THE ELECTRONICS & COMPUTER MAGAZINE
JANUARY 2000
£2.95

ELEKTOR ELECTRONICS

PC TOPICS:

- Bluetooth: wireless data communications
- LPT exerciser

Rhine Tower LED clock

active loop aerial

537 'Lite' computer

compiler for PIC16C84

9 770268 451081 01

This article describes how to write PIC 16C84 assembler code programs using a compiler written for the Windows 95/98 platform.
The PIC compiler allows programs to be written in a high-level language and it generates the necessary assembler code. The code can be modified, assembled or simulated using the (free) Microchip MPLAB software.

By Roger Thomas

compiler for PIC16C84

with code optimisation

Compiler main features for PIC16C84

- *Extensive manual (56 pages) on disk*
- *Three worked out examples on disk*
- *Syntax similar to Visual BASIC and Pascal*
- *Windows 95/98 compatible, no DLLs required*
- *Generates assembler code for Microchip MPLAB (freeware)*
- *Variables:* *Boolean, byte, word*
- *Arithmetic Operations:* *+, –, /, *, mod*
- *Numeric Formats:* *decimal, hexadecimal, binary, char*
- *Boolean Functions:* *=, >, <, >=, <=, <>*
- *Boolean Operators:* *AND, OR, XOR*
- *Compiler Commands:* *if...then…else, select/case, while…loop, table, read, write/read, EEPROM, procedure, directive, ASM directive, input, output, alias, pin-name, RTTC, prescaler, wait, picfuse,*
- *Equation Handler*
- *Code Optimiser*
- *Error messages*
- *Interrupt handling*

There are considerable advantages in using a compiler. The ability to write a PIC program using English like commands is easier than programming directly in assembler language. Time taken to write and test software is usually much less with a compiled language, and to prove the point the Windows PIC compiler itself was written using a compiler.
The PIC compiler is not based on any one high-level language but has elements of Pascal and Visual BASIC. Furthermore the compiler is flexible and allows for different program syntax. The compiler is written for the Windows environment, which should make the software easy to use. As all the necessary PIC codes are defined within the compiler program, no external setup or header files are needed.

The compiler produces assembler code directly from the high-level source program, so that the programmer need not worry about the intricacies of assembler code programming. It can be educational for those learning to program in assembler to see how easily understood high-level commands are translated into the equivalent PIC assembler code.

Assembler code output files produced by the compiler contain both the original high-level program (commented out) and the PIC assembler code ready to be assembled (or simulated) by the Microchip MPLAB software version 4.12 or later. Having both the source and assembler code helps in debugging the program. Writing in compiled language does not preclude modifying or adding assembler code to the program when using the MPLAB software. MPLAB software is freely available from Microchip's web site at *http://www.microchip.com*.

The cost (in programming terms) of using compiled code can be reduced speed of execution as the machine code program may not be as efficient as a program written directly in assembler.

With this PIC compiler this is not the case, in most circumstances the assembler code produced is the fastest code possible. There is very little compiler overhead on the assembler code in terms of needing extra variables or increased number of assembler instructions. The only additional program code required is support for the Boolean and arithmetic commands. Arithmetic is either 8 bit (unsigned) or 16 bit (unsigned). The compiler requires several bytes for storing arithmetic

results, these are labelled _STACKxx in the output assembler file.
The compiler makes two passes of the source program. The first pass creates a list of procedure declarations as the compiler may come across a call to a procedure before finding the procedure declaration. On the second pass the procedure calls are reconciled with the procedure declarations.
Compiler syntax is not case sensitive but the MPLAB software can be, this option is selected in the project hex file. For this reason all the procedure names and variables appear in uppercase in the assembler output file.

To allow the compiler to be used for any similar PIC microcontroller the compiler does not impose any code restrictions. This is left to the MPLAB assembler which will check program size and can more easily produce memory usage maps and cross-reference files.

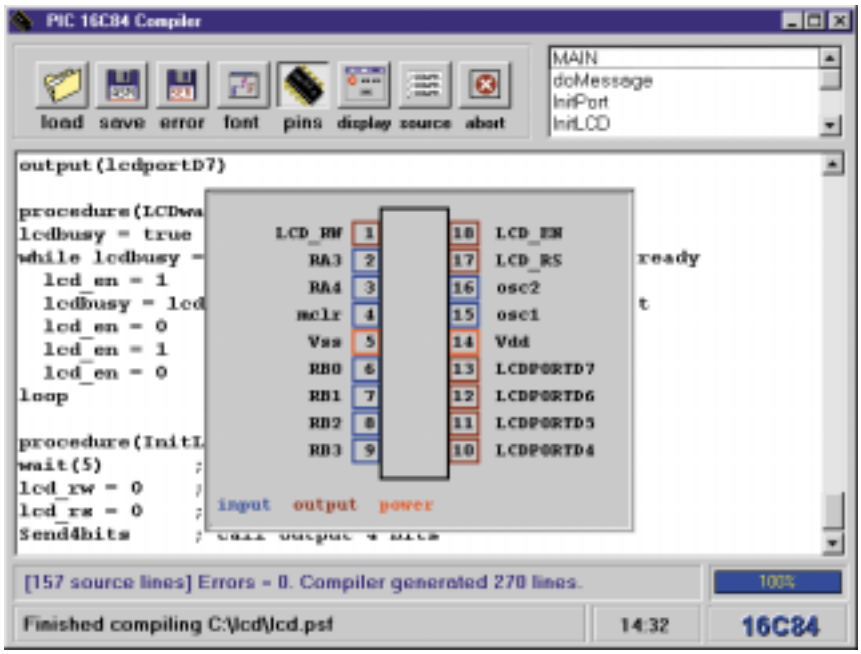


Figure 1. The Compiler window. Note that PIC pin functions can be seen at a glance.

Using the compiler

Use a text editor (such as Notepad, WordPad or the MPLAB editor) to create the high-level source program and save the text file with a .psf file extension (PIC Source File). Ensure that the saved file is text only and does not contain any embedded text formatting information.
Unlike an assembler program that requires a strict code column order (labels, mnemonics, operands, comments), a high-level program freely uses spaces to indent the program. These spaces have no relevance to the program execution and are ignored by the compiler. Using spaces should make the reading and de-bugging the program easier.

User interface

The Windows PIC compiler is very easy to use, apart from using load and save file the compiler does everything else!

As shown in **Figure 1**, the taskbar has a number of icons.

load - press the load button and a directory dialog box will appear listing all the source (filenames.psf) files in the directory, select and load the relevant source file. The compiler will default to the directory that was last used. When the compiler is run for the first time the directory will be where the compiler program is located.

save - after a successful compilation save the assembler source file (same file name but with filename.asm file extension) by pressing this button. This file will be saved in the same directory as the source file. This assembler output file will contain all the additional PIC code required as the compiler will automatically add any support routines.

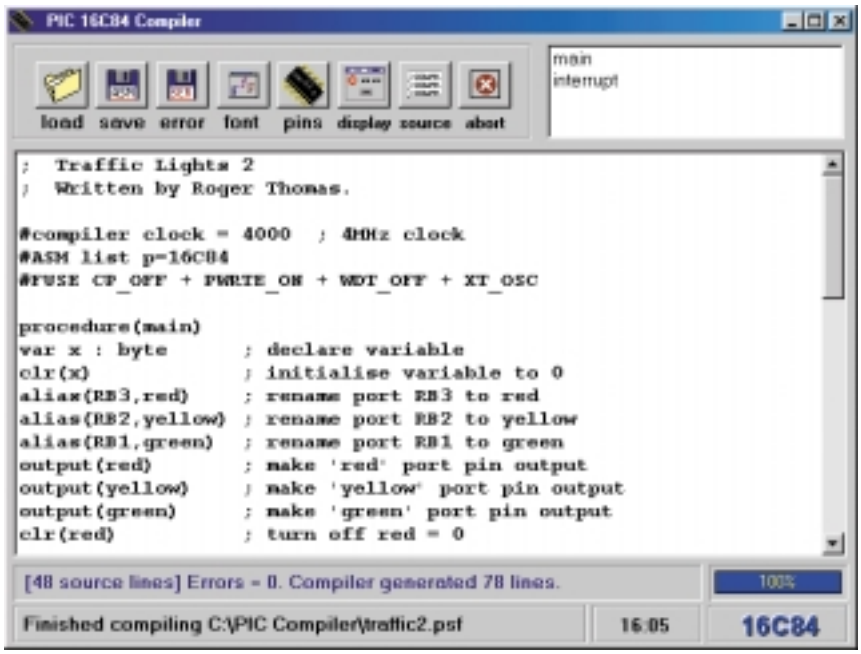
error - saves the error file as a text file to the same directory as the source file. This file has the same file name but with an '.ser' (source error) file extension. It contains all the error messages (which includes the source line number) but not any of the source or assembler code.

font - to change the font or font size of the text displayed on the screen press the font button. A font dialogue box will appear, from which you choose the required font and font size.

pins - will display the PIC pin names and colour coded input or output port pins.

display - if there are any compiler error messages these are inserted into the assembler output file, optionally the compiler will stop and display a mes-

Figure 2. Another example of the PIC Compiler in action. Here, a traffic lights program is being written. Note the procedure names in the top right-hand window.



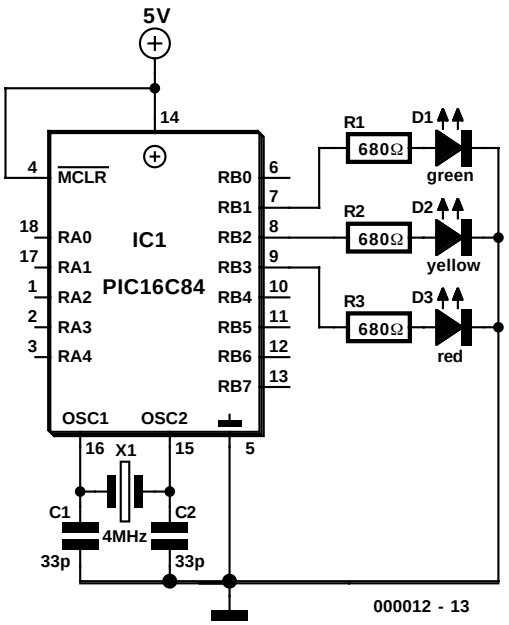


Figure 3. Traffic Lights demo hardware.

sage dialog box (default is to display error message). On the first pass any error messages will be displayed irrespective of this setting.

source - selects whether the source lines should be included in the output assembler file as comments (default is to include source code). Source lines that only contain a comment will always be included.

abort - stops the compilation process.

Next to the eight buttons is a list of all the procedure names used in the program. To find and display a particular procedure, select it from the list (the name will be briefly highlighted) by clicking on the left mouse button. The screen text should scroll and display the relevant procedure (**Figure 2**).

The main procedure of the PIC program is called 'main'. Program execution will start at this procedure, it is called whenever the PIC is reset. The PIC application program is generally held in a continuous loop after any initialisation is done, waiting for events to happen. It is extremely rare for a PIC program to be required to run only once.

Compiler code optimisation

After the compiler has successfully compiled a segment of source program, the code optimiser checks the assembler program for redundant code. Redundant code usually takes the form of unnecessary setting or reading the various PIC status flags.

For example, part of a traffic lights output procedure is shown, the compiler will generate the following code:

(Original source code fragment)

```
green = 0  ; turn green off
yellow = 0 ; turn yellow off
red = 1    ; turn on red

; code not optimised
; green = 0   ; turn green off
   MOVLWH'00'
   BTFSS_STATUS,_Z
   BSF   PORTB,GREEN
   BTFSC_STATUS,_Z
   BCF   PORTB,GREEN
; yellow = 0 ; turn yellow off
   MOVLWH'00'
   BTFSS_STATUS,_Z
   BSF   PORTB,YELLOW
   BTFSC         _STATUS,_Z
   BCF   PORTB,YELLOW
; red = 1       ; turn on red
   MOVLWH'01'
   BTFSS_STATUS,_Z
   BSF           PORTB,RED
   BTFSC_STATUS,_Z
   BCF           PORTB,RED
```

With the code not optimised the compiler has calculated the equation after the equals sign and sets the bit according to the equation result: zero or one. The optimiser looks at the code and finds it has a constant value as it always has the same result and deletes the intermediate calculation.

```
; code optimised
; green = 0   ; turn green off
   BCF   PORTB,GREEN
; yellow = 0 ; turn yellow off
   BCF   PORTB,YELLOW
; red = 1    ; turn on red
   BSF   PORTB,RED
```

PIC interrupts

When an interrupt occurs the program counter is loaded with address location 4, this contains code to save the program status and calls the interrupt handler procedure. After completion of the interrupt handler code within procedure(interrupt), the PIC executes a retfie instruction (return from interrupt). It is not necessary for the source program to re-enable global interrupts as the retfie instruction will do it automatically. The routine to handle interrupts must be called interrupt.
The use of interrupts makes a PIC program more efficient as the alternative is having to continually poll flags to see if a particular event has taken place.
There are four sources of interrupts that the PIC 16C84 interrupt handler software has to deal with:

1. external interrupt on pin RB0;
2. interrupt on change to pins RB4-RB7;
3. RTCC timer overflow;
4. assigned to EEPROM write complete.

The EEPROM write interrupt is taken care of by the writeEEPROM function.

To enable the interrupts use the *irq_enable = true* command. The individual interrupt source must be selected before this command is invoked.
To disable all interrupt use the *irq_enable = false* command.
This command acts globally using the Global Interrupt Enable (GIE) flag irrespective of the individual interrupt being used.

The interrupt handler procedure needs a few bytes for intermediate storage, the interrupt routine has to have its own variable storage as it cannot share storage with the rest of the program.
As an interrupt can occur at any time it is possible that with 16-bit arithmetic this could happen half way through an arithmetic procedure. Assigning a 16-bit variable to another 16-bit variable requires multiple instructions to move the value of the lower and then the higher byte. If an interrupt occurs half way through the process then the variable may end up with the value of the old value (lower byte) and the new value of the higher byte. For this reason the compiler will not allow 16-bit arithmetic in the interrupt handler procedure.

If only one source of interrupts has been enabled then it is not necessary to look at the individual interrupt enable flags. In general it is best to make the interrupt handler procedure as small and execute as fast as possible using simple equations. Note that other procedures cannot be called from within the interrupt handler procedure.

