode only when all the data bits have been translated and transferred from one side to the other.

But what about the duration of a single data bit? Well, the processor is informed about the speed of the data stream by the six switches on S1 that allow you to select a suitable software delay loop. An overview of the switch settings for different baud rates and word formats is given in **Table 1**.

R7 and R9 are simple static pull-ups. The first imposes the receive mode to IC4 during the power-up phase. The second pulls the '485 TTL incoming data line to the positive rail, to force the EIA-232 transmission channel in idle mode when the '485 bus is transmitting. R11, R13 and R10 can be switched in circuit by three jumpers JP1, JP2 and JP3. R11 and R13 are included for noise protection. When the '485 bus is in receive mode, but no device effectively 'talks' on the line, the 'A' and 'B' levels are floating and hence

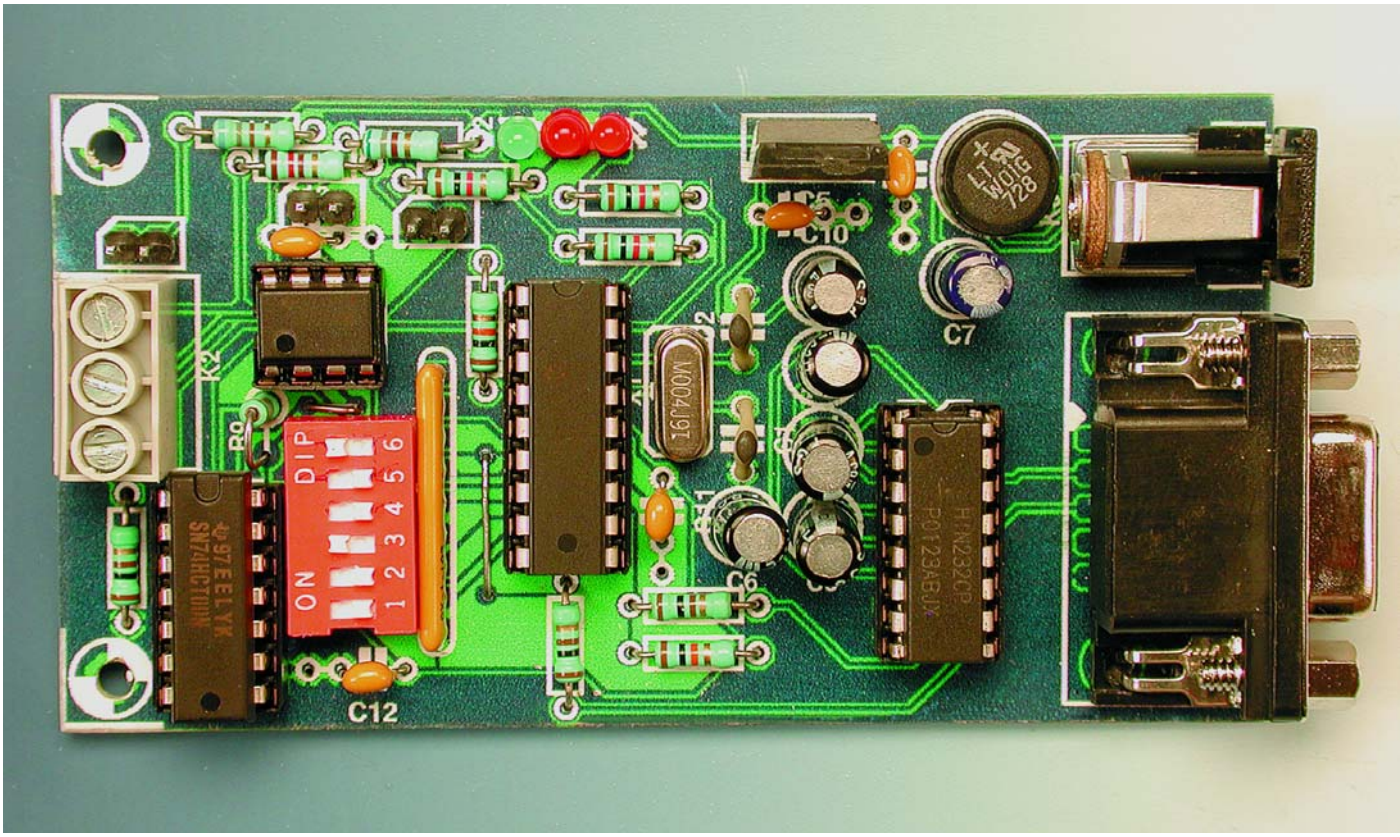


Figure 3. Prototype built up in the Elektor lab.

can be affected by electrical noise picked up via inductive or capacitive coupling. In this case, since the differential input voltage falls in the undefined zone, the output of the '485 receiver will exhibit random pulses and hence random data. To avoid problems, a well defined difference is forced between 'A' and 'B', thanks to R13-JP2, which act on the positive side of the bus, and R11-JP1, dedicated to the negative side. When both JP1 and JP3 are inserted, the bus sees a steady idle condition, ensuring optimum rejection of noise and other disturbances. The value of R13 and R11 ensure a light load to any real driver linked to the '485 bus.

When fitted, jumper JP3, placed in series with R10, acts as a bus terminator. A terminator, despite any reference to Hollywood and cybernetics, is a simple remedy to eliminate reflection observed when a short pulse is applied to an electrical line. In practice, the generator sourcing the pulse exhibits its own electrical impedance, as does the line where the pulse travels. Only if the two impedances are identical, the pulse travels from one end to the other without any trouble. If the two impedance values are not the same, the pulse reaches to the end of the line where it is (partially) reflected, creating an undesired disturbance with possible loss of data. The impedance of the generator, i.e., the '485 bus driver, is known and accountable, while the impedance of the line is hardly ever known, since it depends on the physical length of the conductors, as well as on the data speed. However, from a lot of experimental data acquired in the field, we can assume that a 120-Ω resistor, placed on both physical ends of the line, will ensure a near-perfect impedance match for almost any application. Please note that terminators have to be placed only at the physical ends of the line, even when we link together more than two '485 transmitters on the same bus. Finally, there's R12. It acts as a protection against possible current drain due to voltage difference between different earth potentials. Normally, the voltage of the earth is zero, but in practice, several volts may be measured between a ground point on the device, and the protection earth (PE). R12 provides a safe 'local' ground for the communication line, even when the earth level is not at zero volts.

The circuit configuration around microcontroller IC1 closely resembles a suggested circuit found in the datasheets. Crystal X1 and 22-pF capacitors C1 and C2 are the external elements for the on-chip clock oscillator.

Port lines RB2-RB7 are used to read the switches in S1, while RB1 monitors the incoming data from the EIA-485 receiver, and RB0 does the same for the incoming stream from the '232 side. Resistor network R4 is used to pull up all lines connected to the config switches.

The microcontroller firmware causes LED D3 to light up when a '485 transmission takes place. LED D2 will be on when the '485 bus is receiving, and will be off when the bus is transmitting. Notice that the above visual indicators are provided only for certain presets of the DIP switches, because the firmware in the micro only has enough computing time at relatively slow communication rates.

Thanks to the use of bridge rectifier B1, the unit may be powered by an AC or DC adapter with a minimum output voltage of about 9 volts, connected to K2. Regulator IC5 steps down the unregulated dc voltage across C7-C8 to a clean +5 V. Current consumption of the circuit is of the order of a few tens of milliamps.

Assembly and use

All the components that go into the circuit are accommodated on a printed circuit board of which the layout is given in **Figure 2**. Sockets are fitted for the four ICs with a dual-in-line (DIL) footprint, while IC5, with only three pins, is soldered directly on to the board. Do observe the polarity of all semiconductors as well as the positive/negative references on the electrolytic capacitors.

The PIC microcontroller in position IC1 is available ready-programmed through our Readers Services (order code **020083-41**). Alternatively, you can program your own PIC using the files found on diskette **020083-11,** or on this month's Free Downloads page, see our website at www.elektor-electronics.co.uk

Practical use

If you have a personal computer, you can link K1 to COM1 or COM2 on your PC via a **standard** (non-crossed) RS232 cable with at least wires on pins 2, 3, and 5. Please do not use a 'null-modem cable' (also called zero-modem cable) because it has crossed TxD/RxD wires. At the '485 side a standard PCB screw terminal block is available to accept any wiring, providing the 'A' and 'B' paths, and possibly a third one for the common protection ground. Obviously, the connection to an external '485 device should observe the 'A' and 'B' references, otherwise there's no hope of obtaining any communication. Although a mismatch will not cause physical damage to the circuits, reversing the polarity results in data corruption to the extent that a break is created in the communication path. In practice, confusing 'A' and 'B' will result in total absence of communication.

To be able to use the adapter you should know the speed of the incoming data on the '232 channel, and also the number of bits in every chunk of information. A chunk of information, technically referred to as a **word**, will take some time to travel from the '232 input to the '485 output, and during this time the bus should be switched to transmission mode, otherwise no communication could take place with any hope of success.

As already mentioned, an immediate switch from receive to transmit is guaranteed by the direct monitoring of the incoming '232 data. But when does the '485 bus return to receive mode? Assuming that an EIA-232 dataword consist of the usual 10, 11 or 12 bits (start, data, stop and optional parity), the switches in S1 allow you to inform the processor about the duration of a complete data word. IC4 can then be gated appropriately to ensure a prompt switch to transmission, followed by a quick return to reception, should the external '485 device respond to the stimulus and give an immediate feedback.

Typical applications of a '485 communication path include actuator control or sensor data acquisition. Thus, we are faced with some data to transmit, followed by some other data to receive. In this scenario, the adapter, like a human interpreter, should not introduce any appreciable delay during information transfers from one side to the other, otherwise it fails its job. The firmware in IC1 is actually capable of sustaining communications at different data rates, from 110 to 115,200 bits per second, with different word sizes from 10 to 13 bits per word.

(020003-1)

Microcontroller Basics Course (5)

part 5: the UART, timers and interrupts

By B. Kainka

The previous instalments of the Microcontroller Basics Course have primarily concentrated on programming languages. Now it's time to take a closer look at the microcontroller hardware. This instalment deals with the UART, timers and interrupts.

The internal serial interface of every 8051 microcontroller makes it possible to create a simple data link to a PC. Since the interface is implemented as an autonomous hardware UART, the processor is scarcely burdened by this task. Bytes are received and sent by the UART operating fully in the background. The only things requiring attention are correct initialisation and the ongoing transfer of data to and from the UART.

To understand exactly how the UART is programmed, we need to look at the special function registers (SFR) for the serial interface. To initialise the interface, we must first load SCON register with suitable parameters (see **Table 1**).

SBUF (SFR 099h) is the data register of the serial interface. It is actually a 'front' for two registers, namely the transmit data register and the receive data register. A transmit process is initiated by simply performing a write access to the SBUF register. In the other direction, a received byte can be read from SBUF. In 9-bit mode, the ninth bit (TB8 or TB8) can always be found in the SCON register.

Usually only operating mode 1 is used, which means an 8-bit UART with the baud rate generated by Timer 1. The 8-bit data stream is initiated by one start bit and terminated by one stop bit. The baud rate is 1/16 (if SMOD = 1) or 1/32 (if SMOD = 0) of the overflow rate of Timer 1. SMOD is the most significant bit of the PCON register (Power Control, SFR 87h), which is otherwise responsible for controlling the power-down modes (see **Table 2**).

Table 1 SCON (SFR 98h)

7	6	5	4	3	2	1	0
SMO	SM1	SM2	REN	TB8	RB8	TI	RI

```
SM0 SM1   Serial Mode bits 1 & 2 select the operating mode:
0   0     Mode 0:        8-bit shift register
0   1     Mode 1:        8-bit UART, baud rate via Timer 1
1   0     Mode 2:        9-bit UART, 375 kBaud at 12 MHz
1   1     Mode 3:        9-bit UART, baud rate via Timer 1
SM2:      multiprocessor mode
REN:      Receiver Enable
TB8:      Transmitted Bit 8 for 9 -bit mode
RB8:      Received Bit 8 for 9-bit mode
TI:       Transmitter Interrupt, '1' if transmission successful
RI:       Receiver Interrupt, '1' following receipt of a character
```

Generating a clock signal for the serial interface requires first programming Timer 1. Each of the two timers in the 8051 has two 8-bit counter registers that can be loaded and read. The 89S8252, by the way, also has a third timer, but we will disregard it for now.

```
TL0 (SFR 8Ah): Timer 0, low byte
TH0 (SFR 8Ch): Timer 0, high byte
TL1 (SFR 8Bh): Timer 1, low byte
TH1 (SFR 8Dh): Timer 1, high byte
```

Table 2 PCON (SFR 87h)

7	6	5	4	3	2	1	0
SMOD	–	–	–	GF1	GF2	PD	IDL

```
SMOD      1 = high baud rate, 0 = low baud rate
GF1, GF2  freely usable flags
PD        Power-Down mode (only for CMOS)
IDL       Idle mode
```

Table 3 TMOD (SFR 89h)

7	6	5	4	3	2	1	0
Gate	C/T	M1	M0	Gate	C/T	M1	M0
Counter 1				Counter 0			

```
Gate     The associated counter is enabled via the Int0 or
         Int1 pin.
C/T      0: Timer, 1: Counter
M1   M0  Operating mode
 0    0  13-bit timer/counter
 0    1  16-bit timer/counter
 1    0  8-bit timer/counter with automatic reloading
 1    1  only for Counter 0: two separate 8-bit counters
```

Table 4 TCON (SFR 88h)

7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

```
TF1:     overflow flag for Timer 1
TR1:     start Timer 1
TF0:     overflow flag for Timer 0
TR0:     start Timer 0
IE1:     interrupt flag for Int1
IT1:     interrupt on Int1 with edge triggering
IE0:     interrupt flag for Int0
IT0:     interrupt on Int0 with edge triggering
```

Listing 1. The serial interface in assembler.

```
;Serial port access       (COMPORT.ASM)
;11,059MHz, 9600 Baud
#include 8051.H
         .org 0000H

INIT     clr   TR1              ;stop timer 1
         mov   TH1,#0FAH        ;256-6: 9600 baud
         mov   TL1,#0FAH
         anl   TMOD,#0FH        ;Timer1: 8 bit auto-reload
         orl   TMOD,#20H
         setb  TR1              ;start timer
         mov   SCON,#50H        ;InitRS232
         setb  TI
         orl   PCON,#80H        ;SMOD=1


NEXT     acall RX
         mov   P1,A             ;Port write
         nop
         nop
         mov   A,P1             ;Port  read
         acall TX
         sjmp  NEXT

RX       jnb   RI,RX
         mov   A,SBUF
         clr   RI
         ret

TX       jnb   TI,TX
         clr   TI
         mov   SBUF,A
         ret
        .end
```

The properties of the timers are controlled via registers TCON (SFR 88h) and TMOD (SFR 89h), while the operating mode is configured using TMOD (see Table 3).

Timer 1 should be used as a free-running counter counting the internal clock pulses of the 8051. It is thus configured as a mode-2 timer. Each time the counter overflows, a preset counter state is reloaded into the counter.

Each counter is enabled via its TR bit in the TCON register. TCON also contains the overflow flags for both timers as well as four control bits for interrupt-driven control via the INT0 and INT1 pins (see Table 4).

In order to use Timer 1 as a baud rate generator, all we have to do is to set the TR1 bit. This bit is bit-addressable and can accessed at bit address 8Eh. However, first the division factors must be configured with both timers stopped. In operating mode 2, the high counter byte (TH1) contains the reload value, which is reloaded into the actual counter register (TL1) each time an overflow occurs. For a transmission rate of 9600 baud, a clock frequency of $(16 \times 9.6\ \text{kHz}) = 153.6\ \text{kHz}$ is required if the SMOD bit is set. Since the counter is clocked by the internal clock, which runs at 1/12 of the crystal frequency, a division factor of 6 should be set if the crystal frequency is 11.0592 MHz:

$$11059.2\ \text{kHz} \div 12 \div 6 \div 16 = 9.6\ \text{kHz}$$

A division factor of 6 is achieved by using a reload value of $(256 - 6) = 250$ (FAh), since the counter counts up.

Serial data transmission

The UART can be used with any desired programming language. Using it with assembler is a particularly good way to illustrate the involved processes. **Listing 1** shows a sample program in which the interface is used to provide access to Port 1 via RS232. A PC can thereby modify and read the port status.

For initialisation, all relevant registers are first loaded with their control parameters. The main routine `NEXT` then needs only the two subroutines `TX` and `RX`. The receive subroutine `RX` first polls the RI bit until it finds that a character is present. The receive buffer SBUF is then read and RI is reset. In the other direction, the transmit subroutine TX polls the TI bit until the previously transmitted byte has been fully processed. The byte to be sent is then written to the send buffer SBUF, which starts the transmit portion of the UART.

Each received byte is transferred to Port 1 in the main routine. After a short delay generated by two NOP instructions, the program

reads the status of Port 1 and sends back the byte it has read via the serial interface. We have thus implemented a parallel-to-serial converter that can be used only for port outputs, only for read accesses to the port or for mixed operation. All leads to be used as inputs must be set high by the sent data in order to put them in a high-impedance state. The effect of sending a value of 255 (FFh), for example, is to cause the entire port to be available as an input port and initiate a read transaction.

This program can be tested using the program Terminal.exe, for example, which was already used in the December 2001 issue of *Elektor Electronics* in connection with an IR transceiver. The significant difference here is that what is being transferred is bytes rather than numerical values in text format. Figure 1 shows an example of how the port can be driven. The first two control bytes are returned unchanged, which means that the microcontroller reports that the sent port state was present at P1 with no changes. The third byte (255) sets all port pins High, which makes them usable as inputs. This time, however, the response from the microcontroller shows that one line is being pulled to ground externally. This program can thus be used in this manner to only poll all eight port lines.

It's even easier to use the serial interface in C than in assembler, since suitable modules are already present. The serial interface can be initialised by `InitSerialPort0` using the parameter `DEF_SIO_MODE`. The functions `getc()` and `putc()` transfer individual bytes. A program having exactly the same functions as the previously described assembler program can thus be easily written (see **Listing 2**).

In BASIC-52, the UART is already initialised by the system. It is no problem to do the same job in Basic. However, in this case it is easier to work with numerical values in text format, since the Basic interpreter always keeps an eye on the data received via the serial interface (it's always possible for a `Ctrl-C` to appear if the user wants to stop a program). The Input command thus always receives whole lines, which must be terminated with CR. **Listing 3** shows a very simple program for the 'remote control' of Port 1. These functions can be tried out directly using the Basic terminal program.