It is better to make a copy of any variable that the interrupt handler may use and use the copy. Referencing a variable that the interrupt handler directly uses can have unforeseen results. For example, if x is changed by the interrupt handler then the following program might not function as intended. The value of x may have been altered after the first but before the second comparison command, so no statements are executed.

```
var x : byte
procedure(main)
if x >=6  and x <= 10 then
      ;  x = 4
begin
    statement
end   ;  interrupt occurs here
if x >= 0 and x <=5 then
      ;  x changes to 6
begin
    statement
end
```

If a byte variable needs to be incremented or decremented or set to zero within the interrupt handler then use inc(x) or dec(x) or clr(x) statements as these compile to a single assembler instruction.

Program examples

To help demonstrate the advantages of using the compiler and clarify the language syntax, the documentation file contains some example programs to help illustrate the various compiler commands. As these programs are for didactic purposes, they do not necessarily represent the best software solution. Note that some of the comment lines have been deleted and the assembler file tidied up for publication.

All variable labels that the compiler generates are preceded by an underscore to differentiate them from variables used in the source program.

The easiest method of implementing a **traffic lights sequence** would be to use the wait command after setting the appropriate LED on or off.

```
green = 0  ; turn green off
yellow = 0 ; turn yellow off
red = 1    ; turn on red
wait(3000) ; wait for 3 seconds
```

If you intend to build the circuit shown in **Figure 3** please observe the PIC current limits. The maximum total current output on Port B is 100 mA, any pin has

Listing 1. Traffic Lights (1) Source program

```
;  Traffic Lights 1
;  Written by Roger Thomas.

#compiler clock = 4000   ; 4MHz clock
#ASM list p=16C84
#FUSE CP_OFF + PWRTE_ON + WDT_OFF + XT_OSC

var x : word   ; create 16 bit variable
var y : byte   ; create 8 bit variable

procedure(main)
alias(RB3,red)     ; rename port RB3 to red
alias(RB2,yellow)  ; rename port RB2 to yellow
alias(RB1,green)   ; rename port RB1 to green
output(red)        ; make 'red' port pin output
output(yellow)     ; make 'yellow' port pin output
output(green)      ; make 'green' port pin output
clr(red)           ; turn off red = 0
clr(yellow)        ; turn off yellow = 0
clr(green)         ; turn off green = 0

clr(x)             ; initialise = 0
clr(y)             ; initialise = 0

while true
  inc(x)           ; x = x + 1
  if x = 1500 then
  begin
    inc(y)         ; y = y + 1
    clr(x)         ; x = 0
  end
  else
  begin
    if (y >= 0) AND (y<= 49) then
    begin
      red = 1      ; turn on red
      yellow = 0   ; turn off yellow
      green = 0    ; turn off green
    end

    if (y >= 50) AND (y<= 75) then
    begin
      red = 1      ; turn on red
      yellow = 1   ; turn on yellow
      green = 0    ; turn off green
    end

    if (y >= 76) AND (y<= 110) then
    begin
      red = 0      ; turn off red
      yellow = 0   ; turn off yellow
      green = 1    ; turn on green
    end

    if (y >= 111) AND (y<= 130) then
    begin
      red = 0      ; turn off red
      yellow = 1   ; turn on yellow
      green = 0    ; turn off green
    end

    if y = 131 then
    begin
      clr(x)
      clr(y)
    end
  end
loop
```

```
;                       16C84
;            RA2   1  | i    i | 18   RA1
;            RA3   2  | i    i | 17   RA0
;            RA4   3  | i    i | 16   osc2
;           mclr   4  | i    i | 15   osc1
;            Vss   5  | p    p | 14   Vdd
;            RB0   6  | i    i | 13   RB7
;          GREEN   7  | o    i | 12   RB6
;         YELLOW   8  | o    i | 11   RB5
;            RED   9  | o    i | 10   RB4


_PCL    EQU H'02'
_STATUS EQU H'03'
_C EQU H'00'
_Z EQU H'02'
_RP0    EQU H'05'
PORTB   EQU H'06'
_PCLATH EQU H'0A'
_INTCON EQU H'0B'
IRQ_ENABLE     EQU H'07'
_STACK0 EQU H'0C'
_STACK1 EQU H'0D'
_STACK2 EQU H'0E'
_STACK3 EQU H'0F'
_STACK4 EQU H'10'
_STACK5 EQU H'11'
_STACK6 EQU H'12'
_STACK7 EQU H'13'
_STACK8 EQU H'14'
_STACK9 EQU H'15'
X  EQU H'16'
XH EQU H'17'
Y  EQU H'18'
RED     EQU H'03'
YELLOW  EQU H'02'
GREEN   EQU H'01'

   ORG 0

   goto MAIN

;  Traffic Lights 1
;  Written by Roger Thomas.

   list p=16C84
   __config H'3FF9'

; var x : word    ; create 16 bit variable
; var y : byte    ; create 8 bit variable

MAIN
; alias(RB3,red)     ; rename port RB3 to red
; alias(RB2,yellow) ; rename port RB2 to yellow
; alias(RB1,green)  ; rename port RB1 to green
; output(red)        ; make 'red' port pin output
   BSF  _STATUS,_RP0
   BCF  PORTB,RED
; output(yellow)     ; make 'yellow' port pin out-
put
   BCF  PORTB,YELLOW
; output(green)      ; make 'green' port pin output
   BCF  PORTB,GREEN
; clr(red)           ; turn off red = 0
   BCF  _STATUS,_RP0
   BCF  PORTB,RED
; clr(yellow)        ; turn off yellow = 0
   BCF  PORTB,YELLOW
; clr(green)         ; turn off green = 0
   BCF  PORTB,GREEN

; clr(x)             ; initialise = 0
   CLRF X
   CLRF XH
; clr(y)             ; initialise = 0
   CLRF Y

; while true
_WHILE0
;    inc(x)           ; x = x + 1
   INCF X,F
   BTFSC        _STATUS,_Z
   INCF XH,F
;    if x = 1500 then
_IF1
   MOVF X,W
   MOVWF        _STACK0
   MOVF XH,W
   MOVWF        _STACK1
   MOVLW        H'FF'
   MOVWF        _STACK2
   MOVLW        H'DC'
   SUBWF        _STACK0,F
   BTFSS        _STATUS,_Z
   CLRF _STACK2
   MOVLW        H'05'
   SUBWF        _STACK1,F
   BTFSS        _STATUS,_Z
   CLRF _STACK2
   MOVF _STACK2,W
   MOVWF        _STACK0
   MOVWF        _STACK1
   BTFSC        _STATUS,_Z
   GOTO _ELSE1
;    begin
;      inc(y)         ; y = y + 1
   INCF Y,F
;      clr(x)         ; x = 0
   CLRF X
   CLRF XH
;    end
;    else
   GOTO _END1
_ELSE1
;    begin
;      if (y >= 0) AND (y<= 49) then
_IF2
   MOVF Y,W
   MOVWF        _STACK0
   MOVLW        H'00'
   SUBWF        _STACK0,W
   CLRW
   BTFSC        _STATUS,_C
   ADDLW        H'FF'
   MOVWF        _STACK4
   MOVF Y,W
   SUBLW        H'31'
   CLRW
   BTFSC        _STATUS,_C
   ADDLW        H'FF'
   ANDWF        _STACK4,W
   BTFSC        _STATUS,_Z
   GOTO _ELSE2
;      begin
;        red = 1      ; turn on red
   BSF  PORTB,RED
;        yellow = 0   ; turn off yellow
   BCF  PORTB,YELLOW
;        green = 0    ; turn off green
   BCF  PORTB,GREEN
;      end

;      if (y >= 50) AND (y<= 75) then
_ELSE2
_IF3
   MOVF Y,W
   MOVWF        _STACK0
   MOVLW        H'32'
   SUBWF        _STACK0,W
   CLRW
   BTFSC        _STATUS,_C
   ADDLW        H'FF'
```

```
   MOVWF       _STACK4
   MOVF Y,W
   SUBLW       H'4B'
   CLRW
   BTFSC       _STATUS,_C
   ADDLW       H'FF'
   ANDWF       _STACK4,W
   BTFSC       _STATUS,_Z
   GOTO _ELSE3
;      begin
;        red = 1      ; turn on red
   BSF  PORTB,RED
;        yellow = 1   ; turn on yellow
   BSF  PORTB,YELLOW
;        green = 0    ; turn off green
   BCF  PORTB,GREEN
;      end

;      if (y >= 76) AND (y<= 110) then
_ELSE3
_IF4
   MOVF Y,W
   MOVWF       _STACK0
   MOVLW       H'4C'
   SUBWF       _STACK0,W
   CLRW
   BTFSC       _STATUS,_C
   ADDLW       H'FF'
   MOVWF       _STACK4
   MOVF Y,W
   SUBLW       H'6E'
   CLRW
   BTFSC       _STATUS,_C
   ADDLW       H'FF'
   ANDWF       _STACK4,W
   BTFSC       _STATUS,_Z
   GOTO _ELSE4
;      begin
;        red = 0      ; turn off red
   BCF  PORTB,RED
;        yellow = 0   ; turn off yellow
   BCF  PORTB,YELLOW
;        green = 1    ; turn on green
   BSF  PORTB,GREEN
;      end

;      if (y >= 111) AND (y<= 130) then
_ELSE4
_IF5
   MOVF Y,W
   MOVWF       _STACK0
   MOVLW       H'6F'
   SUBWF       _STACK0,W
   CLRW
   BTFSC       _STATUS,_C
   ADDLW       H'FF'
   MOVWF       _STACK4
   MOVF Y,W
   SUBLW       H'82'
   CLRW
   BTFSC       _STATUS,_C
   ADDLW       H'FF'
   ANDWF       _STACK4,W
   BTFSC       _STATUS,_Z
   GOTO _ELSE5
;      begin
;        red = 0      ; turn off red
   BCF  PORTB,RED
;        yellow = 1   ; turn on yellow
   BSF  PORTB,YELLOW
;        green = 0    ; turn off green
   BCF  PORTB,GREEN
;      end

;      if y = 131 then
_ELSE5
_IF6
   MOVF Y,W
   SUBLW       H'83'
   MOVLW       H'00'
   BTFSC       _STATUS,_Z
   ADDLW       H'FF'
   ANDLW       H'FF'
   BTFSC       _STATUS,_Z
   GOTO _ELSE6
;      begin
;        clr(x)
   CLRF X
   CLRF XH
;        clr(y)
   CLRF Y
;      end
;   end
_ELSE6
_END1
; loop
   GOTO _WHILE0

   END
```

Listing 2. Traffic Lights (1) Assembler program

a absolute maximum current output of 20 mA. Incorporate an appropriate current limiting resistor (R) in series with the LED (in the range 470 Ω to 1 kΩ depending on the LED). Here, 680 Ω is suggested.

The example program will continue to execute until the supply voltage is removed from the PIC.

The source code of the program is shown in **Listing 1**. The 'x' variable is incremented on each loop of the program. After reaching a certain number it then increments the 'y' variable. This is needed to slow the program down — if the 'x' variable was used directly the lights would switch too fast. The brackets separating the 'y' conditions are not required by the compiler but help document the program. The resulting assembly-code file is shown in **Listing 2**. Other programming examples found in the documentation file are *Traffic Lights (2)* and *LCD Display Driver*. The source code and assembly-code listings of these programs may be found in the documentation file.

Syntax and command descriptions

A full description of all available commands and the syntax the Compiler wants to see may be found in the 56-page project documentation file. This file, an MS Word document, may be found on diskette no. **996033-1** which may be ordered through our Readers Services. The disk also contains the example source code files (.psf) and, of course, the Compiler itself (Compiler84.EXE). The readme.txt file explains the extremely simple installation.

(000012-1)

Article editing: Jan Buiting

This experimental circuit board was designed and built as a development aid for those constructors who wish to design their own parallel printer add-on boards. The author used the prototype board whilst experimenting with digital to analogue converters (DACs) and liquid crystal displays (LCDs) amongst other things.

Hardware design & software: Adrian Grace

LPT exerciser

gateway to the PC parallel port

The board of which the circuit diagram is shown in **Figure 1**, connects to the standard 25 way 'D' connector of the computer's parallel port via a 25 way ribbon cable, and is terminated on the circuit board with a transition connector (see the relevant construction notes). The design itself comprises of three circuits — one for each of the registers (DATA register, STATUS register and CONTROL register) associated with the parallel printer port.

Registers

First, a quick word about the registers. The DATA register is an 8 bit output latch which is normally used to transfer data to the printer. The STATUS register has 5 readable inputs and is used by the computer to 'monitor' the status of the printer for error messages etc. The CONTROL register has 4 outputs, and is normally used to control the functionality of the printer.

Data register

The 8 outputs of the data register are connected to 8 LED's (D5-D12) via current limiting resistors (R5-R12). The LEDs themselves are 5mm wide by 2mm thick and fit comfortably into the traditional 2.54mm grid pattern.

Status register

The status register has 5 inputs which are simulated via a DIL switch. The inputs are as follows:

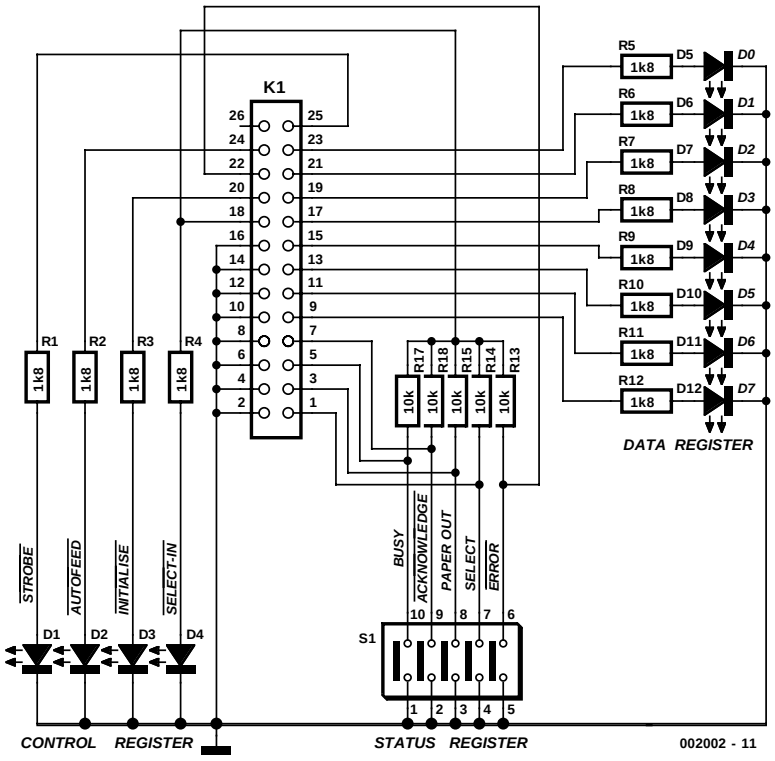


Figure 1. Circuit diagram of the LPT board.

The switch contacts are pulled up via resistors (R13-R17) and their status can be read providing the $\overline{\text{SELECT IN}}$, bit D3 of the control register is at a logic high. This allows the board to become self powered — i.e. batteries not required.

Control register

The control register is 4 bits wide — lower nibble. Each output is connected to an LED via a current limiting resistor in a similar manner to the data register outputs. Indeed, the LEDs should have the same packaging but preferably a different colour. The function of each bit is shown below:

D7	D6	D5	D4	D3	D2	D1	D0
$\overline{\text{Busy}}$	Acknowledge	Paper out	Select	$\overline{\text{Error}}$	Unused	Unused	Unused

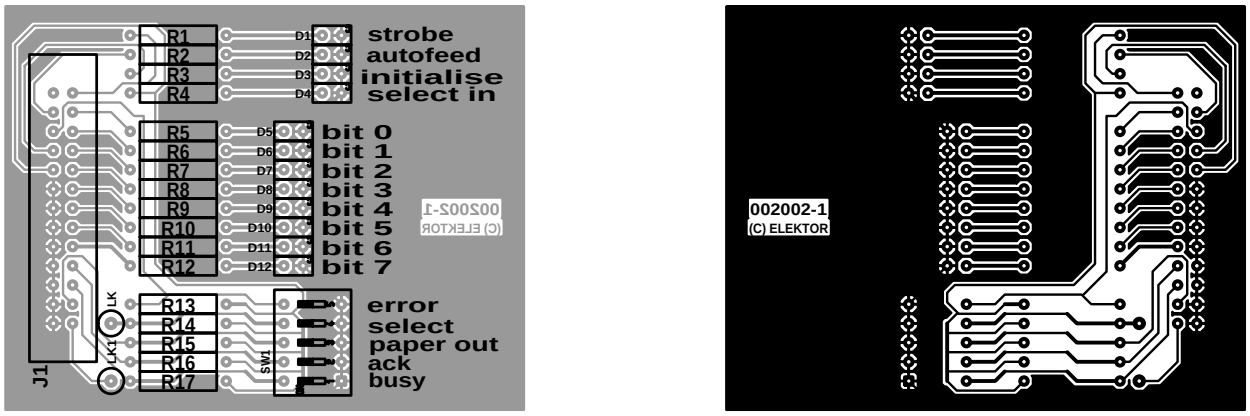


Figure 2. PCB artwork designed by the author.

D7	D6	D5	D4	D3	D2	D1	D0
Unused	Unused	Direction	IRQ	Select In	Initialise	Autofeed	Strobe

LED D4 has a dual purpose, when it is energised it provides power to allow the switch positions connected to the status register to be read.
The upper nibble of the control register contains some control bits which are not addressed in this design.

Construction

The Ultiboard design files of the PCB designed by the author for this project may be found on diskette **006002-1** which is available through the *Elektor Electronics* Readers Services. Population of the board is straightforward, see **Figure 2**. Fit all components on the circuit board except for J1. Don't forget the link. J2 can be fitted to the ribbon cable at this point — after the extra 'way' has been removed, of course, as a 26-way ribbon cable won't go into a 25-way connector! The coloured leader stripe on the ribbon cable (which identifies pin 1) should be aligned with pin 1 on J2.

Now comes the fun bit. Lay the circuit board in front of you so that the position of J1 is furthest away from you. Lay the ribbon cable over the top of the circuit board so that the free end of the ribbon cable is at the J1 position. The ribbon cable should be orientated so that the leader stripe is on the right hand side. OK so far? Slide the ribbon cable into the transition connector J1, so that there is an empty space on the right hand side, and the left-hand side fits snugly against the edge of the connector. J1 can now be squeezed onto the ribbon cable – make sure that it is fitted squarely. This is easy to do if you use a couple of pieces of scrap 'veroboard' on top of each other to protect the pins of J1. Finally flip the ribbon cable over and fit the strain relief clip. J1 can now be soldered onto the circuit board.

Software

The software associated with this design is available on the above mentioned diskette with order code **006002-1**. It has been written in 'C' and is suitable for running under DOS. As you can see in **Figure 3**, the program itself (executable file) is menu-driven. The program will first of all interrogate the computer's BIOS to establish the printer port address and display it on screen. At present, the program will only look for the first port LPT1.
The main menu displays three options, one for each register and an exit path.

The first option is for the STATUS register. With this option, Select In is automatically actuated, allowing the switch positions to be read. The hex value of the switches is displayed on screen until a key is pressed on the keyboard.

The second option is for the DATA register. When selected, a second submenu is displayed along with the present value of the data register. The submenu allows the user to shift bits left or right and increment or decrement the register values. The register value is written out to the parallel port so the bit patterns can be viewed on the LEDs.

The third option is for the CONTROL register. This option just cycles a moving LED up and down the four LEDs associated with the control register — until a key is pressed as before.

Diskette **006002-1** contains the original source code and the intermediate object file as well as the executable file. The source code is highly commented and you are at liberty to experiment with it — that's how I learnt how to write it.

If you intend to go on and experiment further with your own designs connected to the parallel port, **it is strongly suggested that you obtain a 'cheap' secondary LPT port board**. This will prevent you from damaging your main parallel port, which is usually integrated into the motherboard. Damaging your motherboard is not a good idea!

COMPONENT LIST

D1-D4 =5mm x 2mm LED, high efficiency, green
D5-D12 =5mm x 2mm LED, high efficiency, red
R1-R12 = 1kΩ8, 0.25W
R13-R17 = 10kΩ, 0.25W
SW1 = 5-way DIL switch
J1 = 26-way TRANSITION connector, plus strain relief clip.
J2 = 25 way MALE IDC ribbon connector 26 way ribbon cable, cut to length with one 'way' removed.

(002002-1)

Technical editing: Karel Walraven
Article editing: Jan Buiting

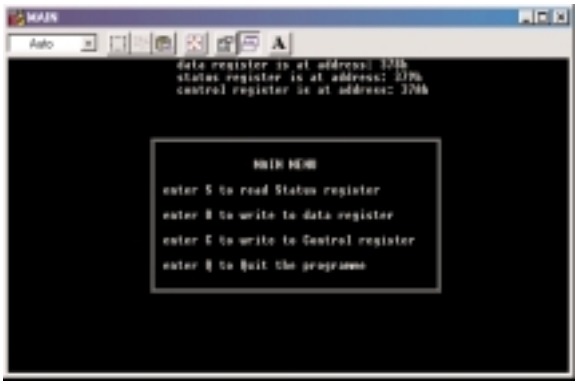


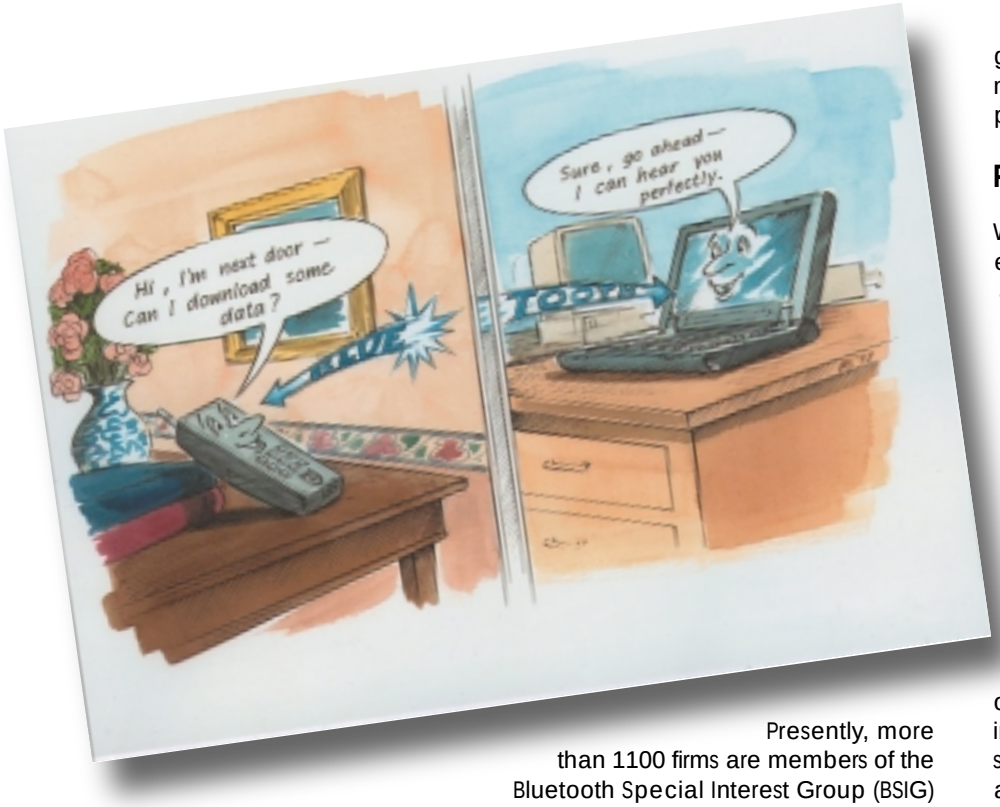
Figure 3. Screen dump showing some of the options offered by the program.

Presently, data transfers between the PC and printers, scanners, fax machines and modems, as well as communication between a laptop and a cellular phone, still require inconvenient cable interconnections and the time-consuming installation of drivers. Wouldn't it be nice if all your data-processing devices could independently converse with each other, with no need for cables or driver installation? This is now possible, thanks to Bluetooth, a short-range radio system that renders cables redundant and which should allow new applications to be developed.

By G. Kleine

Bluetooth

2.4-GHz data radio instead of computer cables



Nokia Mobile Phones, Ericsson Mobile Communications, IBM, Intel and Toshiba recognized the potential of this technology and founded the Bluetooth Special Interest Group (BSIG) [1,2] in May of 1998. Its objective is to define a universal standard for radio data communication between data-processing equipment. In the beginning, the main area of interest was data transfers between cellular phones and peripheral equipment, in order to eliminate annoying and inconvenient cables. Later, the area of interest was extended to include other applications, ranging from wireless connections between PCs and laptops and their peripheral equipment to the ad-hoc configuration of wireless networks.

Presently, more than 1100 firms are members of the Bluetooth Special Interest Group (BSIG) (see the box 'King Bluetooth and his friends'). They include not only all leading computer and communications technology firms, but also automobile and aeroplane manufacturers and representatives of the entertainment industry. The BSIG presented the provisional version of the Bluetooth 1.0 specification in the autumn of 1999. It has been made publicly available on the Internet [2], in order to promote the establishment of the Bluetooth system. This extensive specification fills more than 1500 pages. The primary objective of the Bluetooth project is the development of extremely inexpensive, compact modules that can be built into a variety of equipment. The targeted unit price, given large-scale mass production (more than one million pieces), is less than five dollars.

Frequencies

When the BSIG was looking for licence-exempt frequency bands that are available worldwide, it came across the Industrial, Scientific and Medical (ISM) bands. These frequency bands may be used without any licence or fee for industrial, scientific and medical equipment with a limited radiated power. In addition to the available ISM bands listed in **Table 1**, there are other ISM bands at 5.8 GHz, 24.25 GHz and 122.5 GHz, but these cannot be utilized at a reasonable cost at the present time.

Since the possibility of interference from other unlicensed users of the ISM bands must always be taken into account, and given that Bluetooth should have a data transfer rate of around 1 Mb/s, the only suitable choice is the 2.4-GHz ISM band (2400 to 2483 MHz). This band is 83 MHz wide and thus provides enough room to avoid interfering signals coming from equipment that uses a particular frequency, such as a microwave oven. This frequency band is available worldwide, although its extent is somewhat reduced in Japan (2471 to 2497 MHz), France and Spain.

In order to avoid problems with anticipated interference signals, most of which have fixed frequencies, Bluetooth makes use of spread-spectrum techniques. This means that the frequency is changed rapidly (up to 1600 times per second) in a pseudo-random manner, with the result that enough interfer-

ence-free frequencies are available to allow the desired data transfer rate to be achieved, even with error protection (see **Figure 1**). This process is called Frequency-Hopping Spread Spectrum (FHSS). Bluetooth uses 79 frequencies in the range between 2.402 and 2.480 GHz, with a separation of 1 MHz. Due to the previously-mentioned reduction of the available frequency band, only 23 hop frequencies are used in France, Spain and Japan.