Interrupts

In many cases it is necessary for the microcontroller to perform several tasks quasi-concurrently. An effective way to realise this is to use interrupt handling, with a main routine that does its own work most of the time but

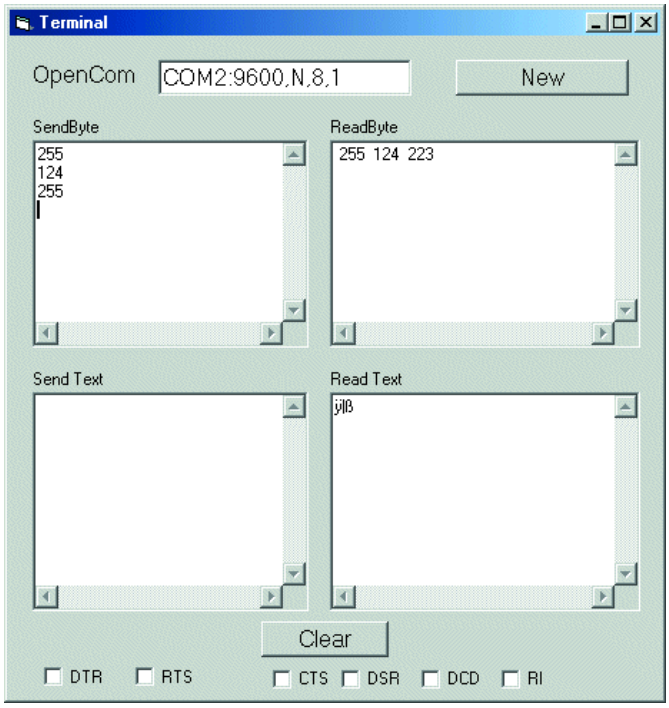


Figure 1. Port outputs and port polling.

it can be briefly interrupted by special events in order to perform a second task.

The principle of such an interruption can be illustrated very easily using BASIC-52. Every 8051 microcontroller has two interrupt inputs called Int0 (P3.2) and Int1 (P3.3). BASIC-52 supports Int1 with the `ONEX1` command. Whenever the P3.3 line is pulled Low externally, this Low edge triggers an interrupt. The program then branches to the line number given after `ONEX1`. The actual interrupt subroutine must end with `RETI`. **Listing 4** shows a simple example. Here a counting loop with incrementing port outputs runs in

Listing 2. The UART in READS-51.

```
// ————————- READS51 generated header ————————
// module  : uart.c
// ——————————————————————————

#define TRUE  1
#define FALSE 0

#include <sfr51.h>
#include <Sio51.h>

main(){
char n;

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

// endless loop
 while(TRUE)
  {
   n=getc();
   P1=n;          // Port output
   n=P1;          // Port read
   putc(n);
  }
}
```

Listing 3. Port accesses in BASIC-52

```
1   REM Port I/O (PORTIO.BAS)
10  INPUT N
20  PORT1=N
30  N=PORT1
40  PRINT N
50  GOTO 10
```

Listing 4. Interrupt control in BASIC-52.

```
10  REM Interrupt (INT1.BAS)
100 ONEX1 500
200 REM main prog
210 FOR N=0 TO 255
220 PORT1=N
230 NEXT N
240 GOTO 210
500 REM Interrupt subroutine
510 PRINT "Interrupt P3.3"
520 RETI
```

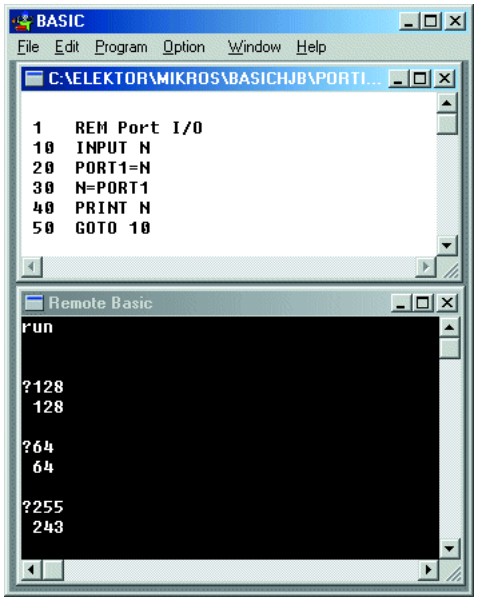


Figure 2. Direct access to Port 1.

the foreground and the subroutine called by the interrupt sends a message to the PC.

BASIC-52 also has the `ONTIME` command, which reacts to a particular time event in precisely the same manner. However, these are only a few of many possible sources of interrupts. The processor also recognises interrupts from its timers and the serial interface. Several interrupts can also be employed in a single program, in which case it is necessary to specify the priorities assigned to the individual interrupts. The interrupt with the highest priority is allowed to interrupt all other interrupt routines.

The processor has a central control element for enabling interrupts in the form of the Interrupt Enable (IE) register (see **Table 5**). The EA bit disables or enables all of the configured interrupts.

For each interrupt there is an associated interrupt address to which the processor automatically jumps. This address must hold a jump instruction to the starting address of the corresponding interrupt routine, which in turn must end with a `reti` instruction.

```
0023h SINT    Serial interface
001Bh TIMER1  Timer 1
0013h EXTI1   External interrupt
              1
              (P3.3)
000Bh TIMER0  Timer 0
0003h EXTI0   External interrupt
              0
              (P3.2)
0000h RESET   Reset
```

The next example (**Listing 5**) shows the serial interface being used with interrupts. In our first serial interface example, the program remains 'stuck' in the receive routine each time it is waiting for a new character to arrive, but here an interrupt is triggered when a character is received.

Received characters are directly transferred to register R7. A square-wave generator whose period is set by the value located in R7 runs in the main routine. This means that the frequency of the generator can be externally changed without any noticeable response-time delay.

Now we want to show you how to handle interrupts in READS-51. This time Timer 0 should trigger an interrupt. Here you have to keep in mind that Timer 1 is often already used for the serial interface, so it is not available for time control. The 89S8252 also has a third timer, Timer 2, which is used in BASIC-51 (for instance) to generate the baud rate. Timer 2 is very versatile, but you must remember that it is not present in the smaller 8051 derivatives and may also have different properties in certain microcontrollers (such as the 80535). If we use Timer 0 here for time control, we can thus use our program with all other 8051-compatible microcontrollers.

Here we want to attempt to generate a precise square-wave signal with an edge spacing of 1 ms (see **Listing 6**). Timer 0 and Timer 1 each receive a clock signal at one twelfth of the crystal frequency. If the crystal frequency is 11.059 MHz, we have to count to 921 to obtain an interval of 1 ms. Since we are dealing with up counters that trigger interrupts when they overflow, we must specify a value of (65536 – 921) = 64516 (FC67 hex). This value is loaded into the counter registers TH0 and TL0 the first time the counter is started and then again each time an interrupt is triggered by a carry-out signal.

An interrupt function can be created very easily in READS-51 using the keyword 'interrupt' and the microcontroller's associated interrupt address. It is also necessary to initialise the timer in the Main function, and the appropriate interrupt bits must be set.

Table 5 IE (SFR A8h)

7	6	5	4	3	2	1	0
EA	–	–	ES	ET1	EX1	ET0	EX0

```
EA      enable all interrupts
ES      enable serial interrupts for sending and receiving
ET1     enable Timer 1 interrupt via TF1
EX1     enable external interrupt 1 (P3.3)
ET0     enable Timer 0 interrupt via TF0
EX0     enable external interrupt 0 (P3.2)
```

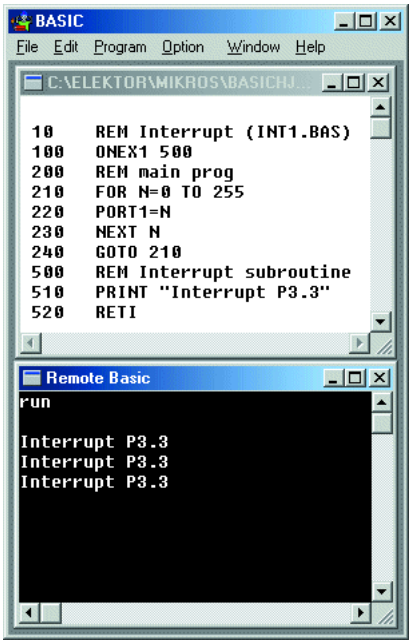


Figure 3. Interrupt-driven message generation.

This program is suitable for investigating the execution times of C programs. An oscilloscope connected to port P1.0 will measure a precisely symmetrical square-wave signal. The duration of each level is approximately 1.25 ms. The code generated by the compiler thus requires around 250 μs for the actual jump to the interrupt function and the re-initialisation of the timer. If a more accurate 1-ms timing event is needed, you can try fine-tuning the initial value of the counter.

The timer has four different modes that can be used to cover various time ranges. In this example we use mode 1, which can generate timing events of up to 65 ms. Without reloading inside the interrupt function, the following times can be achieved (in round numbers):

Mode 0: 13-bit timer (8 ms)
Mode 1: 16-bit timer (65 ms)
Mode 2: 8-bit auto-reload (0–0.25 ms)
Mode 3: TH0 = 8-bit timer (0.25 ms)

In these examples, we have seen how the same techniques can be used in different programming languages. Suitable initialisation of the relevant function registers is a prerequisite for using the hardware. With a bit of experience, you will be able to glean the appropriate settings from an existing program and then use them with a different programming language.

(010208-6)

Listing 5. Interrupt-driven character reception.

```
;Serial Interrupt       (COMINT.ASM)
;11,059 MHz, 9600 Baud
#include 8051.H
         .org 0000H
         ljmp INIT
         .org 0023H
         ljmp RX

INIT     clr   TR1              ;stop timer 1
         mov   TH1,#0FAH        ;256-6: 9600 baud
         mov   TL1,#0FAH
         anl   TMOD,#0FH        ;Timer1: 8 bit auto-reload
         orl   TMOD,#20H
         setb  TR1              ;start timer
         mov   SCON,#50H        ;InitRS232
         setb  TI
         orl   PCON,#80H        ;SMOD=1
         mov   IE,#90H          ;EA +ES

NEXT     mov   A,R7
         mov   R1,A
         mov   A,#255
         mov   P1,A
ON       djnz  R1,ON
         mov   A,R7
         mov   R1,A
         mov   A,#0
         mov   P1,A
OFF      djnz  R1,OFF
         sjmp  NEXT

RX       mov   R7,SBUF
         clr   RI
         reti

        .end
```

The next (and final) instalment of this course deals with the LC display, which is mapped into the address space of the external memory as a set of components.

Listing 6. A timer interrupt in C.

```
// ———————— READS51 generated header ————————
// module  : C:\Rigel\Reads51\Work\Inter\Inter.c
// created : 09:16:45, Thursday, March 07, 2002
// ——————————————————————————
#include <sfr51.h>

void interrupt (0x000B) square(void)
{
  P1_0 = ! P1_0;
  TR0=0;              //stop timer 0
  TH0=0xFC;           //timer 0 1ms
  TL0=0x67;
  TR0=1;              //start timer 0
}

main(){
  int n;
  n=0;
  TH0=0xFC;           //timer 0 1ms
  TL0=0x67;
  TMOD=TMOD & 0xF0;
  TMOD=TMOD | 0x01;   //timer 0 16 bit
  TR0=1;              //start timer 0
  ET0=1;              //timer 0 interrupt
  EA=1;               //enable interrupts
  while(1);
}
```

DMX Dimmer

for semi-professional applications

Design by A. Kneer

Using a digital DMX signal, a 0-10 V control voltage or manual control, this dimmer can individually set the brightness of up to four spotlights, each of which may present a load of up to 1 kW.



The DMX Dimmer described in this article is aimed at the semi-professional application area. It is operated using a light-mixing panel (digital DMX or analogue). It can dim lights (and it does this in an outstanding manner), but it intentionally lacks non-essential frills such as a built-in running-light display or sound-to-light control. Instead, it features optional lamp preheating (with an adjustable range of 0-100% brightness) for gentle lamp operation and it can be used in stand-alone mode, at a fixed dimming level, without a light-mixing panel. Features such

Features

- compatible with DMX-512 and 0–10 V light-mixing panels
- convenient channel selection using thumbwheel switches
- maximum connected load: one 1000-watt spotlight per channel
- phase-controlled dimmer for (halogen) incandescent lamps using triacs
- linear lamp control using digital characteristic curve correction for the phase angle
- configurable lamp preheating: 0 to 100 %
- compact construction and simplest possible mounting on spotlights, light masts or traverses
- robust, interference-immune design

as simple channel selection using thumbwheel switches and digital linearisation of the control range round out the concept.

In lighting technology, spotlight bars with four or six spotlights are common. In order to keep the dimmer featured here simple and compact, it has been designed for single-phase operation. If we assume 1 kW per spotlight, three spotlights can be connected to a 16-A circuit. A 6-light bar thus requires two dimmers with three channels each. In order to allow the dimmer to be fitted next to the spotlights, a metal enclosure in the form of a somewhat oversized three-way mains power bar has been developed. The enclosure also acts as the heat sink for the triacs.

Since decoding the DMX signal anyhow requires the use of a microcontroller, all of the dimming functions are largely implemented in software. The 68HC11F1 microcontroller used here contains 1 kB of internal RAM and 512 bytes of internal EEPROM, and the software has been squeezed into this memory to avoid using an external ROM. This saves space and money. An extensive range of integrated peripheral functions, such as a serial interface and A/D converter, combined with the timer structure of the processor, allows the circuit and the software to be kept very compact.

The block diagram (**Figure 1**) shows the software functions. The software is split into three different tasks having different priorities.

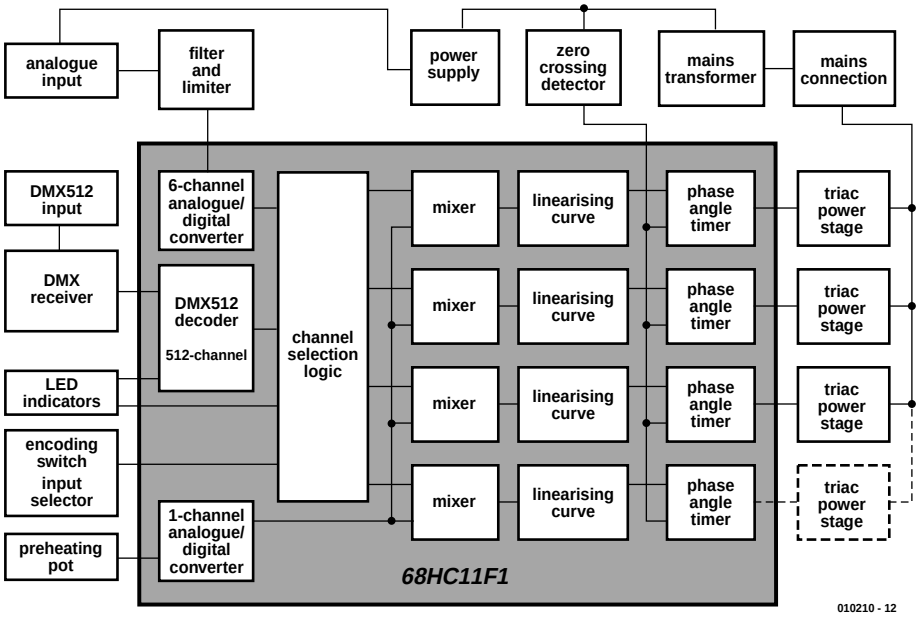


Figure 1. Block diagram of the DMX Dimmer.

The DMX task

DMX-512 is increasingly supplanting the 0-10 V analogue signal that was previously common. The DMX-512 protocol is electrically identical to RS485, so all types of RS485 line drivers can be used. Using a two-wire cable, the control unit sends data frames (each containing eight utility bits) at a rate of 250 kbit/s. The data can be decoded in the receiver using a UART interface connected to a processor. A DMX data frame contains 11 bits (including the start bit and two stop bits) and thus has a transmission time of typically 44 µs. The DMX standard specifies that the data bytes may be send with a maximum spacing of 1 s. However, no minimum spacing is specified. If a DMX receiver is to work with all types of transmitters, it must be able to process all of the data frames, regardless of their time separations. If the light-mixing panel is very fast and sends successive frames without any delays, the microcontroller is left with only the frame time of 44 µs for processing the data. This requires a high-performance processor and optimised software.

The DMX transmitter starts the transfer with a Break (a Low level on the DMX+ line lasting at least 88 µs). This 'rule violation' allows the receiver to recognise the start of the transfer. Next, the transmitter sends the start byte, which indicates the target device for the data. DMX dimmers may only respond to data beginning with the start byte '0'. The start byte thus makes it theoretically possible to transfer data for 512 dimmer channels in addition to 512 scanner channels, 512 moving-light channels and so on. However, this option is seldom used.

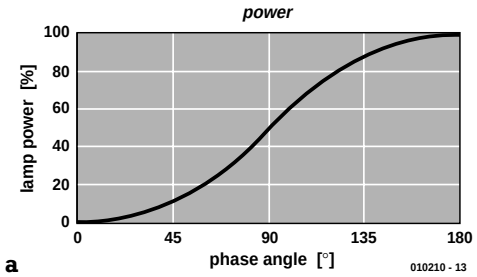
After the start byte, the transmitter starts to sequentially send the data for the individual channels. The final channel to be sent may be channel 512, but this is not mandatory. The sequence may be terminated earlier by sending a Break. You will find more information about the DMX512 transfer modes in the May 2001 issue of *Elektor Electronics*.

The DMX handler in the software, which complies with the DMX512/1990 standard, reads all 512 channels (if the start byte is '0') and stores the values in the RAM. Since the DMX handler has only 44 µs to do its job, this task is assigned the highest priority in the system. The DMX handler is called as soon as a byte has arrived in the UART receive buffer.

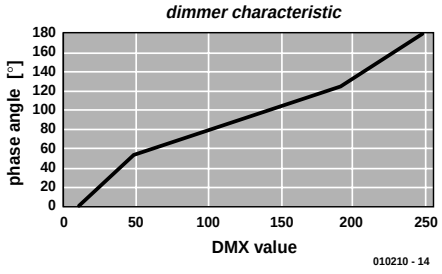
The main task

The main routine of the software evaluates the settings of the controls and drives the light-emitting diodes. The input signals for the dimmer originate either from the internal A/D converter, which receives analogue signals via the 8-way DIN socket commonly used for technical lighting, or directly from the DMX channel data. Depending on the setting of the channel selection switches, the selection logic picks out three of the six analogue channels or three DMX channels and retrieves their brightness values from the A/D converter or the DMX intermediate storage registers. Following this, the values are compared with the brightness values for lamp preheating, and the larger of each pair of values is converted into an 'on'-time control value for the associated triac using a linearisation formula.

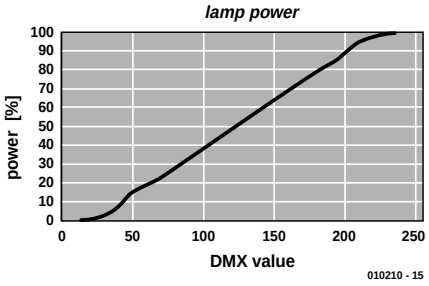
Since the mains voltage is sinusoidal rather than being a square wave, the amount of power supplied to the lamp is not proportional to the phase angle setting (see **Figure 2a**). In order to allow the brightness of the spotlight to be precisely set using the light-



a



b



c

Figure 2. a) lamp power versus phase angle; b) the software uses this curve to linearise the power; c) the result of the linearisation: a linear response with 'flat' portions for low and high DMX values.

mixing panel, even for small values, a linear control characteristic is desirable. In order to achieve this, the dimmer has an inverse characteristic (**Figure 2b**).