Piconets and scatter nets

A pair of devices equipped with Bluetooth modules can autonomously establish contact with each other, using the frequency-hopping technique. In order to prevent mutual interference with other Bluetooth equipment in the same vicinity that might use the same frequencies, the frequency sequence is determined by the address and clock rate of the device that initiates the connection. This device thus becomes the master device in this radio group, while all other participants are designated as slaves and must synchronize themselves to the master device.
Up to eight Bluetooth radios can use a single channel. Such a group forms a network that is called a *piconet* (see **Figure 2**). In this context, the term channel means that all members of the piconet employ the same hopping sequence, which means that they use the same series of frequencies. Since each Bluetooth device is assigned a unique 48-bit address by a central registration agency, it is not possible for two separate channels to have the same hopping sequence.
By independently setting up additional piconets in the same vicinity, it is possible to use more than eight Bluetooth devices in one location, such as an office, with high data transfer rates and without interference.
A slave transmit/receive device operating in a particular piconet is addressed by the master device in one time slot of a time-division multiplexed (TDM) protocol and may respond in the subsequent time slot. The slave is free to participate in another piconet in the remaining time slots. In order to do so, it sets its receiver to the frequency to which the other piconet has just hopped and synchronizes itself to the master device of that piconet. Multiple radio networks that are interconnected in this manner are referred to in Bluetooth terminology as *scatter nets* (see Figure 2). TDM collisions are avoided by the fact that the slave devices synchronize themselves to the clock rate of

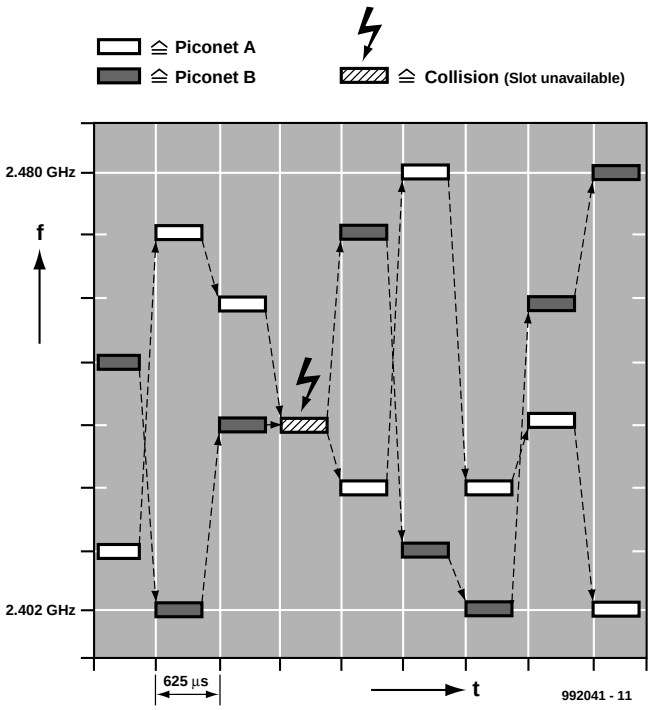


Figure 1. Frequency hopping (simplified representation)

Table 1. Some ISM frequency bands (not available in all countries)		
Frequency range [1]	**Bandwidth**	**Shared with / Notes**
26.957 MHz - 27.283 MHz	0.326 MHz	CB, cordless phones, ...
40.660 MHz - 40.700 MHz	0.040 MHz	small bandwidth
433.050 MHz - 434.790 MHz	1.74 MHz	Amateur radio
868 MHz - 870 MHz	2.00 MHz	little used, small bandwidth
2.400 GHz - 2.483 GHz	83.00 MHz	Microwave ovens, door openers
[1] *Note: additional ISM Bands may be allocated around 5.8 GHz, 24.250 GHz and 122.5 GHz*		

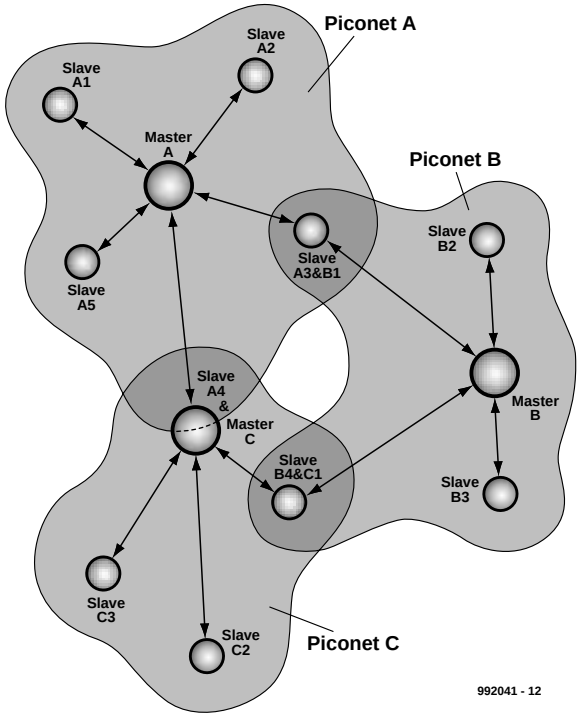


Figure 2. Piconets and scatter nets

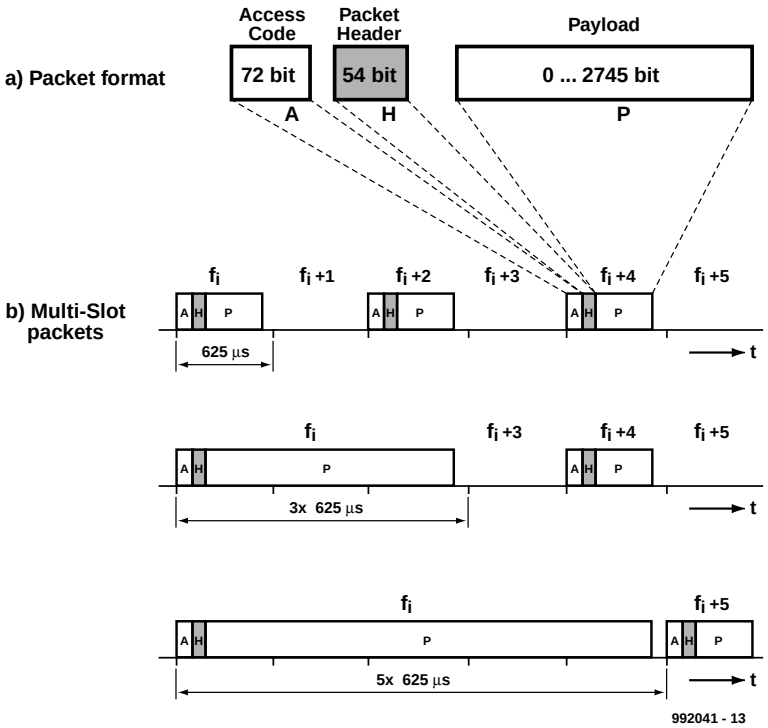


Figure 3. Packet format and multi-slot packets

the master.
As we have seen, up to eight devices are allowed in a single Bluetooth piconet. The higher-level scatter net can contain up to ten piconets, so that it is possible to configure systems containing up to 80 Bluetooth devices within a 10-metre radius. Even when a single device is a member of all ten piconets, the data transfer rate in any piconet is reduced by no more than around ten percent.

Modulation and error protection

Gaussian Frequency-Shift Keying (GFSK) modulation is employed for the frequency-hopping narrow-band carrier of each channel. With a frequency deviation of around 150 kHz, the 3-dB carrier bandwidth is 220 kHz. This fairly simple modulation scheme was chosen in order to keep the cost of the transceiver chips as low as possible. The transmitted power, at 1 mW (0 dBm) nominal, allows for an effective range of at least ten metres under normal conditions. In certain special cases, the generation of higher field strengths (up to 100 mW transmitted power, or +20 dBm) is permitted with spread-spectrum transmissions, which enables effective ranges of up to 100 metres to be attained. The Bluetooth radio module can adapt its transmitted power to the transmission environment, within certain limits.
Error protection in the Bluetooth system is also adaptive, which means that error protection is dispensed with in favour of a higher data transfer rate if there is a very high-quality radio link. Two-stage Forward Error Correction (FEC) is only activated if interference occurs in the radio link. This naturally reduces the data transfer rate. In addition, the Automatic Retransmission Query (ARQ) technique is used, except for voice channels. ARQ allows re-transmission of a packet to be requested. The header of every packet is always protected using FEC, so that even if FEC is switched off, re-transmission can always be instigated via ARQ if interference occurs.

Table 2. ACL link data rates

FEC	Slots	Data rate, symmetrical	Data rate, asymmetrical
none	1	2 x 172.8 kBit/s	172.8 kBit/s + 172.8 kBit/s
none	3	2 x 384.0 kBit/s	576.0 kBit/s + 86.4 kBit/s
none	5	2 x 432.6 kBit/s	721.0 kBit/s + 57.6 kBit/s
yes	1	2 x 108.8 kBit/s	108.8 kBit/s + 108.8 kBit/s
yes	3	2 x 256.0 kBit/s	384.0 kBit/s + 54.4 kBit/s
yes	5	2 x 286.7 kBit/s	477.8 kBit/s + 36.3 kBit/s

SCO and ACL

The Bluetooth Baseband Protocol defines two types of data transfers: point-to-point and single-point to multiple-point.
Point-to-point transfers are referred to as *Synchronous Connection Oriented* (SCO). They are primarily intended to be used for voice data transmission. The link is thus symmetrical, which means that the data rate is the same in each direction. In practice, a full-duplex link is implemented by utilizing alternating time slots, with the data for one direction contained in one slot and the data for the other direction contained in the following slot. In contrast to data transfers, which are not time-critical, it is naturally not possible to use the ARQ technique for speech transmissions if interference occurs on certain frequencies. Instead, the *Continuous Variable-Slope Delta* (CVSD) method is used for voice encoding, since it exhibits good bit-error behaviour and produces only a slight increase in the background noise level in case of errors. The sampling rate for voice data transfers is 64 kb/s, the same as for ISDN.
Single-point to multiple-point transfers are asynchronous and connection-independent; they are based on data packets. Such links are referred to as *Asynchronous Connectionless* (ACL) in Bluetooth terminology. This type of transfer is used by a master device to communicate with several slave devices at the same time. In addition to being used for sending messages to all slave devices, this type of transfer is also used for sending data packets to a particular slave device. In order to increase the data transfer rate, a packet may use not only one 625-µs time slot, but also three or even five slots, as shown in **Figure 3**. To make this possible, frequency-hopping is suspended and the frequency at the start of the packet transfer (f_i) is maintained for the duration of the three or five slot intervals. This allows the 'holes' between successive slots, which are otherwise reserved for frequency hopping, to be used for the data transfer. In order to maintain overall synchronization, frequency hopping resumes on completion of the transfer with the frequency that would normally be used for the fourth slot (f_{i+4}) or the fifth slot (f_{i+5}), as appropriate.

Data transfers from a slave device to a master, or from one slave to another slave via the master, require the permission of the master. For ACL links, the symmetry of the data rates in the two directions is controlled by the master. With asymmetric links, the data rate in one direction can be as high as 721 kb/s, in which case the rate in the other direction is only 57.6 kb/s. Both of these values are based on five-slot transfers without forward error correction (FEC). If FEC and the previously-mentioned ARQ procedure are employed, the data rate naturally drops. A 2/3-rate FEC is used. With a symmetrical link and no error protection, the maximum data rate is 432.6 kb/s in each direction. **Table 2** lists additional data rates for ACL links.
With both types of links, ACL and SCO, there are 16 different types of packets that can be used for data transfers. Some of these are reserved for control functions. Every packet has a 72-bit identification field (Access Code), which is derived from the 48-bit master address and which is protected by FEC. Following this comes a 54-bit header field, which is also protected by 1/3-rate FEC. After this, in a normal packet, come up to 2745 bits of payload data (see **Figure 3**). Three-slot and five-slot packets can transfer correspondingly more payload data.
Bluetooth transfers can also be encrypted, using a 128-bit key for authentification. The user can determine whether he or she wants to use encryption in one direction or in both directions. This setting is saved. This allows the user to exactly specify the equipment with which a cellular phone (for example) is allowed to communicate. For example, you could allow your phone to access your own notebook, while denying any access to your colleagues.
With Bluetooth, it is possible to have either one asynchronous ACL channel or up to three simultaneous, synchronous SCO channels plus one asynchronous data channel with a parallel 64-kb/s speech channel.

Piconet states

Bluetooth modules that are not members of any piconet operate in the Standby mode, in which they search for possible transfers in their immediate environment every 1.28 seconds. They do this by testing 32 of the possible 79 frequencies, which are designated as wake-up carriers. In France, Spain and Japan there are 16 wake-up carriers among the total of 23 possible frequencies.

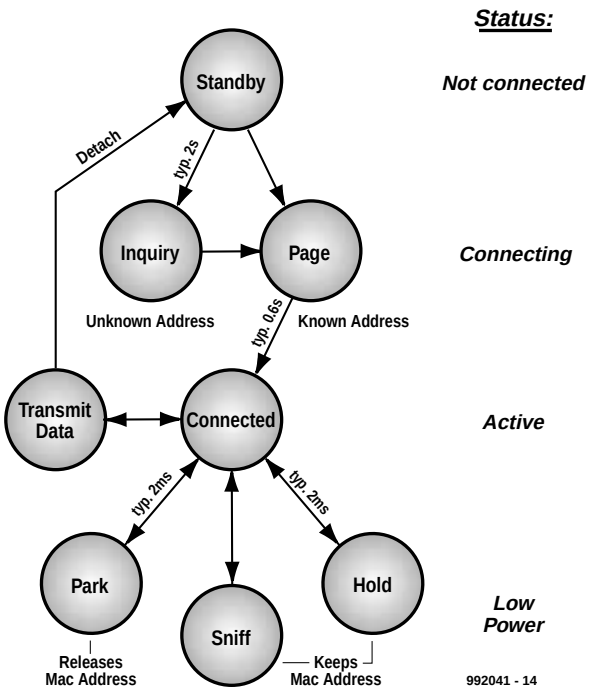


Figure 4. Bluetooth module state diagram

A Bluetooth module that is not yet connected transmits a call via the wake-up carriers, which is answered by any other nearby Bluetooth radio(s). After an introductory handshake, the two devices set up a piconet, in which the device that first issued the call takes on the role of master. Based on its address, it establishes the hopping sequence for this piconet. The slave device, and any other devices that later join this piconet, synchronize themselves to the clock rate of the master device.
Figure 4 shows the state diagram of a Bluetooth module. Starting from the Standby mode, it moves to the Inquiry state on detecting a second device. In this state, it transmits a general call followed by an address request. Once the address of the other device has been determined, or if this was already known in the Standby mode, the module transfers to the Page mode. After a typical delay of 0.6 second, the connection becomes active (Connected state). Data transmission can now take place (in the Transmit Data state). After a successful data transmission, the Bluetooth module can either return to the Standby mode or enter one of three low-power (energy-saving) modes.
The first low-power mode is the Hold mode, in which the device remains an active member of the piconet. When an internal timer in the slave device times out, the slave briefly makes itself known to the master before restarting the timer. If necessary, the slave can leave the Hold mode immediately in order to transfer data. The master can force a slave into the Hold mode, but a slave can also voluntarily enter the Hold mode.
In the second low-power mode, the Sniff mode, the slave is programmed to periodically 'listen' to the piconet to determine whether there is a data transfer waiting for it.
The third low-power mode is the Park mode. In this mode, the slave drops out of the piconet and makes its Media Access Control (MAC) address free (each member of a piconet has a MAC address, with a value of 0 to 7). After this, it remains passive, and its only activity is to maintain synchronization with its piconet master at relatively long time intervals.