The control characteristic of the dimmer is 'flat' at the upper and lower ends of the curve. This is important for analogue mixing panels in order to prevent a few millivolts of residual voltage from causing the lamps to be illuminated as well as to avoid not being able to achieve 100% brightness. The characteristic curve correction yields a nearly linear characteristic curve for power versus digital value (**Figure 2c**). The result from the characteristic curve correction, which is the value for the duration of the 'on' time, is then written to the timer register of the microcontroller.

The timer task

The triacs are driven using the 68HC11 output-compare registers. The Timer task is called on each zero crossing of the mains voltage. When the Timer task is called, the current value of the free-running 68HC11 counter (clocked at 1 μs) is temporarily stored, following which the phase angle is added to this value and stored in the output-compare register. After this, the stored value is continuously compared with the current counter value without any involvement on the part of the software. If the counter value is equal to the value in the output-compare register, the triac is energised. The triac is de-energised by the Timer task at the next zero crossing. As a result, a PWM signal synchronised to the mains frequency appears at the 68HC11 output.

Analogue and DMX inputs

At the upper left of the schematic diagram (**Figure 3**) you will see the power supply for the low-voltage portion of the circuit. Fuse F2 is a temperature-sensitive fuse. If the air temperature inside the enclosure becomes greater than 95 °C for whatever reason, this fuse switches off the dimmer forever. Note that the 95 °C value also applies to soldering, so be careful when assembling the circuit!

The supply voltage for an analogue mixing panel is also tapped off from the regulator, separately fused by F3 (a Polyswitch fuse, or more precisely a PTC thermistor) and fed out via the DIN socket. This also makes it possible to connect a very simple 'mixing panel' consisting of only a 10-V voltage regulator and a potentiometer.

The rectified secondary voltage is tapped off from the anode of D1. Since D1 'isolates' smoothing capacitor C16, the zero crossings of the AC voltage that are important for the timer are also present at this point. The zero crossings are detected by a comparator circuit using IC4. The actual comparator, IC4a, is provided with hysteresis by R21. This is because dimmers are often adversely affected by momentary dropouts in the mains voltage resulting from the effects of lead inductance and resistance when a heavy load is switched near the zero crossing, which the zero-crossing detector incorrectly interprets as zero crossings. The hysteresis minimises this problem. IC4b buffers and inverts the output signal so it can be applied to the interrupt input of the microcontroller.

A network consisting of anti-parallel dual diodes (D3–D6), two Schottky diodes (D10 and D11), capacitors C9–C14 and several resistors (R28–R33, resistor array R46 and R45) forms the six-channel analogue input for the dimmer. Each of the six channels is wired identically. The protective resistors form voltage dividers that reduce the input voltage range of 0–10 V to a range of 0–5 V. Together with the capacitors, the voltage dividers form RC filters that eliminate HF noise voltages. The dual diodes limit any excess positive or negative voltages that may be present. Each of these diodes is connected to the 4.7-kΩ resistor R45 and one of the two Schottky diodes (D10 or D11). Due to the voltage drops of the Schottky diodes, the voltages present at R45 are approximately 0.3 V and 4.7 V (with respect to ground). This means that the voltage at the microcontroller inputs cannot be greater than 4.7 V plus one diode threshold voltage (5.4 V in total) and cannot drop below –0.4 V (= 0.3 V – 0.7 V). On the circuit board, the inputs are connected to a header strip that in turn is connected to the previously mentioned 8-way DIN socket, as shown in **Figure 4**. The RC low-pass filter formed by R43 and C16, which provides a noise-free 5-V reference, also belongs to the A/D converter.

The DMX-512 input is built around an LTC485, a bi-directional RS485 line driver. The transmit function is not used here and is disabled. The input lines are fed to the IC via a low-pass filter that eliminates HR noise. The output of the receiver is connected directly to the serial interface of the 68HC11. The wiring between K3 and the DMX connectors is shown in **Figure 5**.

Three LEDs are driven by microcontroller port pins PD5–PD8 via transistor driver stages to indicate

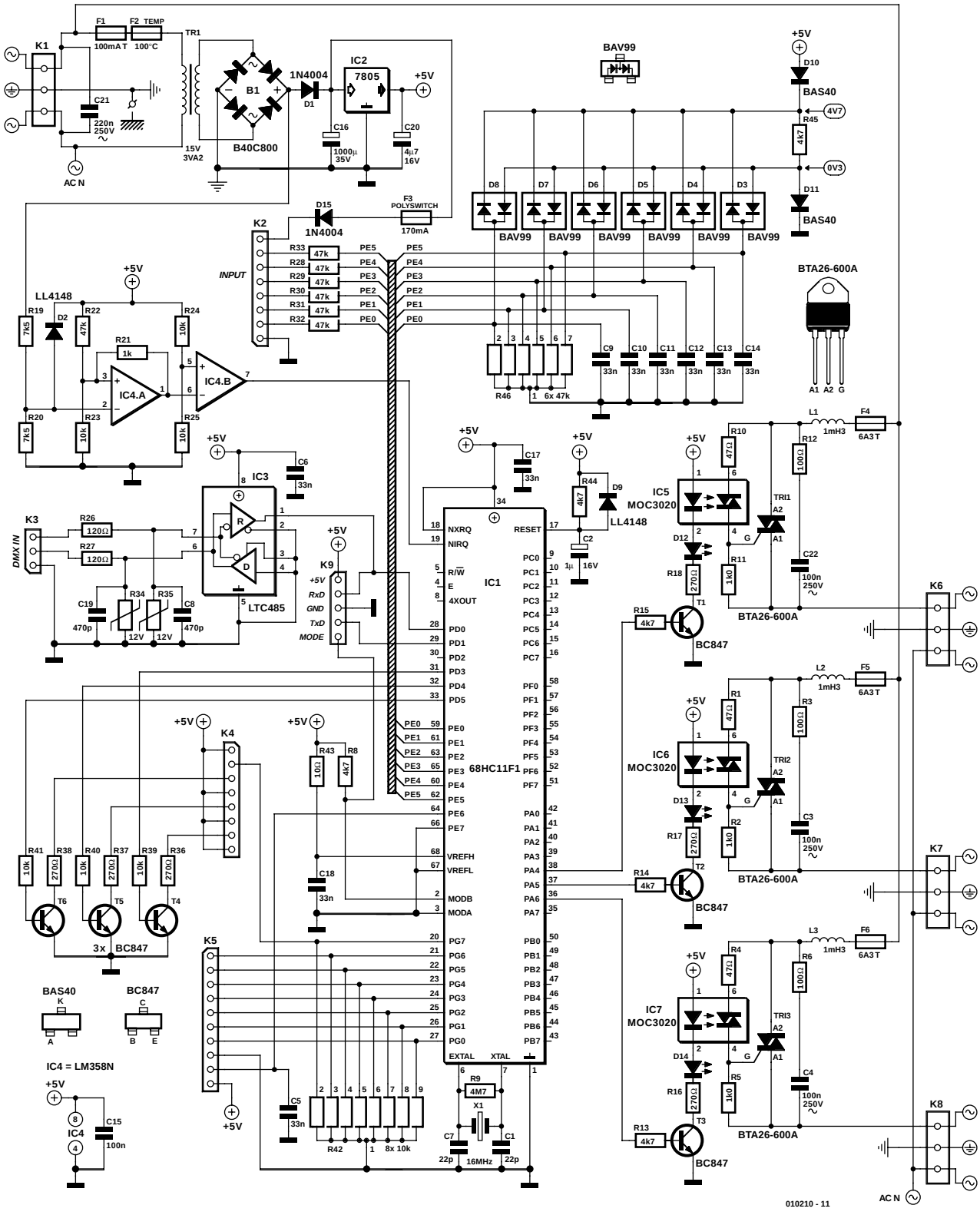
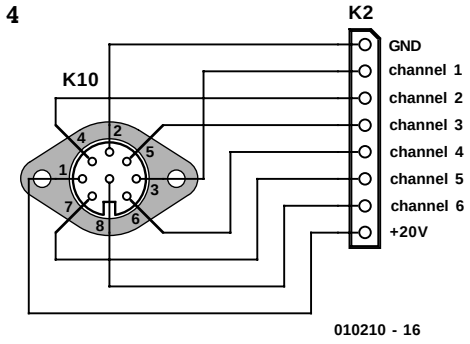
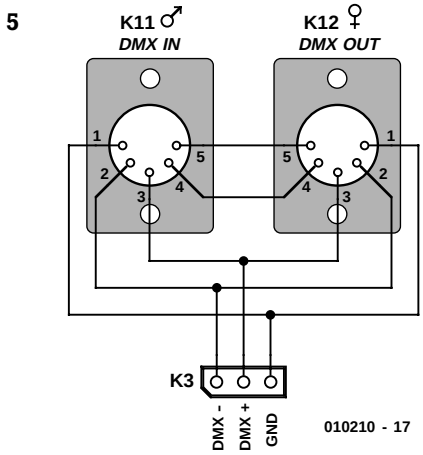


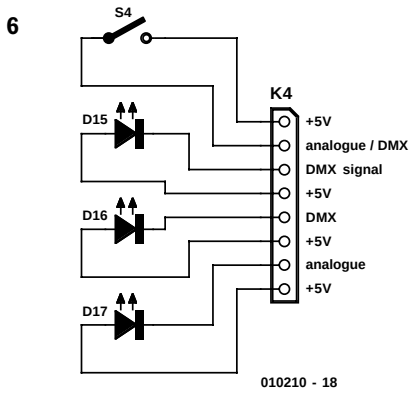
Figure 3. The schematic diagram, showing the inputs on the left and the power electronics on the right. The microcontroller that mediates between the two is shown in the middle.

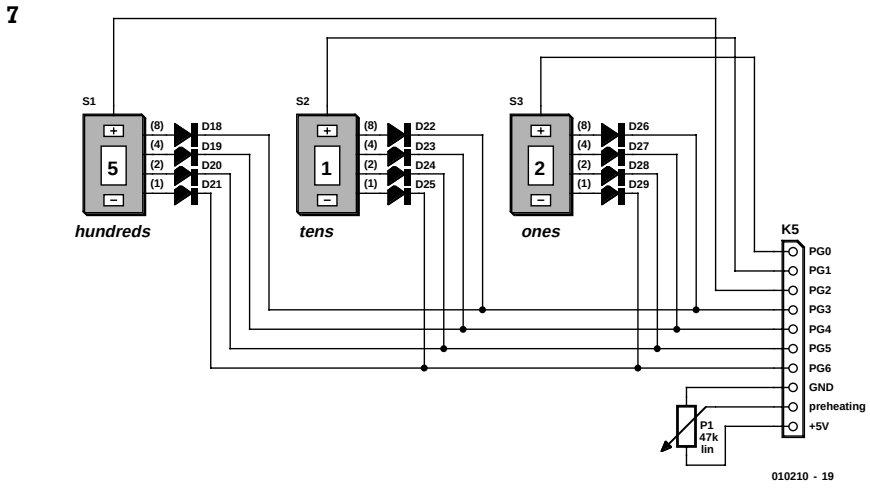
whether the DMX or analogue input is active. These LEDs are connected to K4 along with the DMX/Analogue selector switch. The switch position is read via port PG7. The wiring between K4 and the LEDs and switch is shown in **Figure 6**.

The channel-select thumbwheel switches are connected via K5 (see **Figure 7**). These switches have binary outputs that are wired in parallel after being isolated by external 1N4148 diodes connected directly to the outputs. To evaluate the switch positions, the microcontroller multiplexes the three switches via PG0–PG3 and









Figures 4–7. The connections between the controls and indicators and the circuit board.

recognises the set values via port pins PG4-PG6.

The analogue channels are also affected by the thumbwheel switches. If channel 1 is selected, the control voltage on PE0 determines the brightness of the lamp connected to PA6, the voltage on PE1 regulates PA5 and the voltage on PE2 regulates PA4. If channel 2 is selected, PE1 regulates PA6, PE2 regulates PA5 and so on.

The pot for setting the preheating level is connected to the remaining pins of K5. The wiper of the pot is connected directly to the ADC input of the microcontroller (PE6). A capacitor connected in parallel eliminates potential noise.

Outputs

The processor itself needs no additional circuitry except for the 16-MHz oscillator and the reset circuit (R44, C2 and D9). Resistor R8 connected to MODB ensures that the processor starts up in single-chip mode following a reset. We can thus concentrate our attention on the power outputs.

The three triac output stages are connected to PA4–PA6. The same circuit is used for each output stage.

A type BTA26-600A triac is used here. It is specified for a maximum current of 26 A, which may appear to be overdimensioned, but it must be borne in mind that an incandescent lamp has a very low cold resistance and thus a correspondingly high inrush current (up to ten times the rated current). Besides this, some halogen lamps have the unpleasant property of producing a short circuit when they burn out. The more robust the triac, the higher the probability that it will survive a lamp failure. Compared with the widely used TIC type, the BTA type used here has the advantage that its case is insulated from its leads, so it is not necessary to use elaborate insulation hardware. Each triac is driven by a second triac integrated into an optocoupler. As soon as the microcontroller sets its output High, the attached transistor conducts and current flows through the LED in the optocoupler. The triac in the optocoupler then fires and current flows from the 'live' mains lead to the gate of the power triac. This triac fires in turn, short-circuiting the path via the optocoupler and allowing current to flow through the lamp.

Depending on the switch-on phase angle, it can happen that the full mains voltage is suddenly applied to the lamp, which leads to a correspondingly steep rise in current. However, steep current rises are undesirable from an EMI perspective. In order to limit the steepness of the rising edge, a coil is connected in series with the lamp. This coil is also the component that limits the power capacity of the dimmer to 1 kW per lamp, since its maximum allowable current is 4.5 A.

During switch-on the voltage across the triac undergoes a step change, and this can create interference. In order to reduce this, a RC network is connected across the triac. This network also isolates the triac from voltage peaks, such as those that occur when an inductive load is switched off, and thus prevents false triggering.

In spite of these measures, voltage and current peaks are still generated, and in order to minimise the amount of noise from these peaks that is fed back into the mains network, a 220-nF capacitor is connected across the mains leads at the mains input. This noise-suppression capacitor must be an approved X-type capacitor.

Assembly

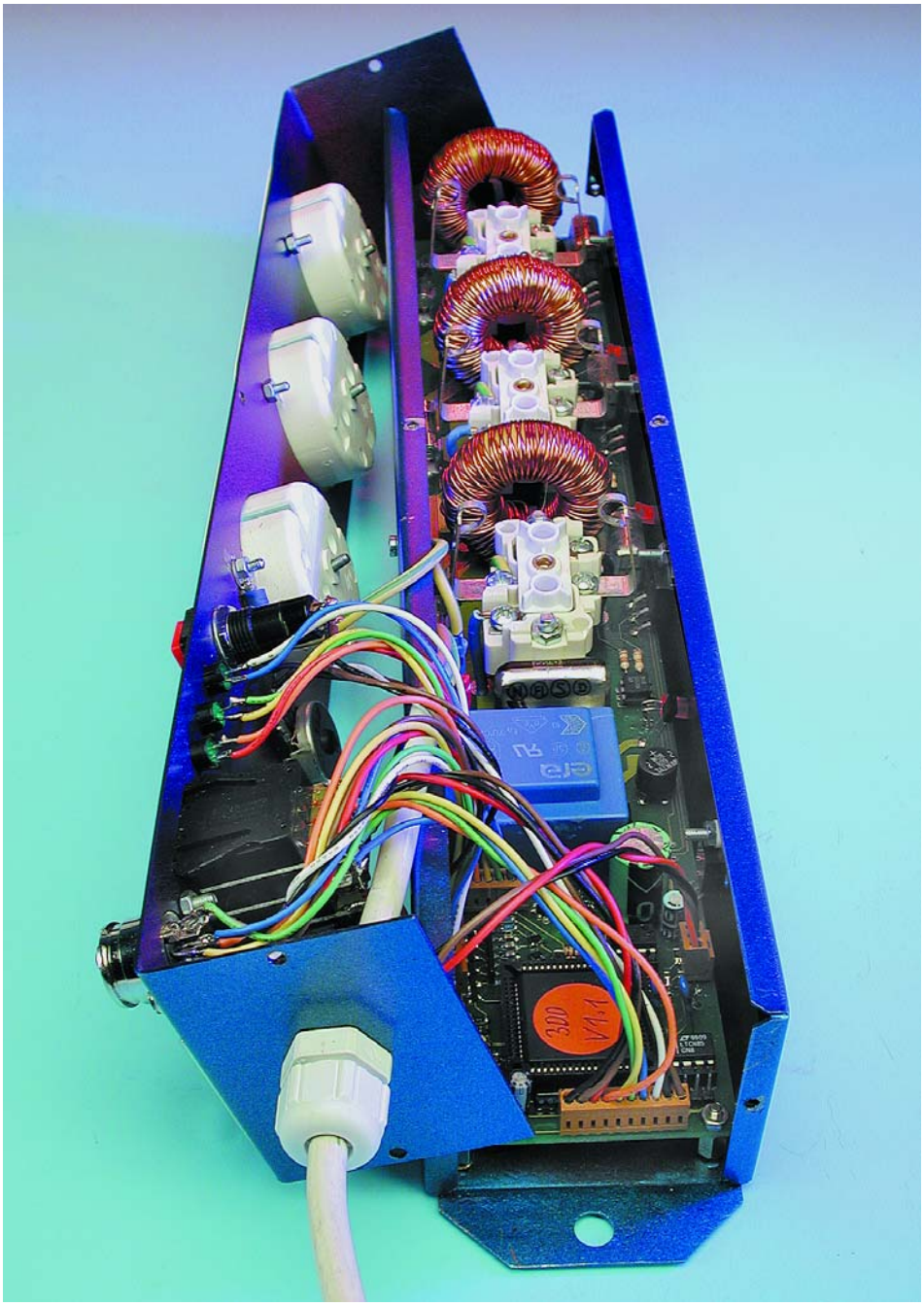
In order to keep the dimmer compact, SMD components (0805 outline, or if necessary 1206) are used for the

control portion of the circuit. The SMD transistors and diodes specified in the components list are not critical (but be careful with D10 and D11), and they can certainly be replaced by similar components having the same lead arrangements. When fitting the components to the double-sided printed circuit board (shown at slightly reduced scale in **Figure 8**), it is best to start with these components. After they are fitted, all of the other components can be fitted, working from the smallest to the largest. When fitting the temperature-sensitive fuse, you must take care not to set it off. Soldering SMD components is a rather fussy process that is best performed under the watchful eye of a magnifying lamp.

The triacs (as well as the voltage regulator) must be cooled. The simplest solution is a metal enclosure. However, you will search in vain for a suitable ready-made enclosure in the electronics catalogues, so for better or worse it must be a DIY item. The best approach is to take inspiration from the small aluminium cases produced by Teko, which are easy to bend. The holes in the lid for the outlets can be cut to fit using a fretsaw, and in addition, as shown in the title picture, you will have to make holes for the three connectors, the thumbwheel switches, three LEDs, the Analogue/DMX selector switch and the pot. Drill the fastening holes for the triacs (as well as the associated LEDs) along one side, together with hole for fastening the voltage regulator. In contrast to the triacs used here, the voltage regulator may not be insulated from the case (using a mica washer, for example).

When you have finished machining the enclosure, set the partly finished printed circuit board into the enclosure and tack the voltage regulator and triacs to the top side of the circuit board using a drop of solder after first firmly screwing them to the enclosure. They can be properly soldered afterwards. This prevents mechanical stresses from arising.

The circuit board tracks passing underneath the coils must be insulated from the coils by sticking pieces of insulating tape on top of them. Attach the coils using cable ties and a bit of hot-melt glue and then solder their leads in place. Fit the earthed mains outlets on 15-mm standoffs so that each outlet stands above a coil. Clamp lengths of copper wire with a diameter of 1.5 mm to the terminals of the sockets, pass them through the holes in the circuit board and solder them in place.



Safety

Now we come to the subject of electrical safety. Whenever you are dealing with electronics working at the mains voltage, and in particular in the case of a Class-I device with a metallic enclosure, the circuit board layout is especially important. A certain separation (3 mm) must be maintained between components carrying mains voltage and the circuit board tracks or the enclosure. Furthermore, all exposed metallic parts of the enclosure and all metallic parts leading to the outside, such as metallic knobs, must be securely tied to the protective earth lead. The circuit ground in the low-voltage part of the circuit is also connected to the protective earth lead (PE). Although 'Schuko' type mains sockets are shown in the photograph, UK readers should use 13A panel mount appliance sockets complying with BS1363 **only**. Do not use non-insulated triacs, since although it would

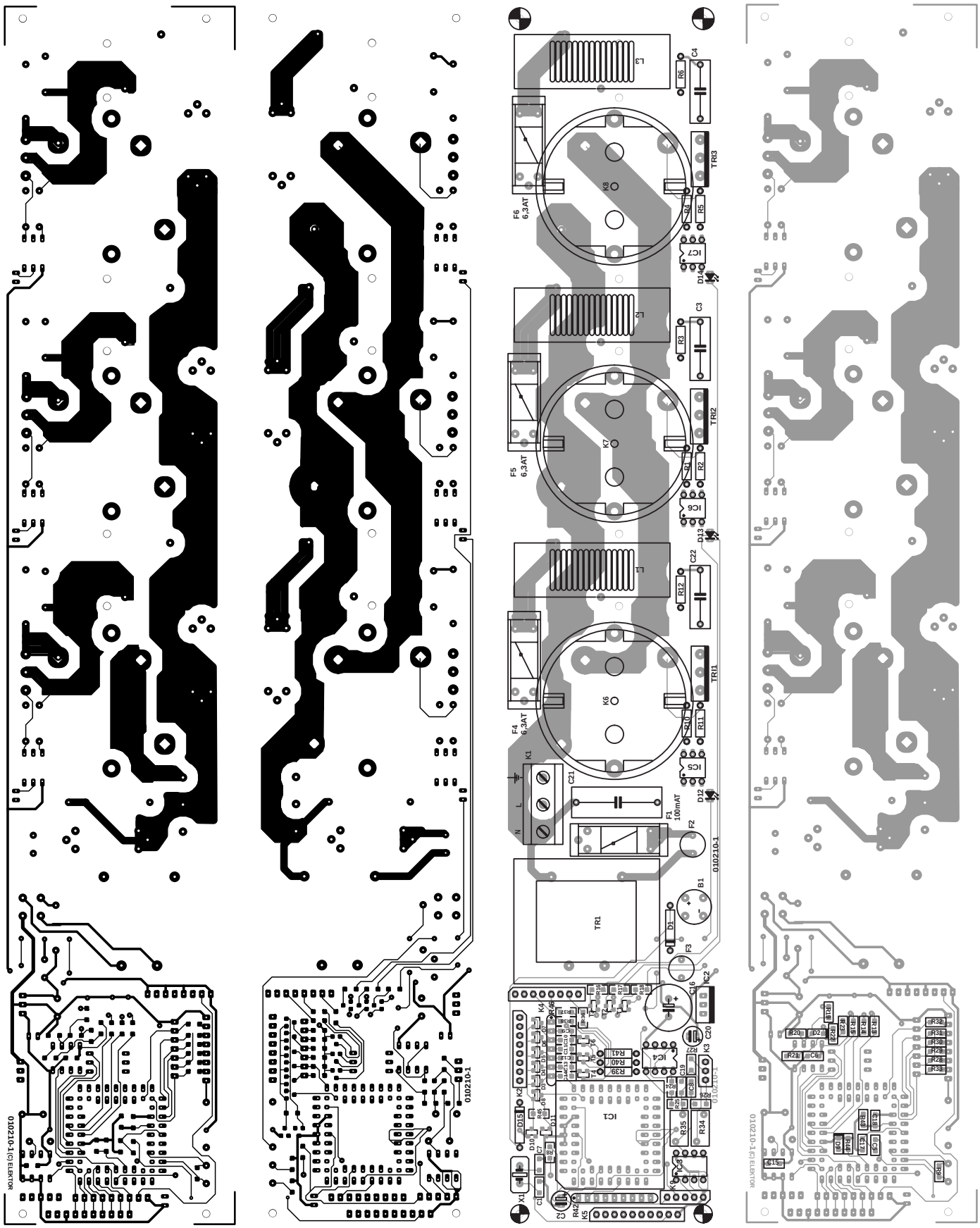


Figure 8. Everything on a single printed circuit board: the component layout and track layout for the dimmer circuit board at a scale of 1 : 1.5.

COMPONENTS LIST
(all SMD devices are size 0805)

Resistors:
R1,R4,R10 = 47Ω
R2,R5,R11 = 1k0Ω
R3,R6,R12 = 100Ω
R7 = not fitted
R8,R13,R14,R15,R44,R45 = 4kΩ7 SMD
R9 = 4MΩ7 SMD
R16,R17,R18 = 270Ω SMD
R19,R20 = 7kΩ5 SMD
R21 = 1kΩ SMD
R22,R28-R33 = 47kΩ SMD
R23,R24,R25 = 10kΩ SMD
R26,R27 = 120Ω SMD
R34,R35 = 12V Varistor K14/12
R36...R38 = 270Ω
R39...R41 = 10kΩ
R42 = 8-way 10kΩ SIL array
R43 = 10Ω SMD
R46 = 6-way 47kΩ SIL array

Capacitors:
C1,C7 = 22pF SMD
C2 = 1μF 16V radial
C3,C4,C22 = 100nF 250VAC, Class X2
C5,C6,C9.-C14,C17,C18 = 33nF SMD
C8,C19 = 470pF SMD
C15 = 100nF SMD
C16 = 1000μF 35V radial
C20 = 4μF7 16V radial
C21 = 220nF 250VAC Class X2

Semiconductors:
B1 = B40C800 bridge rectifier, round case (40V piv, 800mA)
D1,D15 = 1N4004
D2,D9 = LL4148
D10,D11 = BAS40
D3-D8 = BAV99
D12,D13,D14 = LED
IC1 = 68HC11F1FN, programmed, order code **010210-41***
IC2 = 7805
IC3 = LTC485 CN8
IC4 = LM358N
IC5,IC6,IC7 = MOC3020
T1-T6 = BC847

Miscellaneous:
F1 = fuse, 100mA, slow, with PCB mount holder and protective cap
F2 = 100°C thermal fuse (Conrad Electronics #534137)
F3 = 170mA Polyswitch (Conrad Electronics #551198)
F4,F5,F6 = fuse, 6.3A, slow, with PCB mount holder and protective cap
K1 = 3-way PCB terminal block, lead pitch 7.5mm
K3 = 3-way pinheader
K2,K4 = 8-way pinheader
K5 = 10-way pinheader
K6,K7,K8 = 13A panel mounting socket to BS1363
K9= 5-way pinheader
L1,L2,L3 = toroid suppressor coil, 1.3mH, 4.5A (Conrad Electronics #534366)
TR1 = Mains transformer, 15V, 3.2VA
TRi1,Tri2,TRI3 = BTA26-600A (isolated case!), 600V, 25A, 100mA drive current
X1 = 16MHz quartz crystal
68-way PLCC socket
PCB, order code **010210-1***
* see Readers Services pages and website www.elektor-electronics.co.uk

Ancillaries
K10 = 8-way DIN socket, chassis mount
K11 = 5-way XLR plug (male), chassis mount
K12 = 5-way XLR socket (female), chassis mount
D15,D16,D17 = LED, green
S1,S2,S3 = BCD thumbwheel switch
D18-D29 = 1N4148 (solder directly to S1, S2, S3)
P1 = 47kΩ potentiometer, linear
S4 = on/off switch, 1 make contact

certainly be possible to do so electrically, the necessary insulation using mica washers and plastic screws is **not permissible**. The frames of the thumbwheel switches are fastened to the top of the enclosure, which additionally secures the circuit board. Plastic screws are only allowed to be used if a minimum insulation gap of 3 mm is maintained. You can read more on this subject on the 'Electrical Safety Guidelines' page, which appears regularly in *Elektor Electronics*.

Once everything has been solidly assembled and the mains cable has been fitted using a good strain relief, as shown in the photo, you can start with commissioning. For this, first connect three incandescent lamps. As soon as the mains voltage is applied to the device, at least one LED must go on or start blinking. The brightness of the lamps can now be adjusted from 0 to 100% using the pot. Next, you should connect a voltage source that can be adjusted over the range of 0-10 V and set the selector switch to 'Analogue', which causes the 'Analogue' LED (D17) to go on. Set the code switches to 001 and the LED should stop blinking, if it has not already done so. If a voltage is now applied to analogue channel 1, the lamp connected to outlet 1 should light up; if a voltage is applied to channel 2 the lamp connected to outlet 2 should light up and so on. The code switch determines the channel to which outlet 1 is assigned. For example, if the switch value is set to '3', outlet 1 is assigned to channel 3, outlet 2 to channel 4 and outlet 3 to channel 5. If the channel selection is set to '0' or a value greater than '6', the LED flashes to indicate an error.

Next, connect a DMX signal, which will cause the 'DMX' LED to blink in sync with the DMX transfers. In order to have the lamps be controlled by the DMX signal, you must set the selector switch to the 'DMX' position. The 'Analogue' LED will then go out and the 'DMX' LED will go on. It should now be possible to control the lamp connected to outlet 1 using the DMX channel selected by the code switch, with the outlet 2 being controlled by the next higher channel. If an illegal DMX value ('0' or greater than '512') is selected using the code switch, the 'DMX' LED will flash.

One more tip before you get started: the maximum load per channel is 1 kW. In the *Elektor Electronics* lab, we checked out the functions of the DMX Dimmer using 100-W spotlamps (the maximum load that can be used without fitting the triacs to a heat sink). You should do the same when checking out the unit. Besides this, you should check the temperature of the voltage regulator while testing the operation of the dimmer. If this component becomes too hot, you have no choice but to use a higher-power type or a supplementary heat sink.

(010210-1)

The fourth channel

The circuit of the dimmer is designed such that a four-channel version is possible for four-light bars. The signal for the optional fourth triac output stage is present at PA3 (pin 39) of the microcontroller. The author can provide a circuit diagram and a circuit board layout for the four-channel version. Interested parties may inquire at:

AKneer@aol.com.

WorldSpace Radio

the first worldwide satellite radio

by G. Kleine

The new digital WorldSpace satellite radio broadcast service covers a large fraction of the earth's surface using three geostationary satellites. Over four billion people — 80% of the world's population — can receive WorldSpace.



[illustration reproduced courtesy of HIT MIX FMM AG, Unterföhring]

WorldSpace is chiefly aimed at the inhabitants of Africa, the Near East, Asia and Latin America, since these are regions which are not well supplied with radio stations. However, thanks to the rapid advance of satellite technology, the WorldSpace programmes are now available elsewhere as well. Indeed, the West beam of the AfriStar satellite, aimed at Africa, can be received in Europe as far north as Scandinavia and Scotland. In this article we will describe the communications technology behind WorldSpace and the receivers, which are now becoming available.

The WorldSpace satellite radio system arose from the idea of providing a wide range of radio programmes to areas of the developing world not well supplied with radio stations, but without the expense of building a terrestrial transmitter network. The WorldSpace company itself was founded in 1990 by the Ethiopian Noah A. Samara (now 42 years old) and is based in Washington, USA. There are three satellites, AfriStar, AsiaStar and AmeriStar, which send out a wide selection of radio programmes. Each of the three satellites has three partially overlapping footprints, the so-called 'beams' (**Figure 1**). There is a West beam, an East beam, and a South beam.

The satellites are geostationary and are monitored and controlled from control centres on each continent. They were built by Alcatel Space of Toulouse and Matra Marconi Space in France. AfriStar and AsiaStar have already been launched on Ariane rockets from Kourou in French Guyana. AmeriStar will follow this year.

Some well-known electronics companies are also involved with WorldSpace radio: Micronas Intermetall are producing the Starman chipset for the WorldSpace receiver and Rohde & Schwarz produces the signal generators used in the development and production of the receivers.

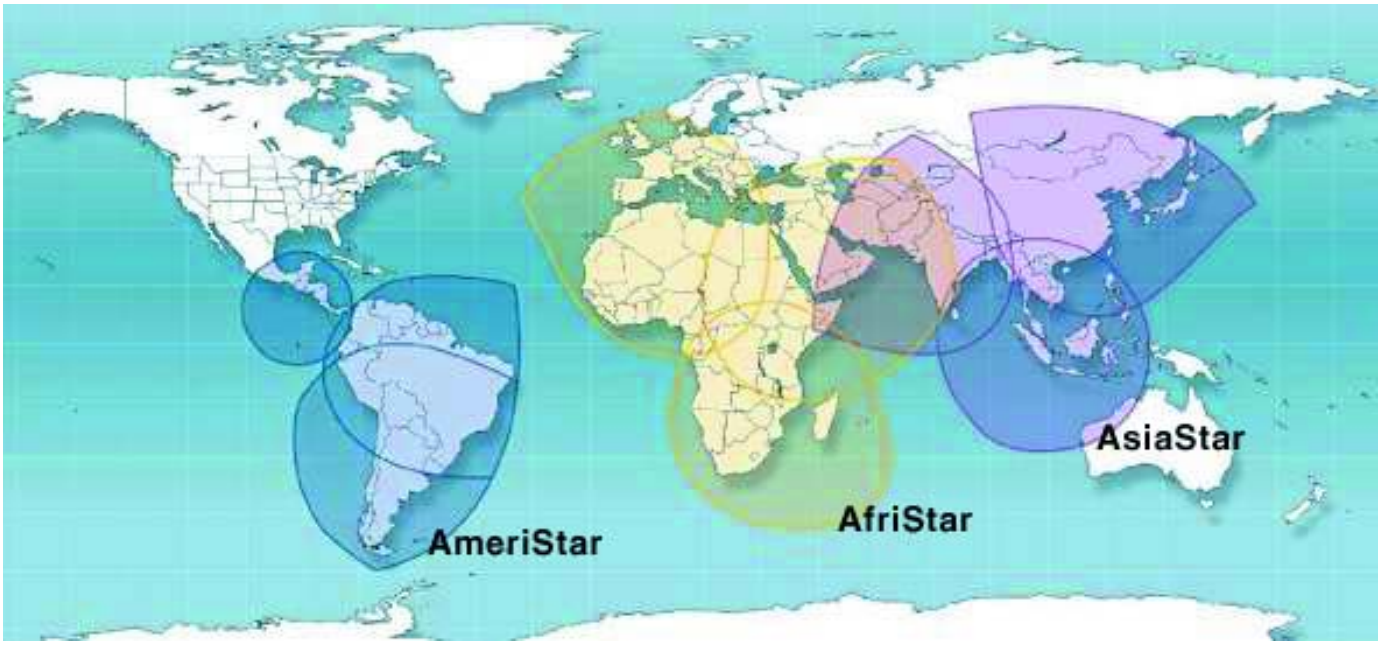


Figure 1. Coverage of the three WorldSpace satellites (illustration reproduced courtesy of Worldspace Germany).

Flexible channels

In each beam two signals are transmitted, each carrying 96 basic channels (so-called Prime Rate Channels, or PRCs), each of which runs at 16 kbit/s in time division multiplex (TDM), providing a total of up to 576 (= 2 x 3 x 96) basic channels per WorldSpace satellite. The digital communication employs compression according to the MPEG layer 3 protocol (also known as MP3). According to the sound quality required a radio station can occupy from one half of a basic channel (8 kbit/s, mono at AM radio quality) to as many as eight basic channels (128 kbit/s, stereo at CD quality): see **Table 2**.

The allocation of the basic channels leased by a programme provider can be changed from time to time, so that for example a news broadcast in six languages at mono quality can share capacity with a hi-fi broadcast at 96 kbit/s (6 x 16 kbit/s = 96 kbit/s). The selection is made via the Service Components of the signal, described later. The programme provider can also allocate a fraction of the basic channels to data, to provide accompanying textual information in a similar fashion to Teletext. Channel selection is also simplified for the user by the use of this side data channel (station name, language and programme type — see **Table 3**).

Further, a programme provider can transmit in encrypted form using so-called 'Pay Audio'. The listener is provided with a code number which must be entered into the receiver in order to receive the channel.

In order to avoid the need to bring all the channels together at a single satellite uplink station, WorldSpace provides for a TDM data stream (one per beam) whereby each programme provider has their own SCPC (single channel per carrier) uplink, in addition to a TDM channel per beam from a central uplink station. Using the 7 GHz band, the various WorldSpace customers transmit their com-

Table 1. Summary of WorldSpace satellites

Satellite	AmeriStar	AfriStar	AsiaStar
Position	95° West	21° East	105° East
Target region	South America, Caribbean region	Africa, Near East, Middle East, Central and Southern Europe	China, India, Indonesia, the Philippines, Thailand, Japan
Uplink	7.025 to 7.075 GHz (X-band)		
Downlink	1.452 to 1.492 GHz (L-band)		
Transmit power per beam	2x 150 W (TWTA)		
Launch date	2002 (planned)	28 October 1998	21 March 2000
First operation	2002 (planned)	19 October 1999	10 June 2000
Operating centre	Trinidad and Tobago	Washington, D.C.	Melbourne
TCR stations	Santiago, Chile Washington, D.C. Trinidad & Tobago	Libreville, Gabon Bangalore, India Port Louis, Mauritius	Melbourne Bangkok, Thailand Port Louis, Mauritius

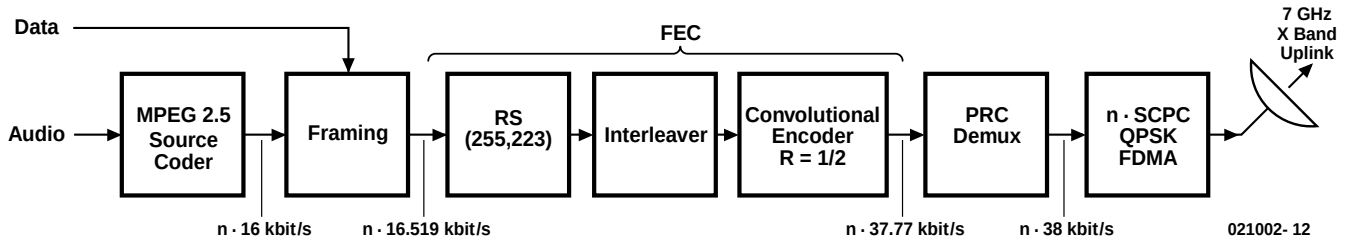


Figure 2. Block diagram of a WorldSpace SCPC uplink station.