Interesting Bluetooth applications

In the first instance, Bluetooth should eliminate cables for connections between laptop or desktop computers and printers, scanners and fixed Local Area Networks (LANs). With it, even (wireless) keyboards, mice and joysticks or trackballs need not necessarily have a line-of-sight path to the computer. Other imaginable applications relate to laptop computers and cellular phones. For example, if you write your e-mail messages in the aeroplane using your laptop, you would no longer have to worry about how to send them. As soon as you leave the aeroplane and switch

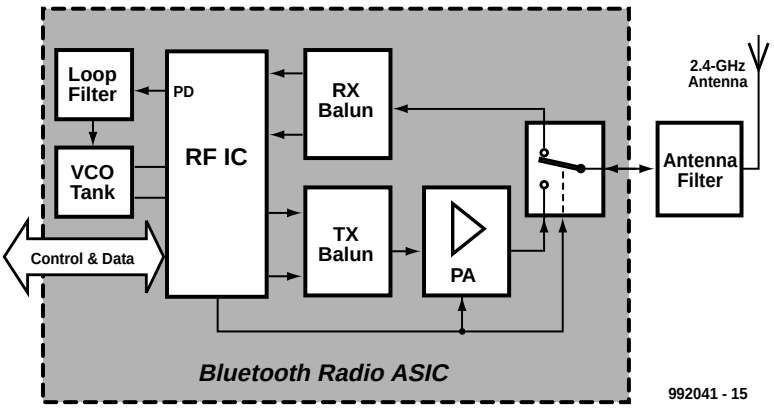


Figure 5. Block diagram of a Bluetooth radio module

on your cellular phone again, the laptop could make a Bluetooth connection and send the e-mail. The use of a Bluetooth radio module inside the aeroplane should also be allowed, due to its low transmitted power level (100 mW maximum).

In the future, you could also dispense with cables when using a cellular phone with a hands-free installation. You could leave your cellular phone switched on in your pocket and enjoy a wireless link to the headset of the hands-free unit. The use of a hands-free unit makes an important contribution to safety for commercial vehicles as well. Finally, you could equip your laptop with speakers and a microphone. The speech signal could be passed on to the cellular phone in your briefcase via a wireless Bluetooth link. This would make it possible to transfer speech, data and graphics using only one device, the laptop computer.

One very practical suggestion is to use Bluetooth for automatic file synchronization between a laptop computer, a desktop computer and a Personal Digital Assistant (PDA). Whenever these devices are located in the same vicinity, they could independently exchange e-mail messages, appointments and addresses, in order to bring each other up to date. It would even be possible for a field-service technician (for example) to use a Bluetooth cellular phone to automatically update or modify his appointments calendar, based on information from the main office. In future conferences and meetings, the participants could bring along their laptops with Bluetooth interfaces and set up a spontaneous network. This would allow the exchange of graphics, texts and data, or the wireless remote control of a projector.

Wireless Internet access using Bluetooth should allow for more freedom of movement. Internet pages could reach your laptop via wireless links from a cellular phone, locally-installed modem or your firm's LAN. A Bluetooth cellular phone could automatically switch from GSM to DECT operation, both at home and in the office, as soon as it makes contact with a local cordless-phone base station. This would allow you to simply use the same unit for all your phone conversations.

From the world of entertainment electronics, there are suggestions for using Bluetooth as the basis for wireless connections to (for example) a video camera or a still camera. The camera could pass the image via a cellular phone to the mains network, or save it on a laptop, all via a wireless link. It would be possible to send still pictures as spontaneous 'electronic postcards' via Bluetooth to your cellular phone and thereby to their ultimate destinations. Remote control of television sets and stereo installations could also be implemented using Bluetooth, which would eliminate the annoying need to search for a line-of-sight path to the equipment.

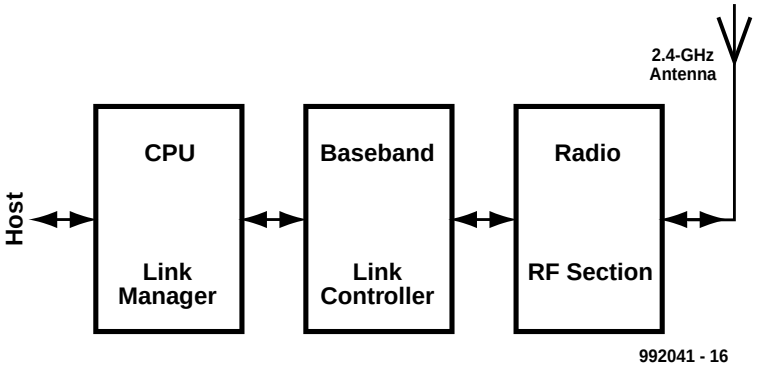


Figure 6. Block diagram of a Bluetooth module

The first Bluetooth components

Ericsson has developed a Bluetooth Development Kit in cooperation with Symbionics. It includes extensive documentation and design-support software. The baseband processor is provided by VLSI, and comes from the VWS26000 Bluetooth family [6]. The radio module (see **Figure 5**), which is a hybrid, is an Ericsson product (PBA 313) [4]. You can also obtain a Bluetooth Developer's Kit from Philips Semiconductors. The baseband portion is once again a VLSI ASIC, and the radio module contains the UAA-3558 Bluetooth transceiver. This kit contains two identical Bluetooth daughterboards that can be used to set up an initial radio link. **Figure 6** shows the typical structure of a Bluetooth module. The firm Cambridge Silicon Radio [7] is working on single-chip Bluetooth components with integrated radio modules. The Bluecore™01, Bluecore™02 and Bluecore™03 ASICs are intended to incrementally incorporate additional Bluetooth functions. The size of a complete module should ultimately shrink to that of a postage stamp. Finally, numerous semiconductor manufacturers, such as Temic, Philips and Motorola, offer 2.4-GHz transceiver ICs especially for use in Bluetooth radio modules. All Bluetooth modules must pass a BSIG certification procedure in order to ensure compatibility.

Competition for Bluetooth

There is also competition for Bluetooth as a short-range radio networking technology. For some time now, it has been possible to transfer data between a PC and its peripheral equipment using an infrared-light interface that complies with the Infrared Data Association (IrDA) protocol. However, infrared links absolutely require free line-of sight

paths and can bridge a distance of a few metres at best, with viewing angles of only 60 degrees. The advantage of Bluetooth here is that it can even pass through walls into neighbouring rooms (see Reference 8 for a comparison of IrDA and Bluetooth). In the USA, several firms have joined forces to promote HomeRF [9] as a standard similar to Bluetooth. The Shared Access Working Protocol (SWAP), generated by the HomeRF Working Group (HRFWG), allows 127 devices per network with an effective range of 50 metres. Surprisingly enough, some of the founders of Bluetooth are also among the initiators of the HRFWG: Ericsson, IBM and Intel, along with Compac, Hewlett-Packard, Microsoft, Motorola and Philips, play a decisive role. In any case, this group has only a few hundred members, while the BSIG can already boast of more than 1000 interested parties. In addition, if you want to obtain a copy of the HomeRF specification you must first pay a $500 membership fee. The DECT standard for cordless phones is also a candidate for wireless LANs. It works in a dedicated frequency band around 1.8 GHz, and has a wider effective range (up to 500 meters) due to its greater transmitted power. DECT and Bluetooth perform equally well for wireless voice communications and for data transfers at low data rates (up to 64 kb/s), but Bluetooth can go beyond this to reach rates of up to 721 kb/s. A consortium based on 3Com, IBM, Intel, AMD, Compaq, Hewlett-Packard, Cisco and Lucent is promoting networking using telephone lines. They call themselves the Home Phoneline Networking Alliance (HomePNA) [10], and their system will probably be used primarily in the USA, where it is common practice to provide every room with a telephone outlet. HomePNA uses the frequency region between 6 and 10 MHz, which lies above that reserved for the Asynchronous Digital Subscriber Line (ADSL) service [11]. Diverse powerline communications systems are also striving to be accepted for home networks. Their advantage is that mains outlets are present in every room. However, the high levels of interference on the mains wiring means that only quite limited data rates are possible.

Future prospect

Bluetooth has the best prospects of all the wireless LAN systems, which are quickly establishing themselves worldwide. In addition to extensive applications in the area of data communications, it should find areas of application in entertainment electronics and commercial transportation. Following the initial applications presented by Ericsson and Nokia, you can expect to see numerous other applications in the coming year. The CeBIT 2000 Trade Fair should prove interesting in this regard, and you can look forward to being able to purchase the first Bluetooth systems in the near future.

(992041-1)

Article editing (German original):
E. Krempelsauer

Literature and links:

[1] www.bluetooth.com
[2] www.bluetooth.net
[3] www.zdnet.co.uk/news/specials/1999/04/bluetooth/
[4] http://bluetooth.ericsson.se/default.asp
[5] www.intel.com/mobile/bluetooth/
[6] www.vlsi.com
[7] www.cambridgesiliconradio.com/
[8] Comparison of IrDA and Bluetooth: www.countersys.com/tech/bluetooth.htm
[9] HomeRF: www.homerf.org
[10] HomePNA: www.homepna.org
[11] Fast Internet access by ADSL, Elektor Electronics, November 1999
[12] www.bluetooth.rsd.de/ and www.bluetooth-testing.com

King Bluetooth and his friends

The name 'Bluetooth' was chosen in memory of the Danish king Harald Blaatand (Harold Bluetooth in English), who lived from 910 to 986 AD. He christianized and united large parts of Scandanavia. The illustration (courtesy of Rohde & Schwarz) shows a runic stone that was found in King Bluetooth's home city of Jelling in central Jutland. It clearly indicates that the king can be considered to be an early proponent of wireless communication between cellular phones and laptops.

The choice of name also naturally refers to the two main initiators of the system, Ericsson Mobile Communications of Sweden and Nokia Mobile Phones of Finland. As telephone manufacturers, they contributed preparatory work to the subject of wireless LANs.

In the meantime, King Bluetooth has found a lot of friends, as indicated by the following excerpt from the list of members of the Bluetooth SIG (see Reference 1 for the full list):

AKG Acoustics, Alcatel, Analog Devices, AMD, Bang & Olufsen, Boeing, Bosch, Casio, Compaq, Dell, Fujitsu, Grundig, Hagenuk, Hewlett-Packard, ICO, LEGO, LG Electronics, Logitech, Lucent, Mitsubishi, Motorola, NCR, National Semiconductor, Philips, Psion Computer, Qualcomm, Rohde & Schwarz, Samsung, Sennheiser, Seiko Epson, Sharp, Siemens, TDK, Telia, Temic, Texas Instruments, VLSI Technology, Volvo, 3Com.

audio DAC 2000

Part 3: practical matters

Readers who have studied and absorbed Parts 1 and 2 of this article will be fully aware of how the Audio DAC 2000 works. What remains to be discussed is the actual building of the converter, and this is done in this third and final part.

Design by T. Giesberts

POWER SUPPLY

Although the summary implies that all parts of the digital-to-analogue converter (DAC) have been discussed, this is not entirely true, because the power supply has not yet been described.

It was seen in Part 2 of this article that the converter ICs need a symetrical ±5 V supply and that this was derived via regulators IC13 and IC14 from the ±12 V supply line for the analogue circuits. Since it is important to keep the supply lines to the converter ICs as short as possible, the regulators are housed on the DAC board.

The receiver section and some other circuits on the DAC board need a single +5 V supply and a symmetrical ±12 V supply. These voltages are produced with the aid of regulators IC15–IC17, which, together with the other components of the power supply, are housed on a separate board, the PSU board.

The circuit of the power supply is shown in **Figure 5**. Note that the +5 V line for the digital circuits is isolated from the ±12 V lines for the analogue circuits.

The earth lines of the two supplies are interlinked on the DAC board between the digital filter and the converter ICs (that is, JP3).

Obviously, the supply consists of not only regulators, but also bridge rectifiers and smoothing capacitors. Resistors R55, R56, and R58 between the rectifiers and smoothing capacitors limit the charging current to the capacitors at power-on, and any resulting interference.

The secondaries of the relevant mains transformers are linked to K11 and K13 respectively. The choice of transformer is up to the constructor, although some suitable models are specified in the parts list.

The power supply is conveniently built on the 'transformer board' described elsewhere in this issue. This board is designed to house all the components required for the present power supply.

PRINTED-CIRCUIT BOARDS

As already mentioned in Part 1, the Audio DAC 2000 is contained on four individual printed-circuit boards: one for the ±12 V and +5 V power supplies; one for the digital audio receiver with display driver; one for a 2-digit LED display; and one for the digital/analogue circuits, the digital filter, the DACs and the analogue output stage. These boards are sections of the double-sided PCB shown in **Figure 6**. This high-quality board is available through our Readers Services. Before any work is carried out, these four sections should be separated from one another along the milled cutting lines, either by snapping or cutting along the lines

It is important to construct the various circuits according to the board layouts and the parts list. It is important that the orientation of the ICs and the polarity of the electrolytic capacitors are strictly observed, since any deviation results unfailingly in a non-working unit.

The DIP switches, S1–S4, are best soldered directly to the board. An exception is S2 if it is foreseen that processor (that is, software) control may be used at a later stage. In that case, the switch may be housed in a good-quality 8-way IC socket, but even then the board connector should be soldered in place at a later stage.

All supply lines are connected to the boards via terminal blocks that facilitate the wiring or servicing of the relevant circuits. The +5 V line for the digital section is linked to the DAC board. The receiver board is powered via the link between K3 and K5. Some protection against (a too) high supply voltage is provided by diode D4 on the DAC board.

The LED display board is linked to the receiver board via a 10-way flatcable. One end of this cable is connected permanently to the board via a 10-way board connector. The other end is terminated into a 10-way socket. Take good care to use the correct length of cable. The displays are soldered directly to the board.