Table 2. Data rates and quality levels

n = number of basic channels	data rate	audio bandwidth	quality	maximum number of programmes per satellite
1	16 kbit/s	4.5 kHz	AM radio	288
2	32 kbit/s	7.5 kHz	FM radio, mono	144
4	64 kbit/s	11 kHz	FM stereo	72
6 to 8	96 to 128 kbit/s	> 15 kHz	CD quality	36 to 48

pressed data streams to the satellites where they are received, demodulated and coded into a WorldSpace standard signal along with data streams from other programme providers. The result is a separate data stream for each beam. The carrier of each TDM signal is modulated using QPSK and has a total bit rate of 3.68 Mbit/s. Because bits are encoded in pairs into the phase of the carrier, we can quote a symbol rate of 1.84 Msymbol/s.

Each of the three beams of the satellite carries two TDM signals: one with 96 basic channels, which is constructed from the SCPC uplinks, and another, which comes from the central uplink and which in the satellite undergoes only a change in frequency, without being demodulated or decoded: the transponder is essentially transparent. This second signal also contains 96 basic channels. Each beam therefore has 192 basic channels available. In order to make more efficient use of the frequency band, circular polarisation is used; this is also ideal for use with flat (or 'patch') antennas at the receiver. Each beam thus includes a right-hand circularly polarised and a left-hand circularly polarised TDM QPSK signal, whose spectra can overlap. **Table 4** gives more information on the carrier frequencies of the TDM signals.

Uplink

The uplink system starts with an MPEG 2.5 encoder, as shown in **Figure 2**. Depending on the quality required, this produces a data stream at a rate of n times 16 kbit/s. Next there is a framing stage which organises the

Table 3. Languages and programme types

Languages	Programme types
English	News
Spanish	Information
Arabic	Sport
Portuguese	Education
French	Children
German	Talk
Mandarin	Word Entertainment
Hindi	European Classical
Bahasa	Soundtrack
Tagalog	Jazz
Swahili	Country
Hausa	Pop Music
Urdu	Urban
Korean	other
Thai	
other	

Abbreviations:

BC	Broadcast Channel
C/N	Carrier / Noise
FDMA	Frequency Division Multiplexing Access
FEC	Foward Error Correction MFP Master Frame Preamble
MPEG	Motion Picture Experts Group
PRC	Prime Rate Channel
QPSK	Quaternary Phase Shift Keying
RS	Reed-Solomon
SC	Service Component
SCPC	Single Channel per Carrier
TCR	Telemetry, Command & Ranging
TDM	Time Division Multiplex
TDMA	Time Division Multiplexing Access
TSCC	Time Slot Control Channel
TWTA	Travelling Wave Tube Amplifier

Table 4. Transponders on AfriStar

Beam	Transponder*	Frequency
East beam	34R	1468.564 MHz
East beam	39L	1470.864 MHz
South beam	44L	1473.164 MHz
South beam	49R	1475.464 MHz
West beam	55R	1478.224 MHz
West beam	60L	1480.524 MHz

*nnR = right-hand circularly polarised, nnL = left-hand circularly polarised

data into a framed format, along with the side channel data. Now at a rate of *n* times 16.519 kbit/s, the data undergo FEC (forward error correction) coding, consisting of a Reed-Solomon encoder, an interleaver and a convolutional encoder.

The Reed-Solomon encoder uses a block code, where 32 extra parity bytes are added to each 223-byte block. The parity bytes are calculated from the data using a complex algorithm which allows the receiver to correct as many bit errors as possible. Because the Reed-Solomon stage converts 223 bytes into 255 bytes, it can be referred to as an RS (255,223) encoder. With 32 parity bytes up to 16 erroneous bytes in each 223 byte block can be corrected. If there are more errors than this, then it is only possible to detect the errors, not correct them.

The interleaver which follows reorders the bytes in time according to a fixed pattern. This means that, if a long burst of errors occurs in transmission, when the bytes are reassembled into the correct order by the deinterleaver in the receiver the erroneous ones will be spread out, and it will then be possible to correct the individual errors.

The final part of the forward error correction coding is the convolutional encoder. This consists of a 7-bit shift register, whose outputs produce two new data streams, each at the original data rate, according to a fixed formula given by two so-called generator polynomials $G_x = 171_{oct} = 1111001_{bin}$ and $G_y = 133_{oct} = 1011011_{bin}$. This is therefore a 'rate 1/2 code': the output data rate is twice that of the incoming data. The convolutional encoder further improves the overall robustness of the communications system to external interference.

The Prime Rate Channel demultiplexer constructs individual 38 kHz channels from the FEC-protected data stream, and they are finally transmitted to the WorldSpace satellites as a bundle of QPSK-modulated signals in frequency-division multiplex (FDMA, or Frequency Division Multiple Access: see **Figure 3**) using carriers in the X-band (7.025 to 7.075 GHz). The SCPC uplink station must of course lie within the footprint of the satellite's receiving antenna. Transmit powers of between 10 W and 100 W are used, with an antenna diameter of two to three metres.

Satellite

The FDMA signal is received in the WorldSpace satellite (**Figure 4**) and QPSK-demodulated. A demultiplexer separates it into the individual 38 kHz signals so that all the PRC channels from all the programme providers are available at the multiplexer that follows. This allows each individual data stream to be switched to one or more of the three channels on the three beams of the satellite. The next block performs rate alignment, where the data rates

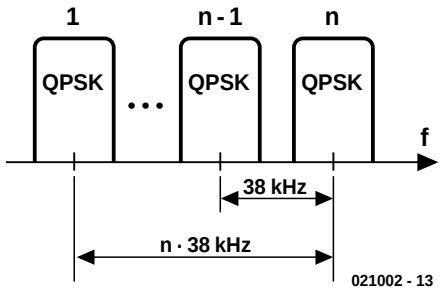


Figure 3. FDMA spectrum from a WorldSpace SCPC uplink station.

of each beam are adjusted to a nominal value. The following TDMA (Time Division Multiple Access) multiplexer organises the data streams for each beam into a single stream which contains all the channels in a time-multiplexed arrangement. In order to make the master frame identifiable a Master Frame Preamble (MFP) is added. Information is also required as to which Prime Rate Channels are associated with which programme channel, or so-called Broadcast Channel (BC). This is held in the Time Slot Control Channel (TSCC) in the data stream, which contains the allocation of the PRCs that follow it.

The overall data stream so produced can now be fed to the QPSK modulator and the satellite's power output stage. This can deliver about 300 W of RF power per beam. The antenna then beams the WorldSpace signal in the L-band (1.452 to 1.492 GHz), with either left-hand or right-hand circular polarisation, back down to earth.

The programme providers can divide their available PRCs (up to eight) into so-called Service Components (SCs). For example, 8 PRCs can be divided into a stereo audio channel

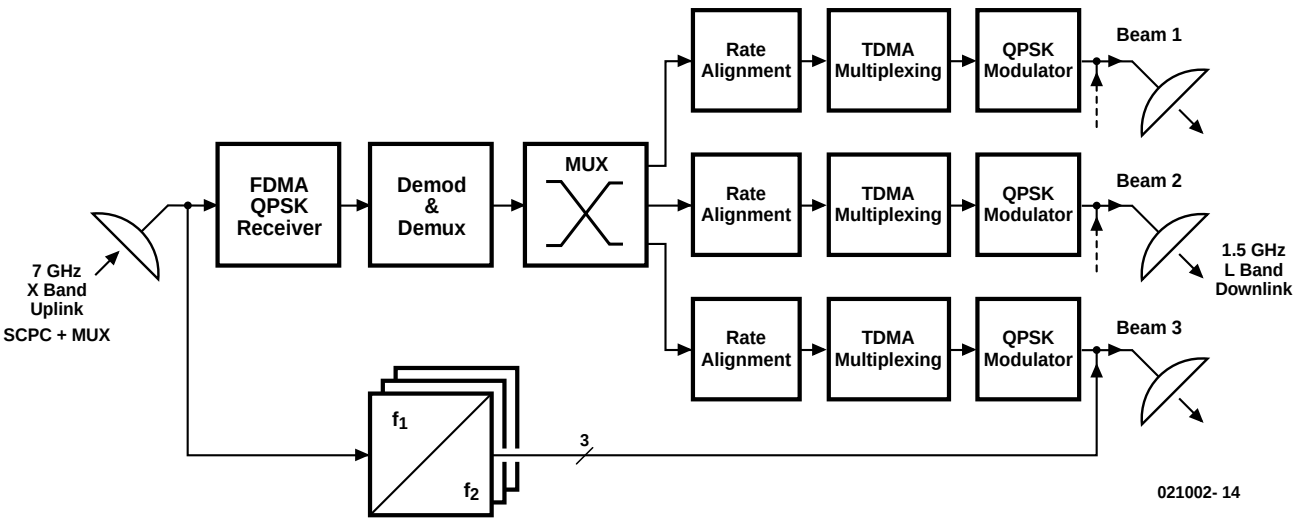


Figure 4. Block diagram of a WorldSpace satellite.

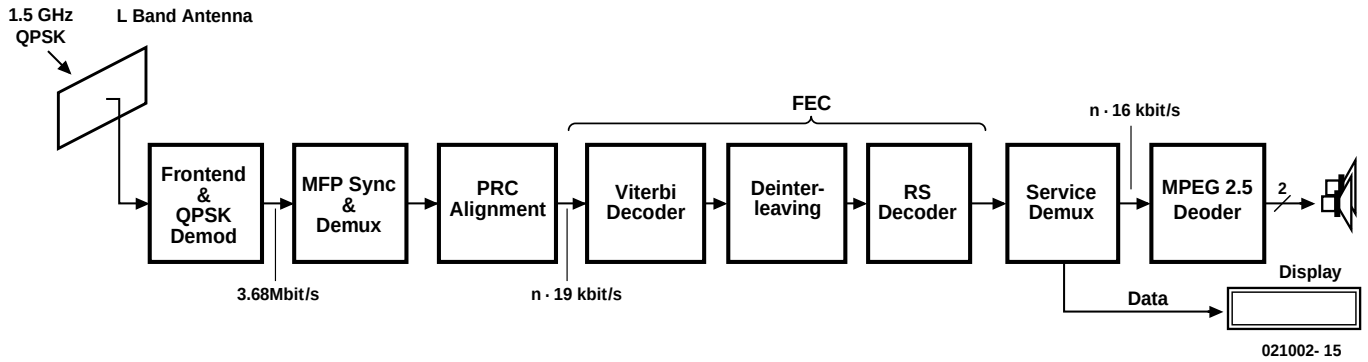


Figure 5. Block diagram of a WorldSpace receiver.

using four PRCs, a mono audio channel using two PRCs and a data channel using two PRCs. The stereo audio channel, the mono audio channel and the data channel thus form the three Service Components of the Broadcast Channel. These can be selected at the receiver at the press of a button.

Receiver

A WorldSpace receiver, as shown in **Figure 5**, includes a flat, or patch, antenna which is not highly directional. Its beamwidth is large, around 70 to eighty degrees both horizontally and vertically. The antenna therefore does not need to be particularly accurately aligned, as is the case with previous geostationary satellite systems.

The HF unit converts the 1.5 GHz signal down to a lower intermediate frequency. This front-end is in a double superheterodyne configuration: the 1.5 GHz signal is first converted down to a first intermediate frequency (IF) of, for example, 116 MHz. After bandpass filtering at this IF the signal passes to a further mixer, which converts the WorldSpace QPSK signal down to a very low second IF of 1.84 MHz. This is the lowpass filtered and converted to digital with an A/D converter. There follows a digital QPSK demodulator whose output is the 3.68 Mbit/s data stream. As described above, this data stream contains all the programmes in one beam in a time multiplex format. Subsequent processing is done in digital signal processors (DSPs) or application-specific ICs, known as the Starman WorldSpace Radio chipset. These originate from Micronas (Germany) and ST Microelectronics (France).

To assist in synchronisation there is a special word in the data stream called the Master Frame Preamble (MFP). Since this is fixed and unambiguous, it can also be used to resolve the phase ambiguity of the QPSK demodulator, which can lock on to the received signal with two different phase offsets. Once the demodulator is synchronised, the receiver can then accept the Time Slot Control Channel (TSCC) information, which forms part of the overall data stream. The extracted information gives the allocation of the PRCs to the various Broadcast Channels. The receiver stores this list in RAM to allow access at any time when selecting stations. This also allows quicker switching between stations.

In the Prime Rate Channel Alignment stage the PRCs that belong to the Broadcast Channel that has been selected are extracted from the overall data stream with the aid of the TSCC information. The microcontroller in the receiver tells the WorldSpace receiver chip which PRCs are required. By concatenating the selected PRCs a new data stream is constructed, which only contains the data for the chosen Broadcast Channel (BC). This now has a data rate of n times 19 kbit/s, where n is the number of PRCs used, from one to eight.

Next follow the three FEC stages required on the receiver side: the Viterbi decoder is the counterpart of the convolutional encoder and provides a first level of error correction. It tracks the transition from one symbol state (two bits) to the next and, knowing the coding algorithm used in the convolutional encoder, calculates the probability that the received transition could occur. If an alternative transition is seen to be more likely than the one received, the Viterbi decoder replaces the received one with the more likely one. These likelihood calculations are carried out continuously over a data window of 64 bits.

The deinterleaver distributes errors that affect consecutive bytes into separate positions spread out in time. The Reed-Solomon decoder that immediately follows can now detect and correct these burst errors.

The service demultiplexer takes the error-free data stream obtained and extracts the desired programme in the form of the n 16 kbit/s packets originally fed in, and sends them to the MPEG 2.5 decoder. At the same time the side channel data are extracted, and these are converted into a suitable form for display by the processor in the receiver. All WorldSpace receivers have a stereo headphone connection, and some also sport stereo loudspeakers. Very compact receivers must of course make do without stereo loudspeakers.

Because the error correction mechanisms behind WorldSpace radio are so cunning, reception is possible with a signal-to-noise ratio of only around 4 dB. The threshold for FM reception, as used for reception of analogue TV (from Astra etc.) lies at a signal-to-noise ratio of around 10 to 12 dB. Beneath this FM threshold the analogue TV picture starts to 'sparkle': brief flashes appear in the lines of the picture, which become more significant and hence more disturbing as the signal-to-noise ratio decreases.

Further, the receiver circuits in WorldSpace receivers are already designed with the nonlinearity of the travelling wave tube amplifier (TWTA) in the satellite transponder in mind. This nonlinearity can be compensated for, which improves still further the reception quality and the communication power budget.

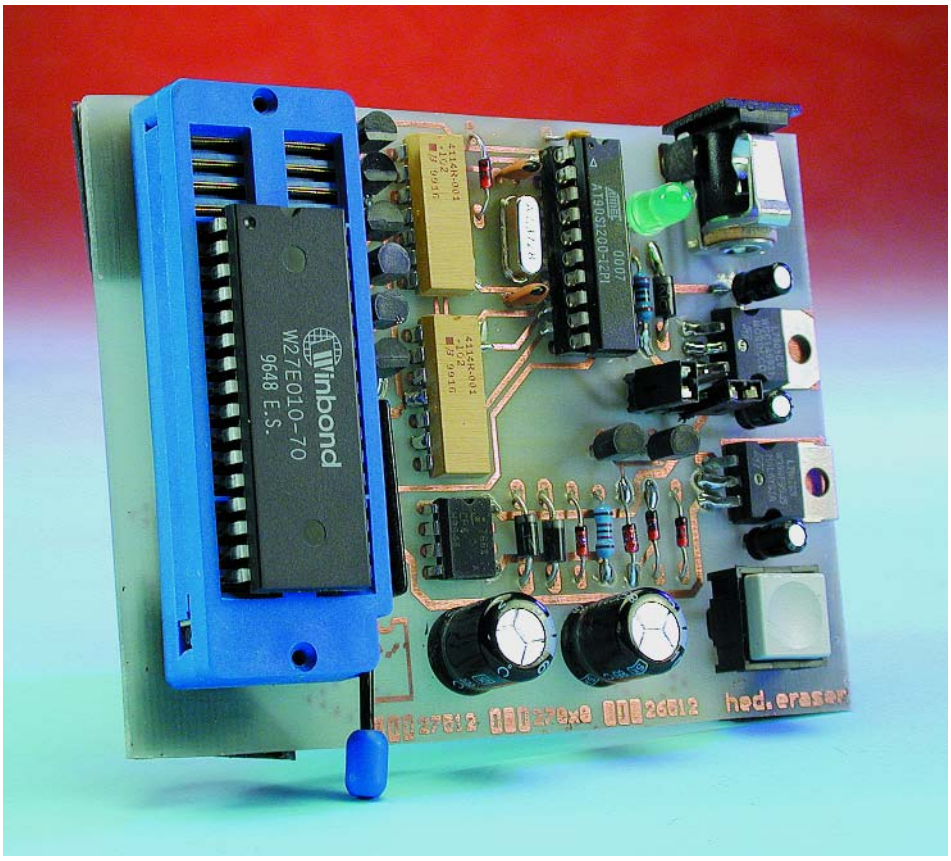
(021002-1)

EEPROM Eraser

V_{pp} instead of UV light

By P. Höpping and K. Gossner (Höpping Electronic Design) paul@hed.de

Although you will stumble over programmers for EEPROMs, matching eraser units are few and far between. The small 'hed.eraser' solves the problem quickly and efficiently.



Even after more than ten years, various Flash technologies have failed to completely supplant EPROMs. This is surprising because erasing an EPROM using a dose of UV light is tedious, while the glass window in the housing of EPROMs adds considerably to the device manufacturing cost. The Electrically erasable EPROM (EEPROM) is an attractive alternative. Just like the EPROM, it can be 'burned' in traditional programmer systems. Moreover, it is 100% electrically compatible with UV erasable counterparts. However, not all programming systems for EPROMs are capable of actually erasing an EEPROM, and that is why such a tool is presented in this article.

The EEPROM Eraser from HED, Germany is not a regular Elektor project in that it is only available from the authors, either as a ready-built unit or as a kit (see inset).

Erasure algorithm

An EEPROM is erased by applying a number of different voltages to different pins, and in a specific order! The algorithm itself is not complicated and consists of the following steps:

1. Apply 5-V supply voltage;
2. pull $\overline{CE}$, $\overline{OE}$ and $\overline{PGM}$ to +5 V;
3. apply programming voltage to A9 and V_{pp} pins;
4. apply erasure pulse to $\overline{CE}$ or $\overline{PCM}$;
5. switch off programming voltage;
6. switch off supply voltage.

Circuit description

An inexpensive, small, microcontroller from Atmel ensures the correct sequencing of signals while observing the required pulse durations. A couple of jumpers are used to set up the configuration required for the EEPROM to be erased.

The rest of the circuit serves mainly to generate the required voltages. Here, we will require a 5-volt supply voltage and an adjustable programming voltage in the range 12 to 14 volts.

The unstabilized supply voltage comes from a battery eliminator. We recommend one because it is less costly and safer to use than a dedicated mains transformer. However, some extra circuitry is required in this case to enable stabilized voltages greater than 12 V to be gener-

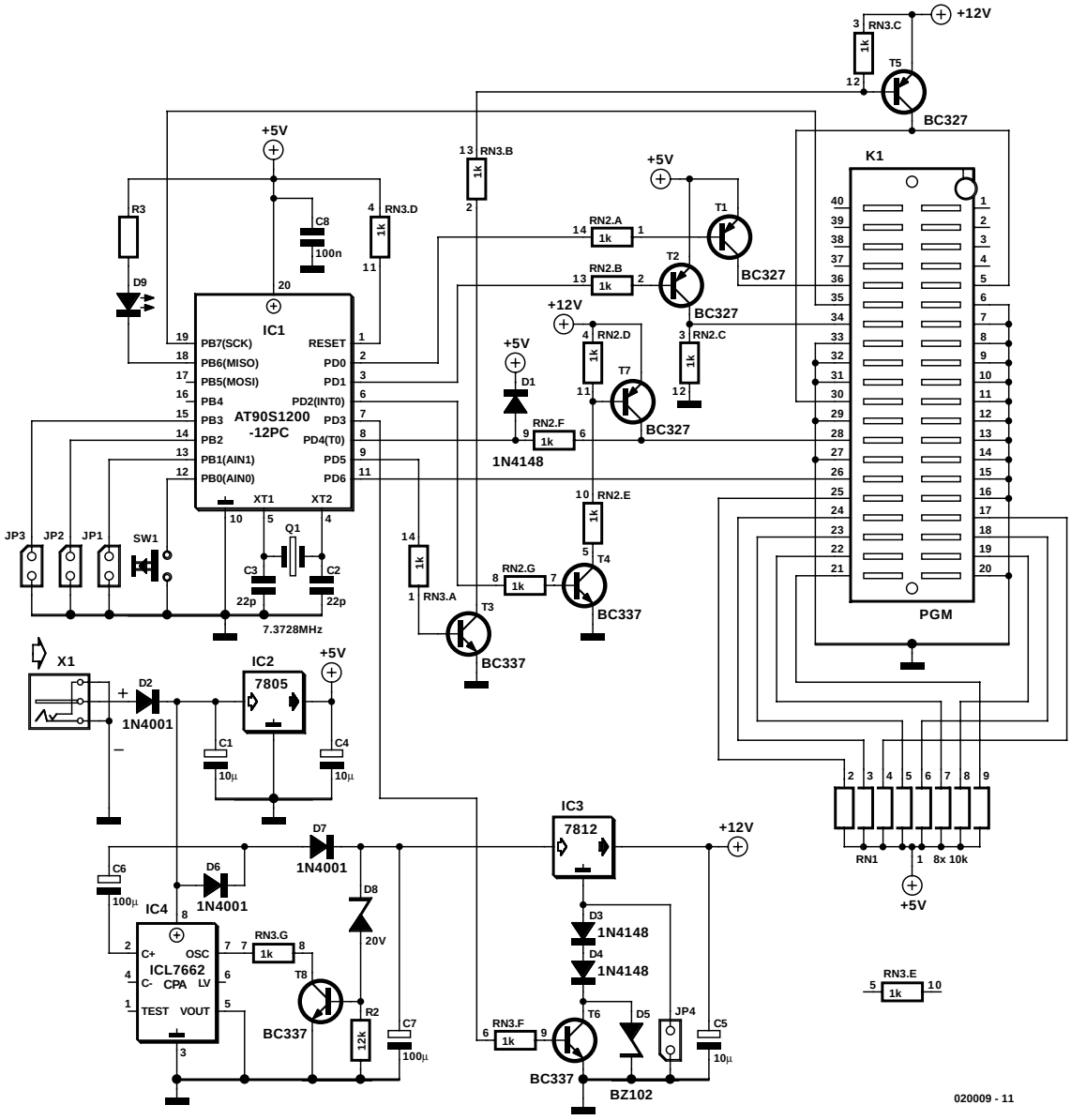


Figure 1. Circuit diagram of the hed.eraser for EEPROMs (board only available from HED, Germany).

ated whilst relying on a cheap mains adapter.

On the eraser board, the input voltage is doubled with the aid of a ICL7662CPA. Because the actual output voltage of a cheap mains adaptor is unpredictable to say the least, it makes sense to add a 20-volts voltage limiter. This is achieved with the aid of zener diode, D8, resistors R2 and transistor T8. The resulting 'raw' voltage has a maximum variation of about 1 volt and can be loaded with 25 mA. If the voltage doubler output voltage rises above the zener voltage, T8 is switched hard, causing the oscillator in IC4 to be stopped. When the voltage returns to the safe range, T8 is switched off and the oscillator is enabled again. By choosing a suitable zener diode, any voltage can be defined within a certain range.

The 'fine' adjustment of the programming voltage is handled by a 7812 voltage regulator. Three diodes, D3, D4 and D5, jack up the 7812 output voltage to three discrete levels between about 12.75 V and 14 V. Although the BZ102 diode may be found under 'Zener Diodes' in your favourite electronics catalogue, in fact, it consists of two ordinary diodes connected in series. And they can be replaced by the same! With EEPROMs requiring exactly 12.0 V for their programming voltage, the diodes may be shorted out by fitting JP4.

Practical use

Building the circuit on the printed circuit board shown in Figure 2 is a breeze and you should be able to complete it in no time at all. One of the few things to note while building the hed.eraser is that the resistor arrays, including RN1, are polarised parts so they have to be fitted the right way around.

Having finished the soldering work, the mains adaptor is set to 12 V output voltage and then connected to the socket on the hed.eraser board (+ on centre pin). Diode D2 protects the eraser against polarity reversal. The microcontroller should be left off the

What about the jumpers?

Depending on the EEPROM you need to erase, two jumpers have to be distributed across four positions.
JP1 and JP2 select the case type, while JP3 and JP4 are used to set the programming voltage level, V_{pp}.

Case	JP1	JP2
DIP28	fitted	open
DIP32	open	fitted

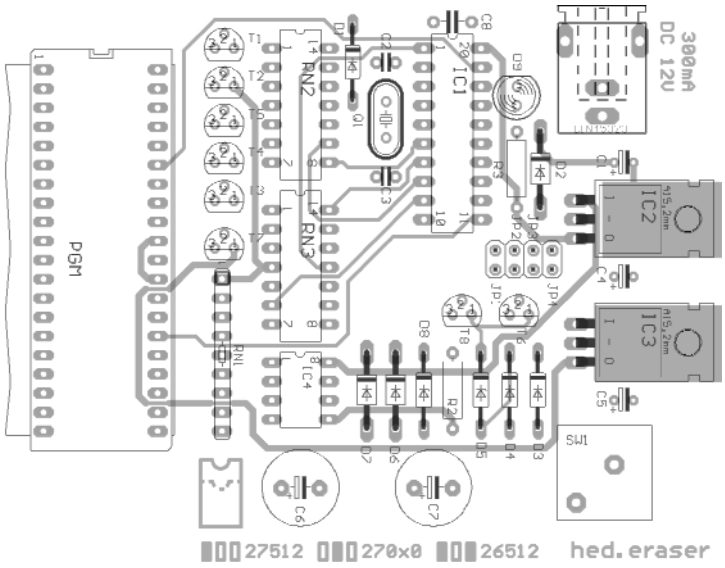
Vpp	JP3	JP4
12 V	open	fitted
12,5 V	fitted	open
14 V	open	open

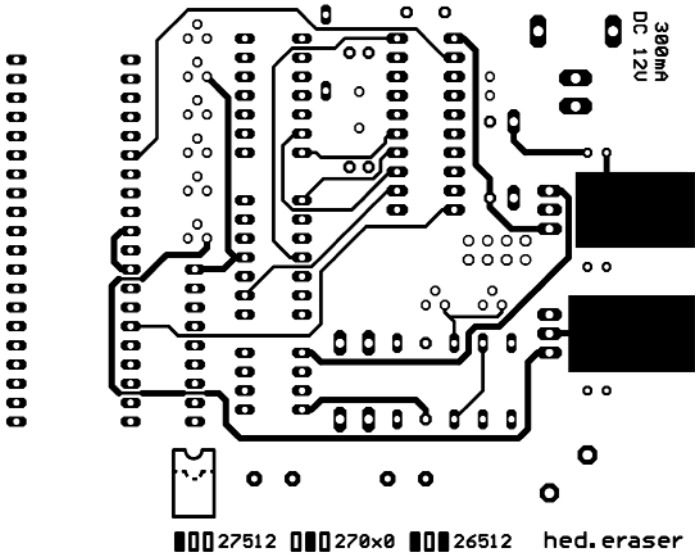
HED also offers two adapters to allow EEPROMs in 32-way PLCC cases to be erased. Note however that SMD components of which the DIL counterpart has 32 pins (for instance, the 27C0x0), need adapter type 'PLCC32', while EEPROMs (like the 27C512) whose PLCC counterpart has 32 pins, but the DIL version, just 28 pins, have to make do with the 'PLCC32_28' adapter.

Suggested supplier

The PCB layout file (.pdf) and the software for the microcontroller (source code and HEX files) are available from this month's Free Downloads section on the Elektor Electronics website.
The author's website supplies the PCB layout as a GIF file, the hed.eraser documentation as a pdf file, as well as the source code and binaries for the microcontroller. The website also allows you to order
– a partial kit containing the programmed AT90S1200, the ICL7662CPA and the three resistor arrays
– a ready-built and tested EEPROM Eraser.

Höpping Elektronik Design
Mr. Paul Höpping
Kantorie 97
D-45134 Essen
Germany.
Tel.: (+49) 201 843331
Fax: (+49) 201 471918
Internet: www.hed.de
Email: paul@hed.de





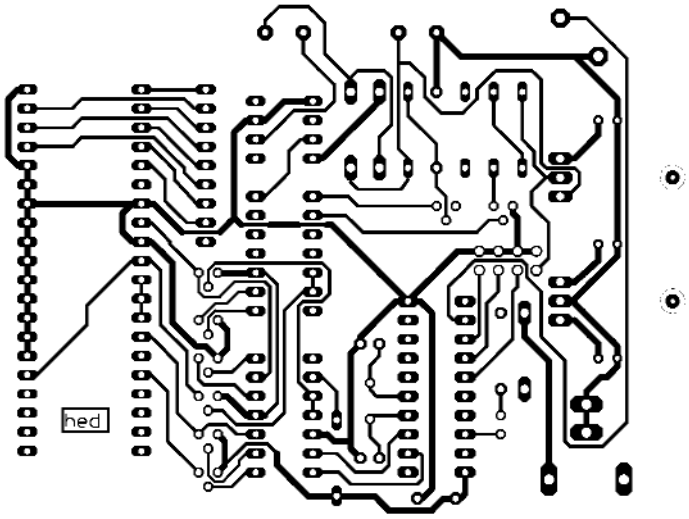


Figure 2. Copper track layout and component mounting plan of the double-sided PCB (board supplied by HED Germany).

COMPONENTS LIST

Resistors:
R2 = 12kΩ
R3 = 270Ω
RN1 = 8-way 10kΩ SIL array
RN2,RN3 = 7-way 1kΩ, DIL array

Capacitors:
C1,C4,C5 = 10µF 25V (lead pitch 2.5mm)
C2,C3 = 100nF (lead pitch 5mm)
C6,C7 = 22pF (ceramic)
C8 = 100µF 25V (lead pitch 5mm)

Semiconductors:
D1,D3,D4 = 1N4148
D2,D6,D7 = 1N4001
D5 = BZ102 (zener diode 1.4V, see text)
D8 = zener diode 20V, 500mW
D9 = LED, 5mm, green
T1,T2,T5,T7 = BC327-40
T3,T4,T6,T8 = BC337-40
IC1 = AT90S1200-8PC (programmed)
IC2 = 7805
IC3 = 7812
IC4 = ICL7662CPA (Intersil)

Miscellaneous:
X1 = 7.3MHz quartz crystal
8-way pinheader with 2 jumpers
S1 = pushbutton, 1 make contact (ITW)
2-way IC socket
40-way ZIF socket
K1 = mains adaptor socket, PCB mount

board while running a quick test of the supply voltages. The following can be measured against ground:

- Socket IC1, pin 20: +5 V ±0.1 V
- 7812 input , pin 1: +20V ±1 V
- 7812 output, pin 3: +14 V ±0.5 V

If this appears to be in order, the supply voltage may be removed and the microcontroller inserted in its socket. If the circuit is powered up again, the green LED should flash twice, and then remain on. This is a sign that the microcontroller is working and the hed.eraser is ready for use.

The hed.eraser is 'matched' to the EEPROM to be handled by setting a few jumpers (**Table 1**). Once the jumpers have been set to the required configuration, the EEPROM may be inserted in the ZIF socket and the 'erase' button may be pressed. The LED will be off during the erasure process, which lasts a couple of seconds.

(020009-1)

Useful literature:

ICL7662CPA Datasheets
www.intersil.com/data/fn/fn3/fn3181/fn3181.pdf

Electrically Erasable PROM (EEPROM) datasheets
SST: www.ssti.com (SST27SFxxx Series)
Winbond: www.winbond.com.tw, go to 'Products' then 'EPROMs'

Microcontroller Dice

using the 89S8252 Flash Board

Design by b. Kainka

Writing a microcontroller program and implementing it in real hardware simply requires a development system. The *Elektor Electronics* 89S8252 Flash board is just the ticket, even if you intend to write code for a target system based on a 'lesser' controller like the AT89C2051.

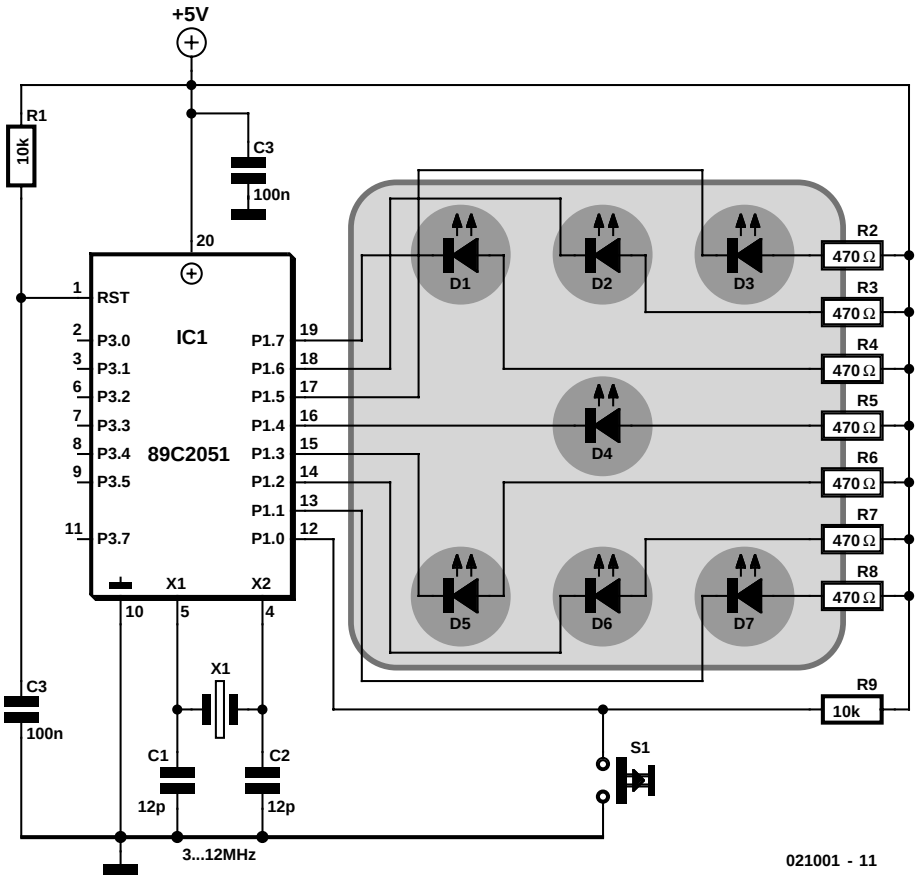


Figure 1. Circuit diagram and connections of the dice circuit to the 89S8252 Flash board.

In this article, our assignment will be to design electronic dice based on a microcontroller type AT89C2051. Admittedly, not a particularly exciting end product, but the assignment is not too complex and so forms a useful example for those that need to get to develop programming skills. Once the basics are mastered, the steps involved in writing code are easily applied to more serious applications.

Hardware

To begin with, LEDs that form the 'face' of the dice have to be connected to the microcontroller. Here, seven LEDs are used. As shown in **Figure** 1, they are connected to the seven connections of port P1. When high-efficiency LED are used, the value of the series resistors may be increased to 470 Ω. As discussed in the Microcontroller Basics Course, LEDs are connected to Vcc via a current limiting resistor. The remaining port connection is used to monitor a pushbutton for activity.

This experimental circuit is either fitted directly on the solder area of the 89S8252 Flash board (December 2001), or on a small dedicated (strip-) board and then connected to the Flash board using wires and connectors.

Listing 1. Output bit pattern allocation

```
;microcontroller dice
;
;  P1.7          P1.3
;
;  P1.6    P1.4  P1.2
;
;  P1.5          P1.1
;
;       P1.0 = key

          mov   a,#00010001b ;output "6"
          mov   P1,a
          mov   a,#01000101b ;output "5"
          mov   P1,a
          mov   a,#01010101b ;output "4"
          mov   P1,a
          mov   a,#01101101b ;output "3"
          mov   P1,a
          mov   a,#11010111b ;output "2"
          mov   P1,a
          mov   a,#11101111b ;output "1"
          mov   P1,a
```

That's all there is in the hardware department and we can now concentrate on developing the software to roll the dice!

Software development

'Assembler' was chosen as the development tool because it is just right for this rather simple application. Moreover, a small program in assembler obviates the need for RAM, allowing the finished program to run happily on smaller controllers.

The program is best developed in small steps. To begin with, we write the bit patterns for the driving of the LEDs. Here, clarity dictates the use of binary notation, allowing the allocation of the seven LEDs to the individual port connections to remain uncluttered. Each LED that's on represents a '0'. Bit 0, however, has to remain at '1' because Port P1.0 is used as an input to read the pushbutton state (**Listing** 1).

A wait loop is required to check the correct operation of the LED driver, as well as for the final program. Here, a subroutine `delay` is employed to provide the necessary idle time. The amount of delay is transferred to the accumulator (**Listing 2**).

In practice, the delay proved too short to enable our eyes to actually see what the program is doing. Consequently, a second subroutine, `delay2`, was created, which waits slightly longer than a second. `Delay2` calls `delay` several times (**Listing 3**) and should itself be called at a suitable pint in the main program. A small test program (Listing 4) is now able to output all bit patterns (representing numbers on the dice) slowly and in sequence, allowing us to verify correct connection of the hardware. Any errors in the previously defined bit patterns should be blatantly obvious now.

Push to roll...

The 'roll' pushbutton is scanned for activity in two loops. As longs the button is not pressed, the program should simply wait and idle away. The moment the pushbutton is pressed, the counter state that represents the dice value should be automatically incremented. Here, register r1 is used to hold the counter state. However, r1 may only contain values 1 through 6 so we need to check if state '7' is reached. If so, the counter has to be reset to state '1'.