The digital audio receiver, IC1, is soldered directly to the receiver board. Take care not to damage this IC by electrostatic discharges. Sockets may be used for IC4 and IC5. Crystal oscillator IC3 is also best soldered directly to the board, since it is then as close as possible to the ground plane.

Start populating the DAC board by soldering IC6 to the board — see **Figure 7**. This tiny SMA (surface-mount assembly) IC is housed in a 28-pin SSOP case, whose pins are spaced at only 0.65 mm. This requires extreme care, a tiny soldering iron tip, and possibly a magnifying glass to check the soldering work. Take good care not to overheat the IC: take a pause between soldering, say, 2 or 3 pins at a time.

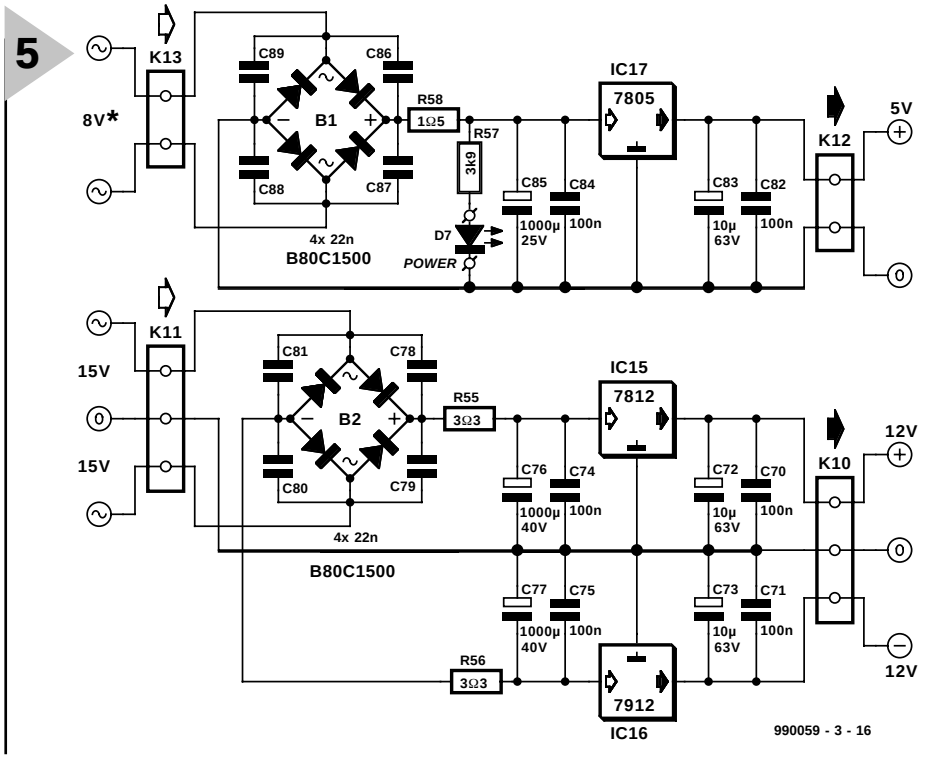


Figure 5. Circuit diagram of the power supply for the Audio DAC 2000. The +5 V line for the digital circuits is isolated on the board from the ±12 V line for the analogue section.

Next, solder the converter ICs, IC_7 and IC8, in place. These are housed in a standard 20-pin SMD case (SOIC), and are easily soldered. It is best to fit op amps IC9–IC12 in good-quality IC sockets.

Capacitors C27–C38 in the analogue section are 1% close-tolerance types in a square radial format, with the terminals placed at two opposing corners. These types are manufactured by EMZ. Their pitch is standard (7.1 mm), so they could be replaced by metallized film polystyrene or polypropylene types. It should be borne in mind, however, that the larger tolerances of these types may result in significant changes in the frequency and phase responses. The EMZ capacitors specified carry a thin dash that indicates which terminal is linked to the outer layer. Make sure that this pin is linked to ground or to the output of an opamp: this makes the analogue section less sensitive to interference. The same applies to axial capacitors C25 and C26: place the band on these at the output side of opamps IC9 and IC11.

The relays are soldered directly to the board. Do not forget wire bridges JP2 and JP3: these are permanent links which may be made in rather thicker wire than usual.

A final practical hint. To improve the channel separation at high frequencies, it is advisable to shield the left- and right-hand sections of the analogue output filter from one another. This is best done by placing a small (86×13 mm) tin-plate screen between Re2 and IC12. The screen stretches from the edge of the board to DIP switch S4: its position is indicated in **Figure 6** by a dashed line. At the ends, scratch away some of the lacquer on the board with a sharp pen knife to ensure that the screen makes good contact with the copper area at the top of the board which functions as ground plane — soldering the ends of the screen to the copper is even better. In the prototype, the addition of the screen improved the channel separation by 12 dB at 20 kHz.

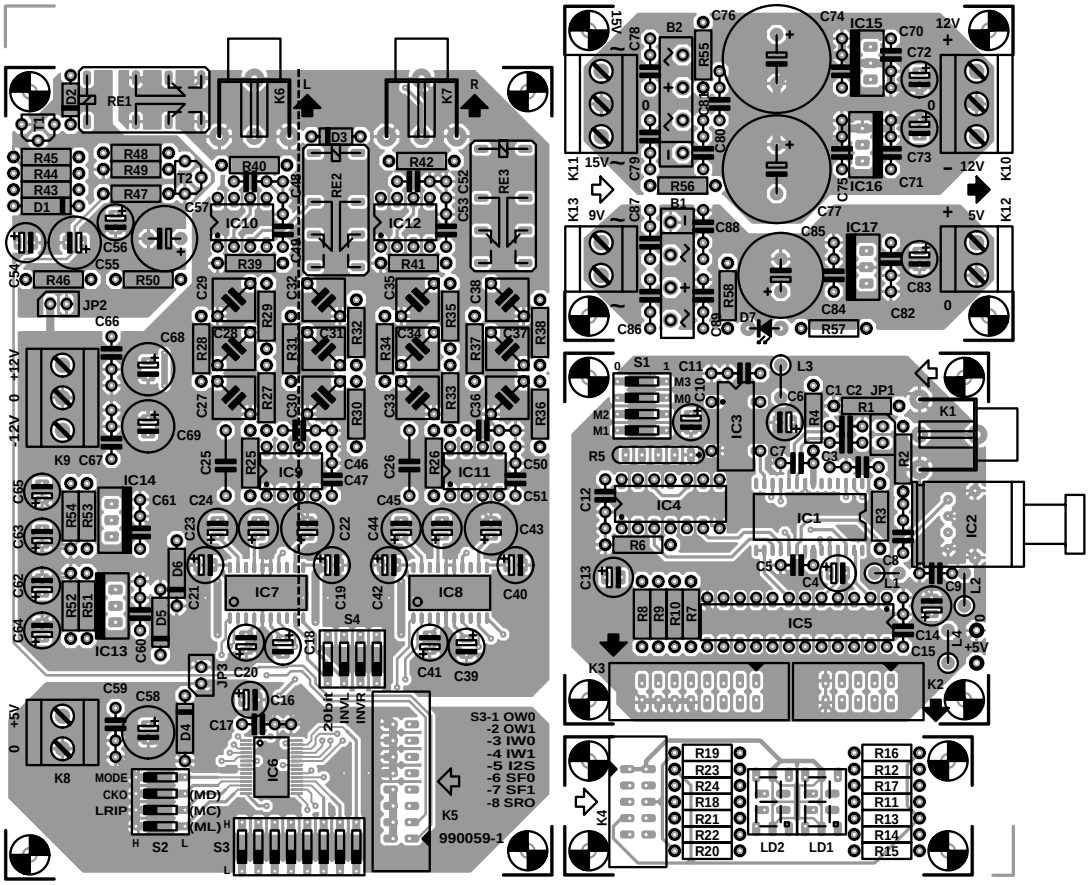
ENCLOSURE

When the four boards have been completed and checked for possible building or soldering errors, they must be combined into a complete Audio DAC 2000 and housed in a suitable enclosure. The most suitable enclosure is a sturdy metal case, which, as far as appearance is concerned, should preferably match the audio installation with which it is to be used.

The prototype is housed in a Monacor Type UC251/SW enclosure. This is 435 mm wide, 230 mm deep, and 44 mm high — see **Figure 8**. In some countries in which *Elektor Electronics* appears, Monacor products are sold under the brand name 'Monarch'.

The manner in which the boards are arranged in the enclosure is optimal and constructors are well advised

6



reproduced at 85% of actual size

Figure 6. The double-sided board must be divided into four sub-boards along the fraised lines.

COMPONENTS LIST

Resistors:
R1 = 75Ω
R2 = 220Ω
R3 = 470Ω
R4,R50 = 4Ω7
R5 = 4-way 10kΩ SIL-array
R6 = 10kΩ
R7-R10 = 22Ω
R11-R24 = 820Ω
R25,R26 = 2kΩ49 1%
R27,R30,R33,R36 = 3kΩ57 1%
R28,R34 = 4kΩ12 1%
R29,R35 = 3kΩ92 1%
R31,R37 = 3kΩ65 1%
R32,R38 = 3kΩ32 1%
R39,R41,R45,R49 = 1MΩ
R40,R42 = 100Ω
R43,R44,R47,R48 = 150kΩ
R46 = 10Ω
R51,R53 = 249Ω 1%
R52,R54 = 750Ω 1%
R55,R56 = 3Ω3
R57 = 3kΩ9
R58 = 1Ω5

Capacitors:
C1,C2 = 10nF ceramic
C3 = 68nF
C4,C6,C10,C16,C62-C65,C72, C73,C83 = 10µF 63V radial
C5,C7 = 47nF ceramic
C8,C9,C11,C12,C15,C17,C46-C53, C59,C60,C61,C66,C67,C70,C71, C74,C75,C82,C84 = 100nF ceramic
C13,C14,C23,C24,C44,C45, C54 = 47µF 25V radial
C18-C21,C39-C42 = 4µF7 63V radial
C22,C43,C58,C68,C69 = 100µF 25V radial
C25,C26 = 47pF [1] axial (EMZ)
C27,C33 = 2nF2 1% [1] (EMZ)
C28,C34 = 4nF7 1% [1] (EMZ)
C29,C35 = 330pF 1% [1] (EMZ)
C30,C36 = 1nF 1% [1] (EMZ)
C31,C37 = 1nF5 1% [1] (EMZ)
C32,C38 = 270pF 1% [1] (EMZ)
C55 = 220µF 25V radial
C56 = 1µF 63V radial
C57 = 470µF 25V radial
C76,C77 = 1000µF 40V radial
C78-C81,C86-C89 = 22nF ceramic
C85 = 1000µF 25V radial

[1] *polystyrene/polypropylene*

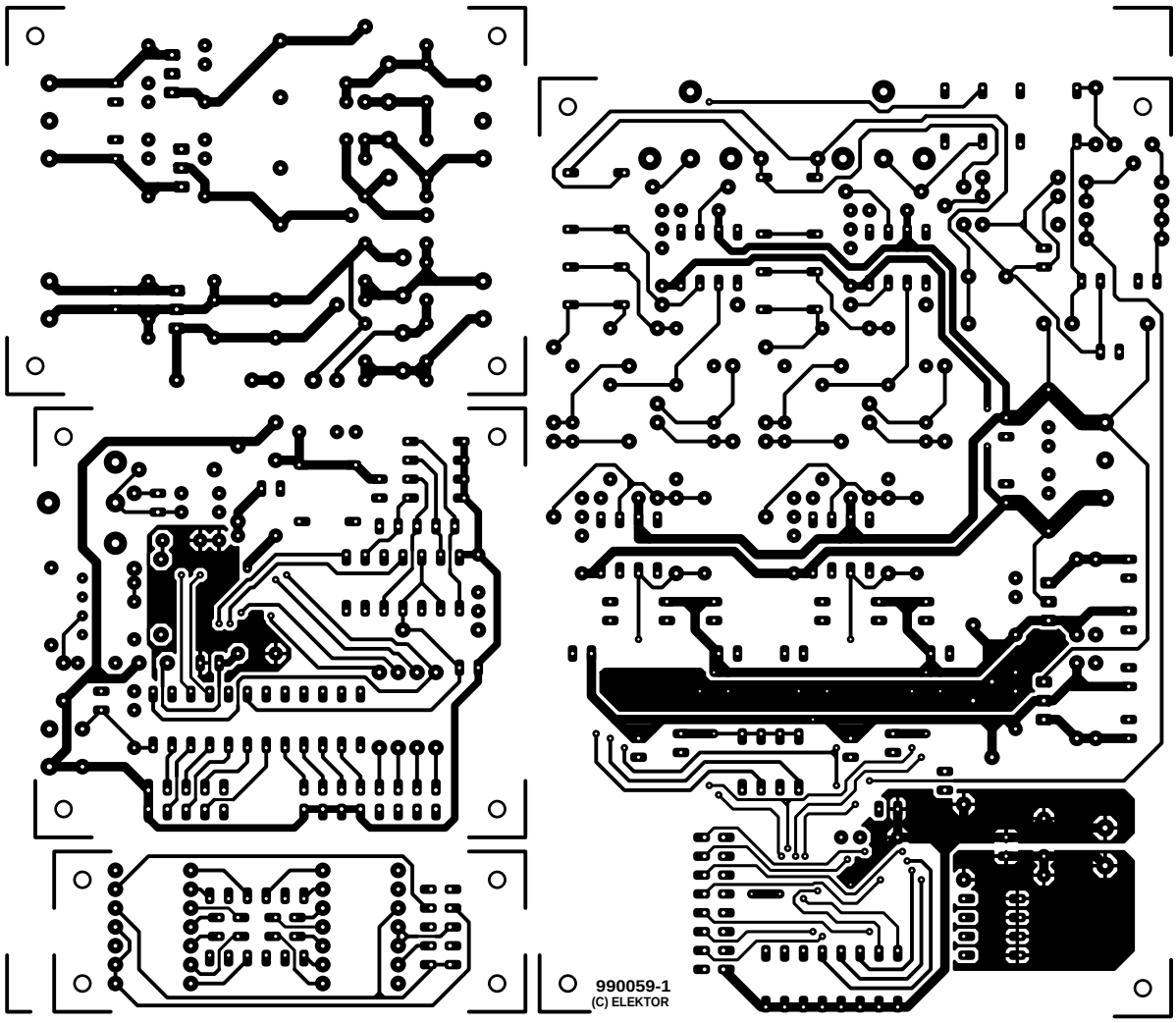
EMZ, Elektromanufaktur Zangenstein
Hanauer GmbH & Co.
Siemensstrasse 1
D-92507 Nabburg
Germany
Tel. +49 9433 898-0
Fax +49 9433 898-188