The decisive function of the program consists of assigning the individual bit patterns to the counter states. This is achieved with comparisons and jump instructions. The instruction `cjne` (compare and jump if not equal) does it all. In rapid succession, the contents of register r1 are compared to values 1 through 6. If a match is found, the associated function is performed. A 'non-equal' outcome causes the program to branch to the next comparison. Using this allocation, a simple dice program may be written, as shown in **Listing 5**.

In principle, we've completed the assignment. The die works! Each time the button is pressed, a rapidly changing readout is produced that stops at a number between 1 and 6.

Here, the program development was shown to consist of three steps. In practice, however, more than 20 steps were required. Each time, the source code text had to be

Listing 2. Creating a delay

```
delay     mov   r7,a
delay0    mov   r6,#255
delay1    djnz  r6,delay1
          djnz  r7,delay0
          ret
```

Listing 3. Longer delay.

```
main      acall  delay2
out6      mov   a,#00010001b
          mov   P1,a
out5      acall delay2
          mov   a,#01000101b
          mov   P1,a
out4      acall delay2
          mov   a,#01010101b
          mov   P1,a
out3      acall delay2
          mov   a,#01101101b
          mov   P1,a
out2      acall delay2
          mov   a,#11010111b
          mov   P1,a
out1      acall delay2
          mov   a,#11101111b
          mov   P1,a
          sjmp  main
```

Listing 4. Successive dice patterns

```
delay2    mov   a,#255
          acall delay
          acall delay
          acall delay
          acall delay
          ret
```

changed, the program compiled and copied into the microcontroller. Why so many times?

Well, because small errors will always creep in. The phased development of a program should result in a feeling best described as slowly crawling to your target. The 89S8252 Flash board considerably lightens this strategy of *rapid prototyping*. Using many small steps, trial and error will eventually produce the desired result. Each time, testing a newly written part of the program requires only a few seconds. You may remember them fondly and with nostalgia, but the times when EPROMs had to be erased and re-programmed to eliminate any small error are long gone.

Practical tip

The LEDs in the dice circuit you've built not only indicate the operation of the running program, but also the download process. That happens because the higher-value port connections P1.7, P1.6 and P1.5 are also involved in the processor programming process. The download tool uses a protocol on these lines to allow serial programming via the SPI interface. The pulses that occur during serial transfers become visible in the relevant LEDs. Very useful, actually, because it allows you to see if a download operation is in progress and successful.

The rudimentary software does not, however, provide feedback about the result, so that at least three errors may 'bug' you:

1. Supply voltage not applied to Flash board.

2. Communication ports COM1 and COM2 ports interchanged.

3. Communication port used is erroneously used by another program.

The dice LEDs will faithfully indicate that everything is okay as far as the program download process is concerned. This 'luxury' is worthy of building into other projects as well. That's why we propose to fit two LEDs with suitable series resistors in the prototyping area of the Flash board. One indicates the presence of the supply voltage, the other any signals on P1.7. In this way, you always have a clear indication of what's happening.

Another tip

After booting, Windows ME reportedly fails to clear the High on the DTR line of the PC's serial ports. This may cause problems with the Flash board which is held in the rest state.

Listing 5. The first actually working dice.

```
;dice1.asm

;microcontroller dice
;
;  P1.7          P1.3
;
;  P1.6    P1.4  P1.2
;
;  P1.5          P1.1
;
;       P1.0 = key
;

#include 8051.H
          .org 0000H


main      mov   a,#0FFh
          mov   P1,a       ;all leds off
          mov   r1,#1      ;first number
loop      jb    P1.0,loop ;wait for key
key       mov   a,r1       ;r1 = memory
          inc   a
          mov   r1,a
          cjne  r1,#7,out
          mov   r1,#1
out       cjne  r1,#6,out5
          mov   a,#00010001b
          mov   P1,a
out5      cjne  r1,#5,out4
          mov   a,#01000101b
          mov   P1,a
out4      cjne  r1,#4,out3
          mov   a,#01010101b
          mov   P1,a
out3      cjne  r1,#3,out2
          mov   a,#01101101b
          mov   P1,a
out2      cjne  r1,#2,out1
          mov   a,#11010111b
          mov   P1,a
out1      cjne  r1,#1,out0
          mov   a,#11101111b
          mov   P1,a
out0      jnb   P1.0,key
          sjmp  loop

          .end
```

Listing 6. Improved program featuring slow-roll imitation.

```
;dice2.asm

;microcontroller dice
;
;  P1.7          P1.3
;
;  P1.6    P1.4  P1.2
;
;  P1.5          P1.1
;
;       P1.0 = key
;

#include 8051.H
          .org 0000H
```

```
main    mov   a,#0FFh
        mov   P1,a          ;all leds off
        mov   r1,#1         ;first number
loop    jb    P1.0,loop     ;wait for key
key     acall play          ;r1=r1+r1
        mov   a,#01h
        mov   P1,a          ;all leds on
        mov   a,#5
        acall delay
        jnb   P1.0,key

        acall play          ;roll
        mov   a,#10
        acall delay
        acall play
        mov   a,#25         ;slower
        acall delay
        acall play
        mov   a,#50
        acall delay         ;slower
        acall play
        mov   a,#100
        acall delay         ;slower
        acall play
        mov   a,#150
        acall delay         ;slower
        acall play
        mov   a,#200
        acall delay         ;slower
        acall play
        mov   a,#250
        acall delay         ;slower
        acall play
        mov   a,#250
        acall delay         ;last
        acall delay
        acall play          ;stop
        sjmp  loop

play    mov   a,r1          ;r1 = memory
        inc   a
        mov   r1,a          ;r1=r1+1
        cjne  r1,#7,out
        mov   r1,#1         ;reset to 1
out     cjne  r1,#6,out5
        mov   a,#00010001b ;output "6"
        mov   P1,a
out5    cjne  r1,#5,out4
        mov   a,#01000101b ;output "5"
        mov   P1,a
out4    cjne  r1,#4,out3
        mov   a,#01010101b ;output "4"
        mov   P1,a
out3    cjne  r1,#3,out2
        mov   a,#01101101b ;output "3"
        mov   P1,a
out2    cjne  r1,#2,out1
        mov   a,#11010111b ;output "2"
        mov   P1,a
out1    cjne  r1,#1,out0
        mov   a,#11101111b ;output "1"
        mov   P1,a
out0    ret


delay   mov   r7,a
delay1  mov   r6,#255
delay2  djnz  r6,delay2
        djnz  r7,delay1
        ret

        .end
```

The solution to the problem is to either disconnect the Flash board from the interface, or run the Flash download tool on the PC. The latter clears the DTR line to Low.

Improvements

As long as the dice runs on the Flash development board there's room for improvements to the program. Real dice never stop as sudden as our electronic equivalent, so one suggestion would be to mimic the 'roll' action slowing down gradually. Here, the effect is achieved by added software functionality. The dice are rolled in the routine `play`, while everything to do with waiting takes place in `delay`. When the button is released, the die 'rolls on' for six more states, slowing down each time a new number is reached. A great improvement, we'd say, towards more realistic behaviour of our little circuit.

While developing the program, it was noticed that the LEDs flicker somewhat unsteadily when the pushbutton is being pressed. This cosmetic problem was solved by making all seven LEDs light while the button is pressed.

Listing 6 shows the penultimate version of the program, ready for downloading to a small microcontroller system. By 'penultimate', we do not mean that there will be a follow-up article. Rather, you are encouraged to improve the program yourself, which will certainly present a challenge.

Target system construction

In order to transfer the functionality of the microcontroller dice to a standalone circuit, the hardware has to be defined. Well, besides the components already mentioned, not much more is required than a controller, a quartz crystal and a reset capacitor. Basically, that's all you need to pencil down in the way of a circuit diagram.

An important difference between the controllers we've in mind should be mentioned, however: port lines P1.0 and P1.1 of the AT89C2051 do not have internal pull-up resistors. The reason: they can be also be used as analogue comparator inputs. For LED control, that is not a problem. A pushbutton, however, does require an external pull-up resistor, which is not found in the development system.

The program you have managed to develop is copied into the smaller microcontroller using a suitable programmer. Next, everything is soldered together and tested.

(021001-1)

Data Monitor for Märklin Model Railway Systems

pulse packets on the PC monitor

Design by Nils Körber DHØHAN

www.koerber-home.de

Regardless of whether they're generated by DELTA Control, Control Unit, Digital Railrunner, EEDTS Pro or something else, digital locomotive control signals in the 'Motorola format' all look pretty much alike on the rails. If you want to view these signals live on a PC monitor, you need the data monitor described in this article.

The Märklin digital control protocol has already been described a good number of times (for instance, in the May 1999 issue of *Elektor Electronics*). We can thus limit our description of the theory to the bare essentials. The data are transmitted in the form of pulse packets consisting of 18 individual pulses or 9 pulse pairs. The voltage levels on the rails that are used to transmit the pulses are approximately –15 V and +15 V, with the exact values varying according to the control unit used and the degree to which it is loaded by the connected devices.

Depending on the situation, an individual pulse is interpreted as a bit, while a pulse pair (consisting of two individual pulses) is interpreted as a 'trit' (trinary bit). The associated logical value is encoded by the duty factor of the signals or the pulse width of a particular pulse. In the digital data stream on the rails, the bits and trits have the following forms:

bit '0'
26 µs at +15 V, 182 µs at –15 V

bit '1'
182 µs at +15 V, 26 µs at –15 V

trit '0'
26 µs at +15 V, 182 µs at –15 V
26 µs at +15 V, 182 µs at –15 V

trit '1'
182 µs at +15 V, 26 µs at –15 V
182 µs at +15 V, 26 µs at –15 V

trit 'open'
182 µs at +15 V, 26 µs at –15 V
26 µs at +15 V, 182 µs at –15 V

In the old Motorola format, a complete pulse packet consists of four trits for the locomotive address, one trit for the 'Function' function and four trits for the speed, including changing direction. In the new Motorola format, the last four trits are replaced by eight bits for the speed, directional information and functions f1, f2, f3 and f4. The pulse packets thus have the same lengths in both formats, namely approximately 3.75 ms.

For the Data Monitor, the signal from the booster that is present on the rails is connected to the serial interface of the PC after undergoing a level adjustment. The Data Monitor program then samples the signal present at the serial interface at quite short intervals for the duration of the measurement interval (several seconds) and stores the results. Following the completion of the measurement, the measured values are evaluated and can then be displayed in detail on the PC screen.

The quality of the measurement is entirely dependent on the frequency and constancy of the sampling rate. Certain things must be taken into account in this process (as discussed further under 'Software'). With a constant sampling rate, at least two samples should fall within a relatively short pulse width of 26 µs in order to obtain a reasonable measurement result. This leads to a minimum sampling rate of around 83.5 kHz. Such a sampling rate can easily be achieved with a modern PC, even under Windows. On the author's PC, which has an Intel Celeron processor clocked at 466 MHz, a sampling rate of around 350 kHz was achieved.

The hardware: a voltage divider

The signal present on the rails is connected to the serial interface of a

PC. Since voltages greater than 15 V can also be present on the rails, a voltage divider (**Figure 1**) consisting of two 4.7-kΩ resistors ensures that the levels arriving at the interface lie securely within the range specified by the RS232C/V.28 standard.

At the serial interface, the RI (Ring Indicator) and GND (Ground) leads are used. On a 25-way sub-D plug, RI is located on pin 22 and GND on pin 7, while on a 9-way sub-D plug RI is assigned to pin 9 and GND to pin 5. The two-wire cable to the PC may be relatively long without causing any problems, but you should be careful to maintain the correct polarity. In the Märklin scheme, terminal 'B' is connected to the middle pickup rail.

Please also note that due to the connection to the PC, the motive power circuit of the model railway is no longer floating, but instead is tied to earth via the GND connection. If this is not compatible with other circuits in your model facility, they must first be isolated from the motive power circuit.

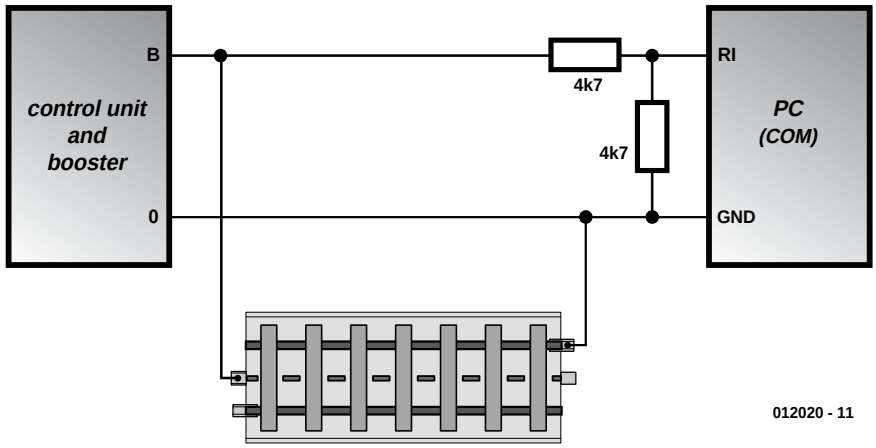


Figure 1. Connections and circuit diagram.

Software

The Data Monitor program has been generated using Microsoft Visual Basic 5. Access to the serial interface is provided by the PORT.DLL dynamic linked library, which regular readers of *Elektor Electronics* already know from other projects (and which can be downloaded from the *Elektor Electronics* website).

User interface

As can be seen from **Figure 2**, the user interface is divided into five regions, as follows:

– Settings

This can be used to select the serial interface port and specify the number of samples for the measurement.

– Measurement

The Start button starts a measurement. The Start Realtime button also starts a measurement but disables interrupts during the measurement in order to prevent interference from other Windows processes, thereby increasing the quality of the measurement and the sampling rate. While the measurement is being made, the Monitor Active check box displays a tick mark. In between measurements, the RI check box shows the level on the RI lead using periodic sampling.

– Evaluation

The various buttons are used to control the detailed display. The number of the first displayed sample is also shown here. One special feature is the Next Pulse Packet button. This button positions the detailed display to the beginning of the next pulse packet (more precisely, just before the beginning) and then evaluates the content of this pulse packet. The result of the evaluation is then shown in text form at the top of the Detail display.

– Summary Display

The complete measurement is shown here, with the samples combined into blocks. If a '1' is located within a block (this corresponds to the 15-V level), the block is displayed as active. A red line is also shown at the start of the current detail display location.

– Detail display

The Detail display shows the values of the individual samples in increasing sequence using four picture boxes. The first sample dis-

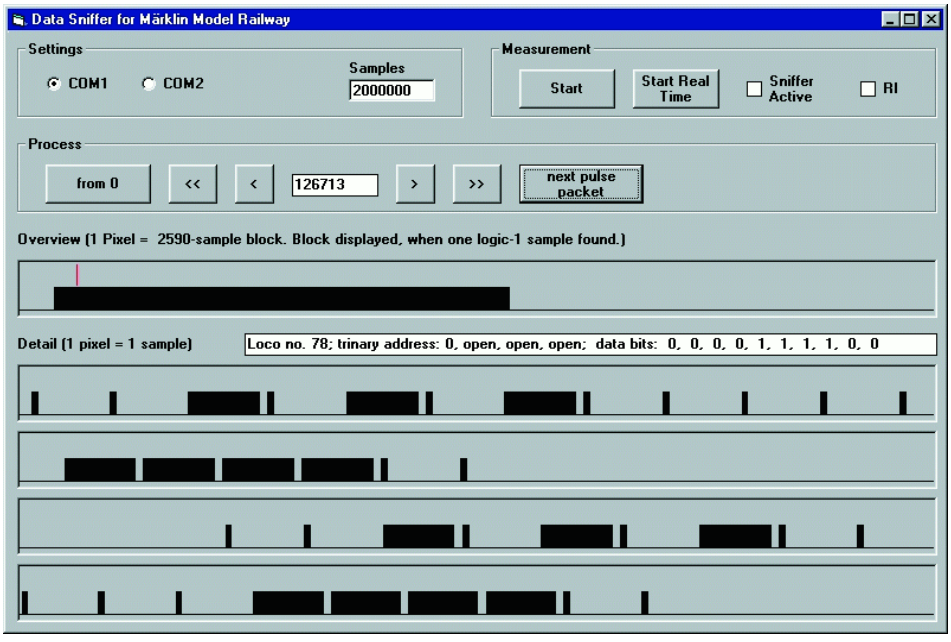


Figure 2. The user interface of the Data Monitor program, showing a pulse packet in the 'old' Motorola format for locomotive number 78.

played can be controlled using the Evaluation button in the controls region. There are always 3088 samples visible.

The figure shows a pulse packet in the old Motorola format for locomotive number 78. The displayed data bits (0000111100) or data trits (00110) mean that Function is 'off' and the Speed is '6'. The display begins with sample 130297 of a total of 2,000,000 samples. The entire measurement for this example took around six seconds.

Special features and system requirements

In order to obtain reliable measurements, the Start Realtime button should normally be used. This disables the maskable interrupts during the measurement interval and thus minimises disturbances by other Windows processes. During the measurement process, all other actions that create work for the PC should also be avoided as much as possible — for example, moving the mouse.

The Data Monitor is available as a complete VB5 setup. This means that it is not necessary to install Visual Basic on the computer before using the program. The software can be downloaded free of charge from the *Elektor Electronics* webserver or obtained on diskette from Readers Services (order number **010202-11**). This applies to both the source code as well as the compiled PC program. As a special feature, the program has been complied in system code (native code) rather than P-code. This results in an increased sampling rate.

The Data Monitor requires a PC with a free serial port (COM1 or COM2) and Microsoft Windows 95, 98, NT 4.0 or 2000 as the operating system. The program should run properly on all computers with a clock rate of around 200 MHz or greater.

The motive power circuit of the railway, or the output of the controller or booster, must either be floating with respect to earth or tied to earth via the track lead (not the middle pickup rail lead!).

(012020-1)

Literature and Internet addresses:

Digital Control for Model Trains, *Elektor Electronics*, May 1999.

Author's home page with other projects for model railways (including non-electronic projects): www.koerber-home.de

Souping Up Graphics Cards

more speed — by software

By Harry Baggen

To computer game addicts, the GPU and the graphics card in their PCs is invariably too slow. True, many of the latest games place heavy demands on the GPU and the graphics card as a whole. Following the introduction of nVidia's latest speed devil, the GeForce 4, souping up graphics cards has become a popular pastime again.



A long, long time ago (in computer terms, that is…), the sole function of the graphics card was to convert image data into a video format that could be viewed on a monitor. Such functionality now seems antediluvian. These days, graphics cards for home PCs sport a GPU (Graphics Processing Unit) that's more complex than the latest CPU on the motherboard. A lot of special functions and graphics processing is handled directly by the GPU, without any intervention from the CPU. In this way, it is possible to compute complete shapes, shadows, light effects, etc. at such a speed that smooth motion is perceived on the computer display. This is, of course, essential with many high-end computer games, which, not surprisingly, typically exploit the power of the modern GPUs to the full. Fond memories, suddenly, of IBM's mid-1980's 'Alley Cat' computer game which could be played using a super-VGA card, DOS, the standard PC loudspeaker and just 250 kB RAM…

The following factors contribute to the computing power of a graphics card: processor speed, memory access speed, and the extra functions squeezed into the GPU hardware (like the number of pipelines available for calculating geometrical shapes). The latter are, of course, beyond our grasp or even control and that is why the sheer computing power of the GeForce 4 can not be matched by its predecessors, the GeForce 2 or 3. Still, there's room for tuning, tweaking or souping up, or whatever computer nerds want to call it.

The GPU and memory clock frequencies are strictly separated and easily modified by writing appropriate data to a couple of graphics card

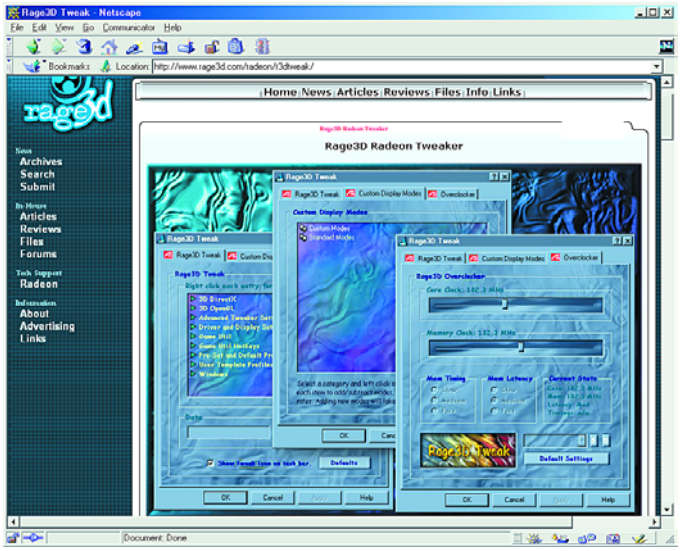
registers. No hardware mods required! For example, by means of smart software, the GPU clock may be increased from 220 MHz to 250 MHz. Depending on the available cooling capacity and the quality of the components on the graphics card, there's always some headroom which speed tweakers will love to put to the test. If you reach too high, your STOP sign is the screen which will show irregularities in the picture, or even complete absence of an image. In most cases, everything will be all right again if you step the clock down to a lower value. In rare cases, however, components on the graphics board may overheat, causing the whole board to give up the ghost and you ca write FUBR on it. Alas, that is the inevitable and well-known risk of overclocking!

As I write this there are only two major manufacturers of graphics processors with 3-D capability, and a handful of 'lesser' competitors in the market. Most games enthusiasts will have a graphics card in their PC based on an **nVidia** processor like the GeForce 2, 3 or 4, or one from **ATI** (Rage, Radeon, 7500, 8500). When searching for tuning tools for graphics cards, in practice the largest part of the market is easily accounted for by concentrating on these two manufacturers. That is not to say that no such tools are available for GPUs from smaller manufacturers, as we wil see further on.

Our survey of GPU overclocking tools should start with the familiar **Powerstrip** from the Taiwanese company Entech [1]. This program offers you a vast number of configuration options for the graphics card. It supports several displays and automatically recognizes the features of most graphics cards, from a Matrox Millennium I right up to the very latest Radeon 8500 and GeForce 4. Two slide controls allow you to set the GPU and memory speed separately. The program also allows individual profiles to be created for applications, monitors, colour correction and speed settings. An excellent feature of PowerStrip is that it's also suitable for newcomers to the art of overclocking!

Judging by the abundance of websites exclusively devoted to these cards, GeForce products are immensely popular and have many staunch supporters. Most of these websites have some kind of download section containing tweaking programs. Because all programs are nearly identical, we can limit ourselves to one of these sites. The **GeForce Zone** website [2] contains a tweaking page listing ten or so overclocking programs including NVMax, Coolbits and Rivatuner, editing tools for the graphics card BIOS and many other bits and bobs.

For ATI cards it's best to go straight to the **Rage3D** website [3]. There you will find an Aladdin's Cave of utilities for modern ATI GPUs (roughly anything from the Rage to the Radeon 8500). All the latest drivers and loads of utilities have been brought together on these pages. Under the heading 'Tech Support' you'll find various tweak utilities like Rage3D Tweak, UniTuner, Radeon Tweaker and RadTweak.





A very cool, stylish page offering a lot of overclocking tools is webbed by German **Grafikkarten Tools** [4]. One or more programs for all famous chips like Voodoo, S3, nVidia, Matrox, ATI and Kyro, may be found here. The same, basically, goes for the **Guru3D** website [5], although this also offers a mass of drivers and test utilities.

The highest speed that can be obtained from a graphics card is often determined by the capacity of the cooling device fitted on the GPU and/or the memory ICs. Intrepid tweakers will not hesitate to add extra cooling here and there, and so squeeze the highest possible performance from all that silicon on the graphics board. At **Tweak3D** [6] you'll find a rather elaborate description of how this may be done in practice.

(025034-1)

Internet Addresses:

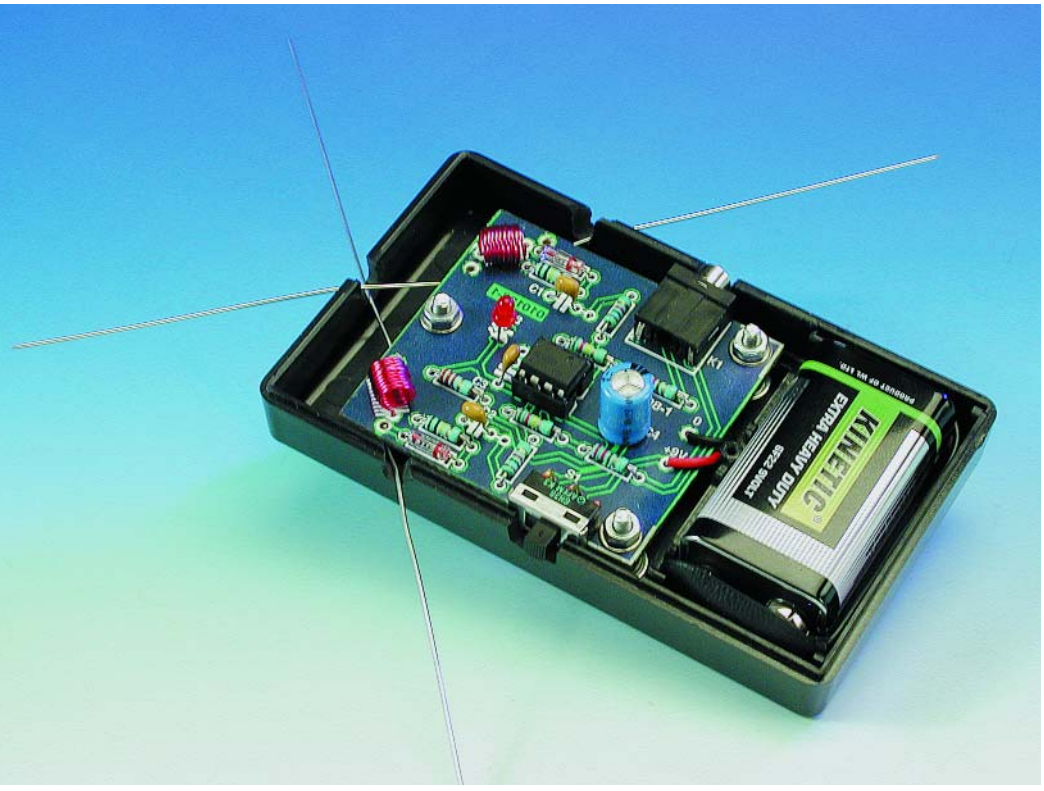
[1] Powerstrip (shareware): www.entechtaiwan.com/ps.htm
[2] GeForce Zone www.geforcezone.com/tweaks.htm
[3] Rage3D: www.rage3d.com/radeon/
[4] Grafikkarten Tools: www.grafik.overclocker-tools.de/
[5] Guru3D: www.guru3d.com/files/
[6] Tweak3D: GeForce3 cooling: www.tweak3d.net/reviews/misc/gf3cooling/

Mobile Phone Sniffer

track down mobile phones using this handy directional finder

Design by B. Kainka

In some situations and locations, the use of a mobile phone is forbidden. You may be discussing sensitive information in a closed meeting or holding important exams. How can you be sure that a concealed phone is not being used to pass information outside? Maybe you're just curious and would like to know if someone nearby is using a phone. This neat unit will not only detect the presence of mobile phone activity (voice or text), it can home in on the culprit!



If you are not familiar with the characteristic RF pulses sent out by a mobile cell phone just go over to an FM radio and switch the mobile on or off while holding it close to the radio aerial. These radio frequency (RF) pulses are an annoyance to amateur radio enthusiasts because they always seem to be picked up by sensitive input stages and you sometimes hear them on Outside Broadcast television programmes where people close to the presenter are using a mobile.

Detecting these pulses is not difficult; all you need is a small aerial, a diode detector and amplifier. Better still, why not double-up on the entire circuit, use directional aerials offset by 90° and feed the signals into a stereo headset to produce a directional detector for pin-pointing cell-phones!

Apart from mobile phones this unit can also pick-up the signals sent out by modern digital cordless (DECT) phones. These operate around the same frequency as cell phones. As with cell phones you will not be able to eavesdrop on conversations because the speech is digitally encoded but you will be able to hear the telltale RF pulses.

Microwave ovens operate in the SHF band at about 2,400 MHz and you can use this unit to check for RF leaks around the door seal. Adding a few turns to the two coils and increasing the antennae length will allow you to experiment at lower frequencies in the short wave radio spectrum (not in stereo however).

Elementary, my dear Watson

This circuit for the Sniffer unit (**Figure 1**) is quite simple. Looking at just one half of the circuit, the dipole antenna arms A1 and A2 form a resonant circuit with L1. Diode D1

Mini Project series

This new series is designed with the novice or younger reader in mind. The projects described will have an emphasis on low build cost and simplicity. The aim is to motivate and encourage self-build and experimentation rather than providing a heavy technical content. Along the way you will undoubtedly pick up useful technical knowledge and insights along with the fun.
All the components used in the projects should be widely available from the majority of the usual component suppliers.

The Elektor Electronics Mini Project series is also freely available on our website at www.elektor-electronics.co.uk. We hope this will prove not only a useful introduction to electronics but also stimulate new circuit design ideas from our readers. We want to provide the best possible service so go along to the website and tell us what you think is good and how it can be improved. We look forward to your comments and suggestions, especially if you have any circuit ideas yourself that would be suitable as a Mini Project feature. Do not hesitate to contact us.
E-Mail: editors@elektorelectronics.co.uk

detects the RF pulses and the signal is amplified by IC1.

A dual operational amplifier type LM358 forms the headphone amplifier. The 'power-on' LED D3 also provides a reference voltage for the virtual earth inputs for IC1. The output signal is direct coupled to the standard Walkman type headphones via R4; this means that the amplifier operates in class 'A' mode giving lower distortion even at high output signal levels.

The elements for each dipole antenna are made up of two lengths of wire from 3 to 7.5 cm long arranged in-line. These two antennae are fitted at 90° to each other at the corners of the PCB. This arrangement works rather like your ears and gives you spatial awareness of the transmitter direction (handy for pin-pointing a cellphone if many phones are sending simultaneously).

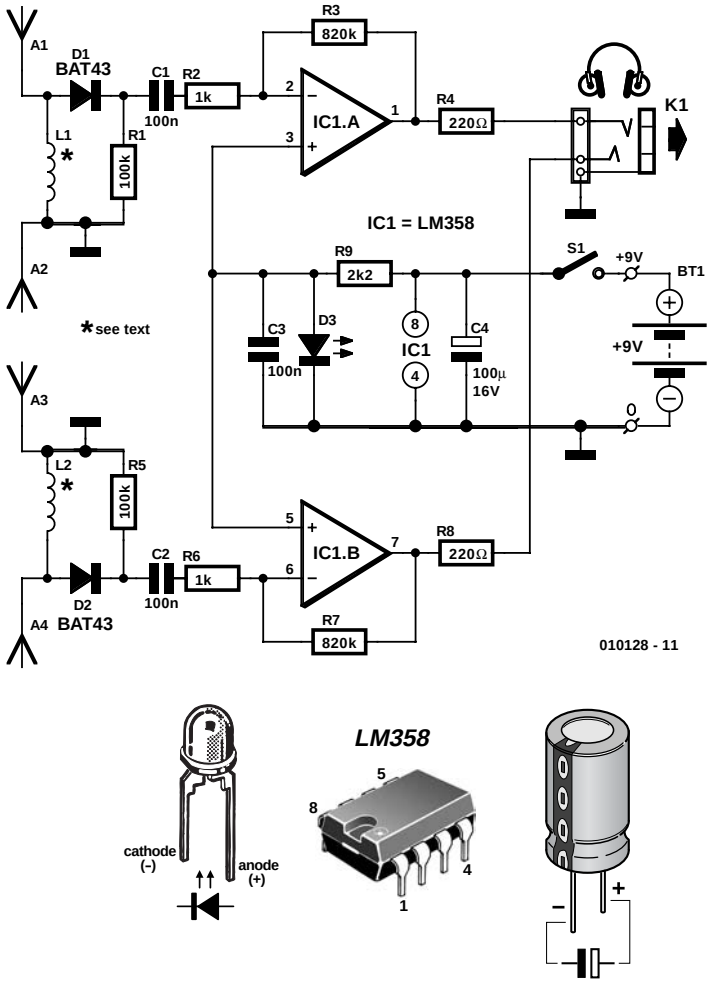


Figure 1. The Sniffer circuit diagram consists of stereo UHF receivers!

Construction

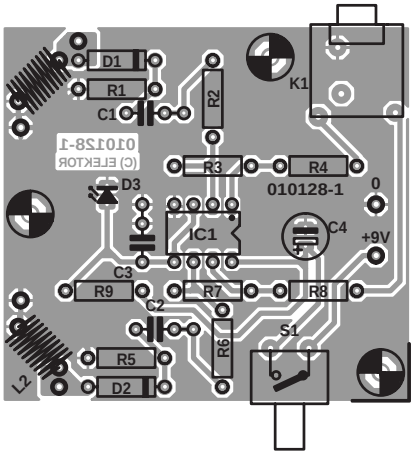
Detector diodes D1 and D2 are shown in the parts list as BAT43 or BAT45 Schottky diodes. Three alternative germanium diodes are also listed, and during tests these actually improved the Sniffer units sensitivity! The popular BAT85 Schottky diode is unsuitable in this application because it has poor high frequency characteristics. The construction of L1 and L2 is not critical. In the prototype, we wound the ten turns around a 5mm diameter LED before removing the LED and fitting the coils to the PCB. The on/off switch (S1) for the unit is connected to two pads on the PCB (**Figure 2**). If the switch is fitted with changeover contacts, the unused connections can simply be clipped off. A press-to-make push-button can be used for S1; this would ensure that you never accidentally leave the unit switched on.

Make certain that the battery connector is correctly connected to the PCB. The red cable will carry +9 V from the battery but if you are in any doubt check with a voltmeter. Fit all the components to the PCB but do not plug IC1 into its socket yet. Before connecting power to the Sniffer unit, double-check that all the polarised components are correctly fitted. If the LED lights when the battery is fitted, S1 is switched on and you are confident that all is in order, turn off the unit and fit IC1 carefully into its socket.

Fit the Sniffer unit into any suitable case that will accept the PCB and battery.

The phone Sniffer's sensitivity is dependent on three factors: The type of diodes used, the length of the antennae and the voltage gain of amplifiers IC1A and IC1B.

The antenna length is matched to the radio frequency of interest. GSM phones transmit at around 950 MHz so each arm of the dipole should be half the wavelength or 7.5 cm (3 cm for phones operating in the new 1800 MHz band). Try experimenting with aerial length. The wire type is not critical but its best to choose a fairly thick wire gauge that will not bend too easily. It's important to make strong soldered joints for the aerials so it is best to fit solder posts to the PCB aerial pads and solder the aerials to the posts.



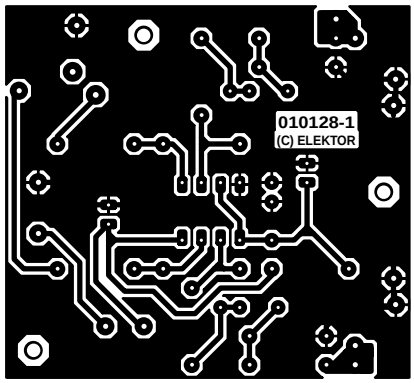


Figure 2. PCB layout and component placement.

Trying it out

Test the unit with a mobile phone. Turn the ringer off on the phone and then ring up the mobile from another phone. There is no need to answer the mobile because it emits RF pulses whenever it receives an incoming call request. Start off 1 to 2 metres away from the phone switch on the Sniffer and make sure that the bleeping sounds in the headset are equal volume when the Sniffer is pointing directly at the phone. Point the Sniffer off to the left of the phone and the bleeps in the left headphone should get softer. If the opposite occurs you have mixed up the wiring to the headphone socket. Don't panic, the problem can be fixed without even warming up the soldering iron: If it's not comfortable wearing the headphones round the other way, then just flip the Sniffer unit over and use it upside-down!

The range at which the Sniffer can detect a mobile is dependent on the RF power output of the phone. GSM phones alter their radiated power dynamically. The nearer the phone is to the base station, the less transmit power it will need to reach it. This is partly done to conserve battery life.

The amplifier gain can be increased to improve the Sniffer's detection range. Feedback resistors R3 and R7 set the gain of amplifiers IC1.A and IC1.B. Try increasing both of these up to 1 MΩ or even 10 MΩ. It will not cause any damage and you will soon know when the amplification factor is too high because the headphone noise level increases and eventually the amplifier will start to whistle.

I'm sure I'm not the only one who rings my mobile number to find out where I have left the phone but what if the mobile's ringer is turned off? No problem for the Sniffer, it can simply pin-point the (silently) ringing mobile by homing in on its RF pulses! Sounds like a good idea for a party game…

(010128-1)

COMPONENTS LIST

Resistors:
R1,R5 = 100kΩ
R2,R6 = 1kΩ
R3,R7 = 820kΩ
R4,R8 = 220Ω
R9 = 2kΩ2

Capacitors:
C1,C2,C3 = 100nF
C4 = 100µF 16V radial

Inductors:
L1,L2 = 10 turns of 0.3-0.5mm diameter (30-24 SWG) enamelled copper wire, internal diameter 5mm (see text)

Semiconductors:
D1,D2 = BAT43, BAT45, AA112, AA116, AA119
D3 = LED, low-current
IC1 = LM358 P

Miscellaneous:
K1 = STEREO socket for 3.5mm jack plug, PCB mount (e.g. Conrad Electronics #732893)
S1 = on/off switch or pushbutton, 1 make contact
9-V battery (PP3) with clip-on leads
8-way IC socket
Case with battery compartment, e.g., Pactec type K-RC-24-9VB-BC (Conrad Electronics #522864)
Antennae: 2 wire pieces, length 3 to 7.5cm each (see text).