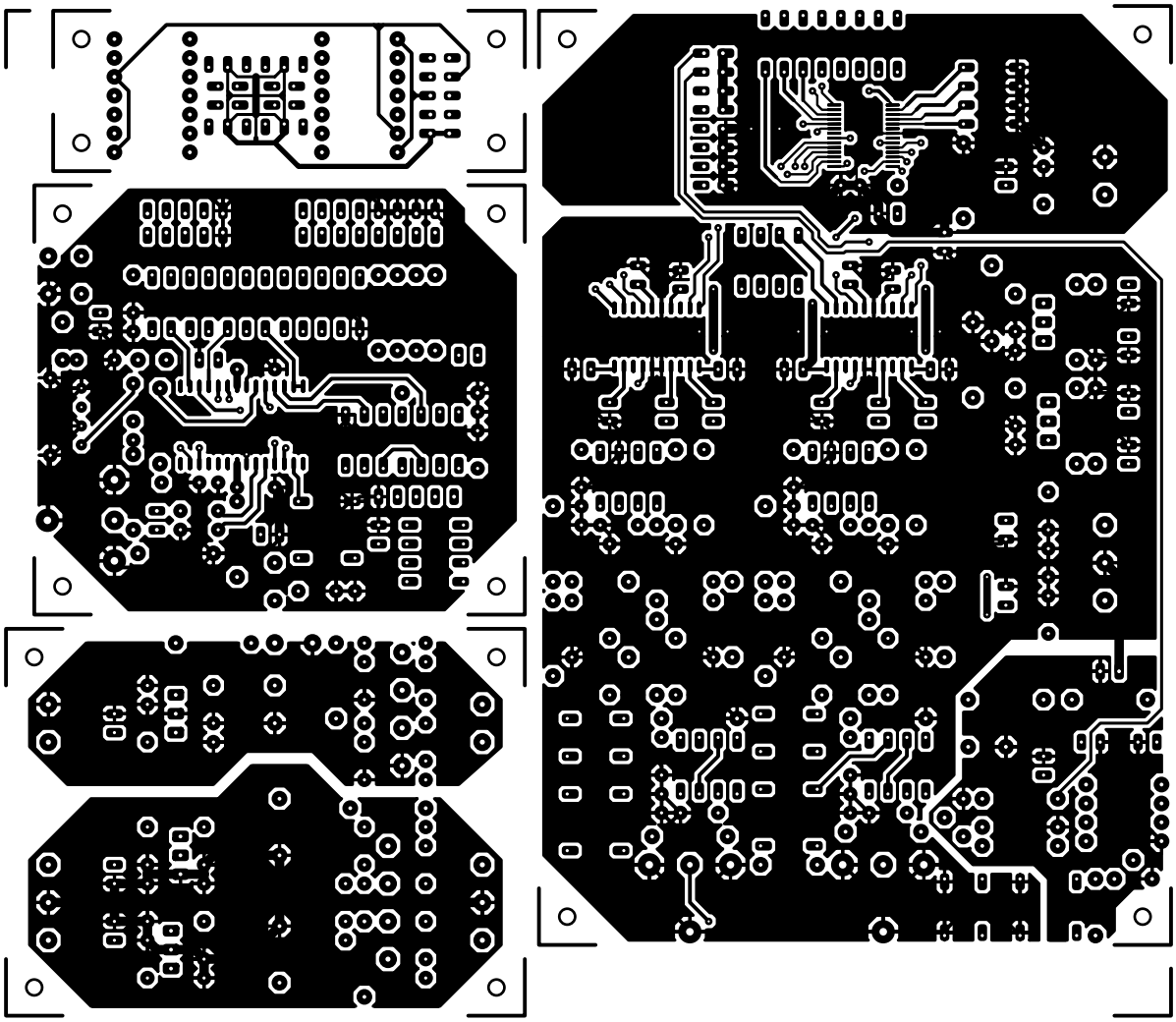
Inductors:
L1-L4 = 47 µH

Semiconductors:
D1 = 1N4001
D2,D3 = 1N4148
D4,D5,D6 = 5V6 1W3 zener diode
D7 = LED, red, high-efficiency
LD1,LD2 = HDN1075O (Siemens)
T1,T2 = BC517
IC1 = CS8414-CS (Crystal)
IC2 = TORX173 (Toshiba)
IC3 = 6.144MHz SG531P (Seiko Epson)
IC4 = 74HCT32
IC5 = GAL22V10B-25LP (ready-programmed, order code **996530-1**, see Readers Services pages)
IC6 = DF1704E (Burr-Brown)
IC7,IC8 = PCM1704U (Burr-Brown)
IC9...IC12 = OPA627AP (Burr-Brown)
IC13 = LM317 (TO220)
IC14 = LM337 (TO220)
IC15 = 7812
IC16 = 7912
IC17 = 7805

Miscellaneous:
JP1 = 2-way pinheader + jumper
JP2,JP3 = wire link *
K1,K6,K7 = cinch socket, PCB mount (Monacor/Monarch type T-709G)
K2 = 10-way boxheader
K4 = 10-way PCB-connector (for flat-cable)
K3,K5 = 16-way boxheader
K8,K12,K13 = 2-way PCB terminal block, raster 5 mm
K9,K10,K11 = 3-way PCB terminal block, raster 5 mm
S1,S2,S4 = 4-way DIP-switch
S3 = 8-way DIP-switch
B1,B2 = B80C1500, rectangular case
Re1,Re2,Re3 = V23042-A2003-B101, 12V/600 Ω (Siemens)
Supply transformers: 2x15 V/4 VA (e.g. Block FLD4/15; Hahn BVUI 3020165; Monacor FTR-415), and 2x8(or 9) V/4 VA (e.g.. Block FLD4/8; Hahn BVUI 3020161; Monacor FTR-49 – see transformer board elsewhere in this issue)
PCB, order code **990059-1**, see Readers Services pages.

* see text


990059-1
(C) ELEKTOR



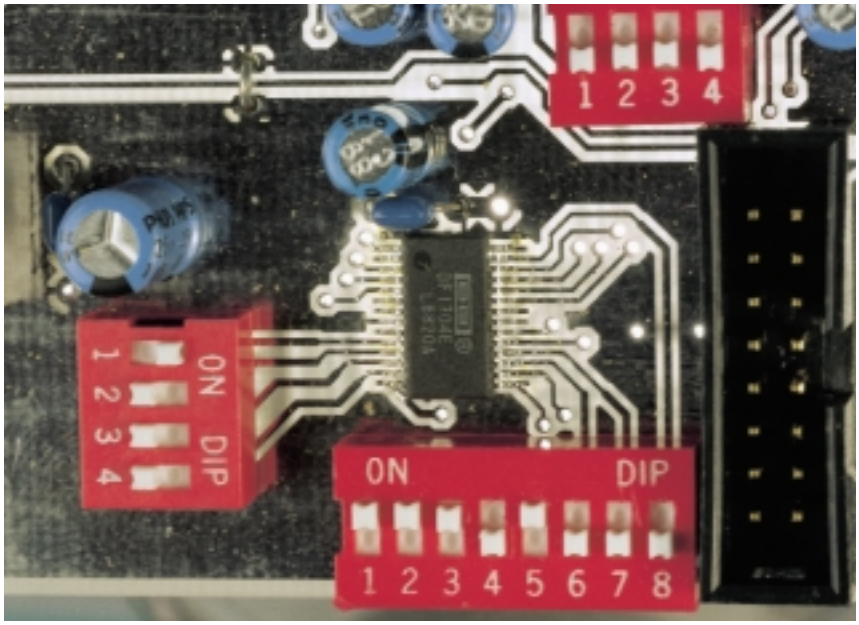


Figure 7. Soldering IC6 into place requires a steady hand, good eyesight and precision tools.

to use the same arrangement: the DAC board in one corner, the receiver board next to it, the supply board in front of this, and the transformer board in the remaining corner.

The only items to be fitted on the front panel are the mains on/off switch and the display that shows the sampling rate. If desired, power diode D7 may be added to this, but this is not really necessary since LD1 and LD2 function very well as on/off indicator.

The interwiring may be gleaned from Figure 8, but is summarized for convenience's sake.

- K2 on the receiver board is linked to K4 on the display board just behind the front panel via a 10-core flatcable.
- K5 on the DAC board is linked to K3 on the receiver board via a 16-core flatcable. This link also connects power to the receiver board. Mind the orientation of pin 1 on the connectors.
- K12 (+5 V) on the supply board is linked to K8 on the DAC board via two cables.
- K10 (±12 V) on the supply board is linked to K9 on the DAC board via three cables.

FINALLY

Testing a digital-to-analogue converter by ear is hardly possible or sensible. Noticeable differences, such as can be detected in the case of loudspeakers, cannot be expected. Nevertheless, a test audience felt that the DAC 2000 sounded better than a number of other available types of DAC. They found the sound cleaner and the stereo image clearer.

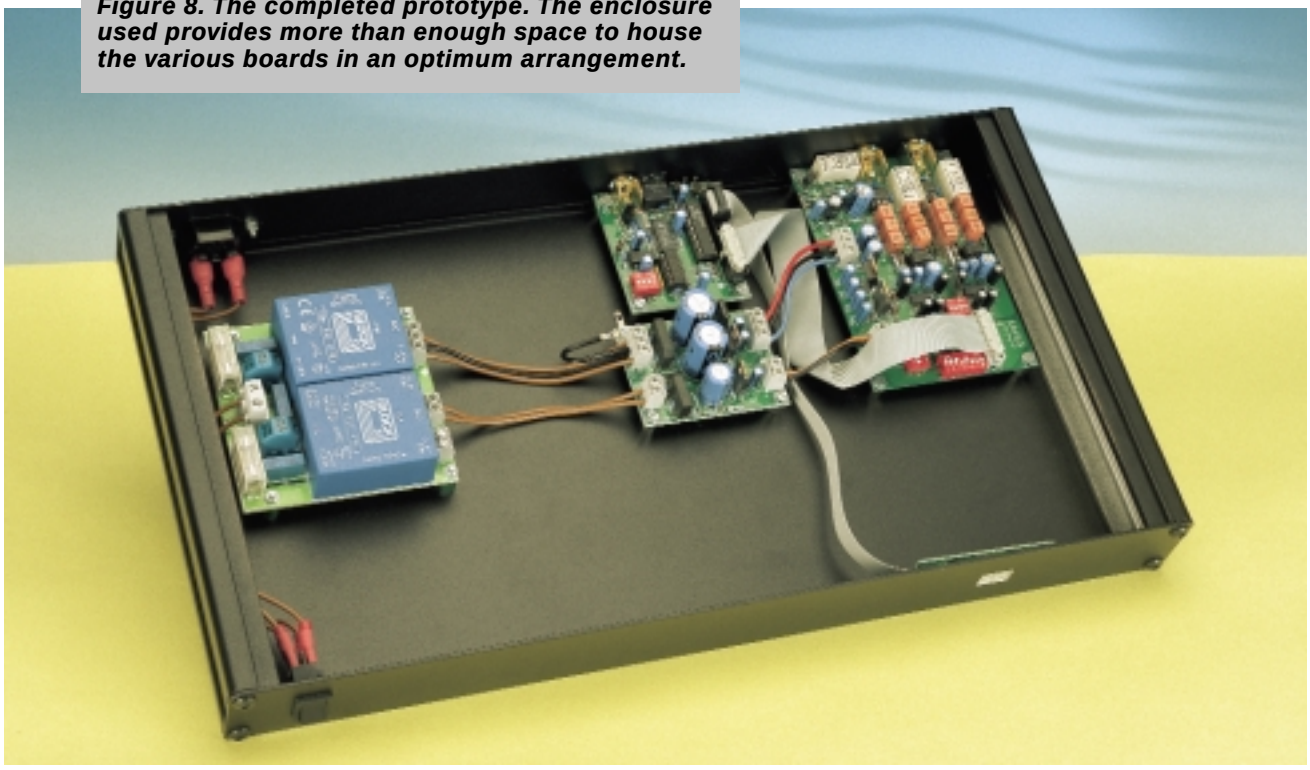
The test results in the box give a further judgment of the quality of the converter. They call for a few comments.

- The bandwidth of sampling frequencies 32 kHz, 44.1 kHz, and 48 kHz, is exactly equal to half the sampling rate, since at these frequencies the bandwidth of the analogue filter is larger than that of the steep-skirted digital filter. At 88.2 kHz and 96 kHz, the bandwidth is determined by the analogue filter.
- The THD + N at a sampling rate of 96 kHz is measured at a bandwidth of 22 kHz, because at lower sampling rates the analogue output filter has a bandwidth of 26 kHz. The reduction gives a more honest comparison of the three measurements.

[990059-3]

Text: S. van Rooij

Figure 8. The completed prototype. The enclosure used provides more than enough space to house the various boards in an optimum arrangement.



Technical specification

Properties

- *1 coaxial input and 1 optical input*
- *suitable for sampling rates of 32–96 kHz*
- *2-digit readout of sampling rate*
- *8× oversampling*
- *24-bit digital filter*
- *24-bit digital-to-analogue converters*
- *digital de-emphasis*
- *switchable third-order analogue output filter*
- *isolated supply lines for digital and analogue sections*

Electrical characteristics

Nominal input voltage at coaxial input	*0.5 V_{pp} into 75 Ω*
Nominal output voltage	*2.1 V r.m.s.*
Frequency range (–3 dB)	*0–f_S/2 (f_S = 32/44.1/48 kHz)*
	0–42 kHz (f_S = 88.2/96 kHz)
Amplitude at 20 kHz	*–0.94 dB (f_S = 32, 44.1, 48 kHz)*
	–0.66 dB (f_S = 88.2, 96 kHz)
Bandwidth analogue filter	*26 kHz (Butterworth at f_S = 32/44.1/48 kHz)*
	42 kHz (Bessel at f_S = 88.2/96 kHz)
Output impedance	*100 Ω*
Signal-to-noise ratio	*≥ 114 dBA*
THD + N (1 kHz, B = 80 kHz)	*0.0016% (44.1 kHz, 16-bit)*
	0.001% (48 kHz, 24-bit)
	0.0008% (96 kHz, 24-bit, B = 22 kHz)
IMD (60 Hz/7 kHz, 0 dB)	*0.0035%*
Linearity error	*< 0.5 dB/–90 dB (according to datasheet)*
	0.2 dB/–110 dB (measured)
Channel separation (1 kHz)	*> 115 dB*
Dynamic range	*> 100 dB*

Measurements were made with switch settings as follows

S1	***S2***	***S3***	***S4***
-1 off	*-1 on*	*-1 on*	*-1 off*
-2 off	*-2 off*	*-2 on*	*-2 off*
-3 off	*-3 off*	*-3 on*	*-3 off*
-4 on	*-4 off*	*-4 off*	*-4 NC*
		-5 on	
		-6 off	
		-7 off	
		-8 off	

Performance characteristics

For completeness' sake, the electrical specifications are complemented by a set of performance characteristics. Some comments on these are:

*Curve **a** is the frequency response of the analogue output filters, measured by injecting a current into the current-to-voltage converters, so that the first filtering by C25 and C26 is included.*

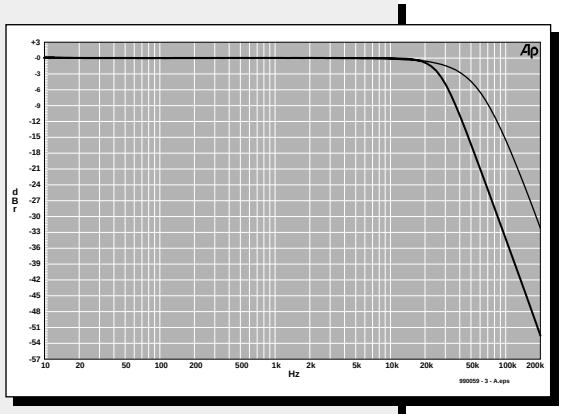
*Curve **b** is the THD + N characteristic at full drive. This was measured with the aid of a test compact disk (16-bit, 44.1 kHz). The increase in distortion above 3 kHz is small and remains below 0.005% up to 20 kHz. At higher audio frequencies the speed of the DACs will of course have an effect.*

*Curve **c** illustrates the linearity of the DACs. The amplitude sweep was carried out with test tones of 400 Hz, provided with dither to make measurements up to –110 dB at 16 bit possible.*

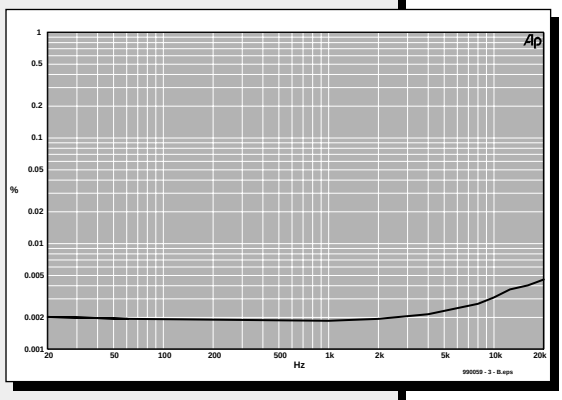
*Curve **d** represents the channel separation between the two channels from 40 Hz upwards. Below this only the noise threshold would be measured. Even at 20 kHz the channel separation is >88 dB in both cases. Measurements were made with the tin plate screen fitted as mentioned in the text.*

*Curve **e** shows the frequency spectrum at 1 kHz at full drive and a sampling rate of 48 kHz at 24 bit. Note that all harmonics are well below –100 dB.*

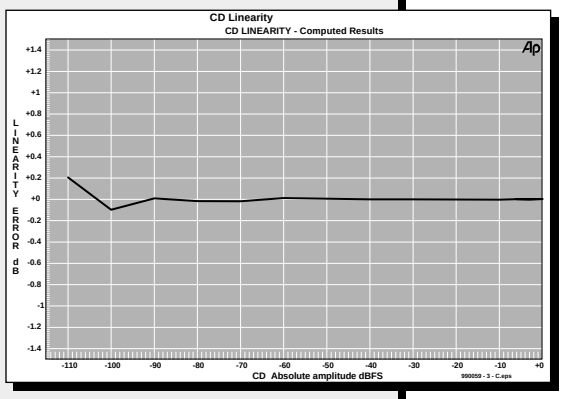
a



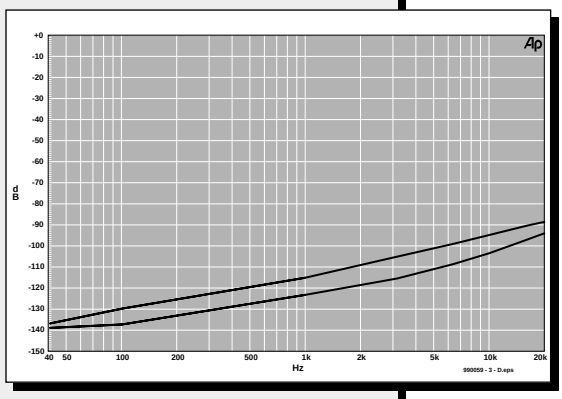
b



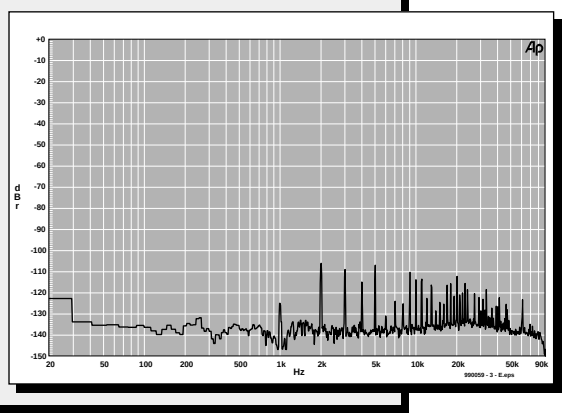
c



d

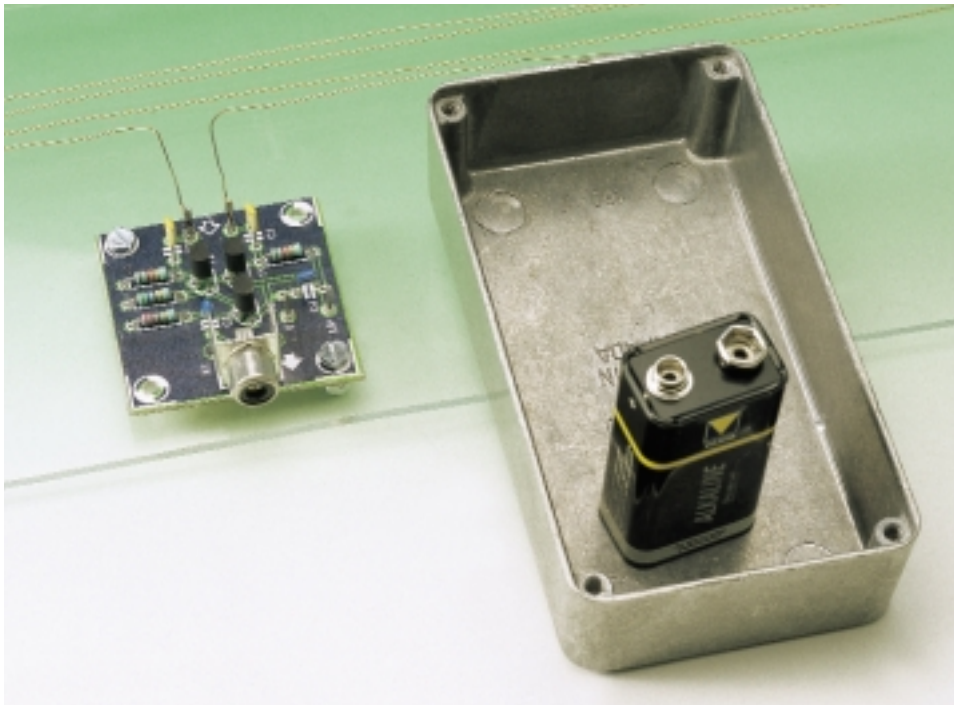


e



wideband active loop antenna

for general-coverage receivers



There's nothing better than a long-wire antenna for short-wave reception, but not everyone has the necessary space. The active loop antenna described in this article can be an excellent alternative to a long-wire antenna where space is restricted. Not only is it easy to build, it also provides outstanding performance and is very inexpensive.

design by G. Baars

Anyone who wants to use a general coverage receiver for more than just listening to the BBC World Service or the Voice of America (VOA), soon discovers that a simple telescopic antenna has its limitations. The efficiency of such an antenna is very low, and it also picks up a lot of interference when used indoors. If you take long-distance short-wave reception a bit more seriously, you will soon find that you need something better.

What then should it be? Whole books have been written about antenna technology, and there are countless types and varieties of antennas. Before plunging into the forest of possibilities, it's a good idea to first consider what requirements the antenna in question should meet.

For a short-wave or general-coverage receiver, it is important that the bandwidth of the antenna is sufficient to cover a range of (say) 3 to 30 MHz, and that it has a reasonable efficiency within that range. If you are looking for the simplest solution, then the 'good old' long-wire antenna is an excellent choice. Of course, there are numerous other types of antennas that are also suitable, but their construction is in most cases considerably more complicated.

The common feature of the these types of antennas is that they take up a considerable amount of space, which is reason enough to regard them as typical outdoor antennas. What can you do if you have nowhere near enough space for such an antenna? Must you content yourself with the telescopic antenna, or are there other affordable possibilities?

TAKE A LOOK AT THE WINDOW

A loop antenna is a good indoor alternative for a long-wire antenna. Some people may respond to the term 'loop antenna' with aversion, since they have the idea that the only place for such antennas is a museum. This is

absolutely not true! The basic loop antenna design can still be used to construct quite useful antennas, and these antennas have the significant advantage that they are excellent for indoor use, due to their form and characteristics. For example, if such an antenna is wound around a window frame, it takes up hardly any space and is also practically invisible. In addition, a loop antenna has the desirable characteristic of reacting only to the magnetic component of the transmitted signal, which means that it rejects a large number of electrical disturbances. These features of the loop antenna — compactness and interference rejection — should not be underestimated.

1

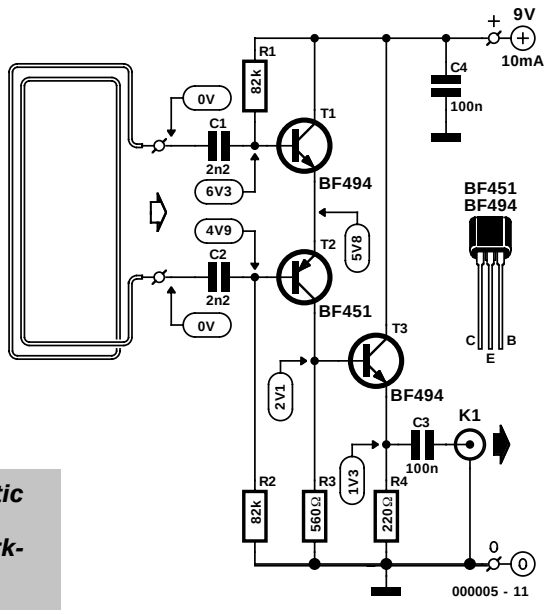


Figure 1. The schematic diagram of the active loop antenna is remarkable for its simplicity.

LET'S GO ACTIVE!

In terms of effective surface area, the antenna described here can be roughly compared to an average long-wire antenna, since it is made from 10 to 15 metres of wire. However, since it is folded into a loop and installed indoors, the loop antenna has different characteristics and its efficiency is significantly lower.

To deal with this, there is actually no other choice than to implement an 'active' version of the loop antenna, which simply means providing it with a built-in amplifier. At the same time, the amplifier also allows the impedance of the antenna to be matched to the standard 75-ohms cable impedance.

The schematic diagram in **Figure 1** shows that such an amplifier need not be all that complicated. As can be seen, the loop antenna is connected to the inputs of a differential amplifier built using discrete components. It employs the well-known high-frequency transistor BF494 and its PNP equivalent, the BF451. The differential gain stage provides an amplification of around 10 and has a bandwidth of more than 30 MHz, which thus covers the entire short-wave band.

Transistor T3 acts as a buffer and impedance converter. The amplified signal passes to the output connector K1 via capacitor C3. A coaxial cable can be used to carry the signal from the output connector to the radio receiver.

CONSTRUCTION

Figure 2 shows the track layout and component layout of a printed circuit board design that is suitable for constructing the wideband loop antenna. This board is unfortunately not available through our Readers Services, so you will have to etch it yourself. After this is done, assembling the board should not take more than around half an hour, given the small number of components used.

Constructing the antenna itself is possibly even easier. Assuming that the dimensions of the window frame are 1 by 1.5 metres, you should wind two to four loops of insulated hook-up wire around small nails located at the corners of the frame. The core diameter of the wire is not particularly important. Be sure not to use a metallic window frame, since the antenna will not work at all with a metallic frame.

For the prototype, the window frame measured 82 by 133 cm and three turns were used. This gave outstanding results. The amplifier was fitted in a small box located at the bottom of the window frame. Since the circuit does not draw more than around 10 mA, a 9-V battery is fully adequate for the power supply. However, a stabilised (and well filtered!) mains adapter can naturally be used instead, if desired.

PERFORMANCE

With an antenna, a practical test says a lot more than a whole list of numbers. The active loop antenna was thus extensively tested in combination with the short-wave receiver described in the January 1999 issue of *Elektor Electronics*. In a one-to-one comparison, the active loop antenna proved to perform just as well as a long-wire antenna. Both types of antenna produced essentially the same S-meter readings over the full range of the receiver. By comparison, the telescopic antenna was significantly inferior; its S-meter readings remained at around one third of the level of the other two types.

(000005-1)

Text (Dutch original): S. van Rooij
Design editing: K. Walraven

COMPONENTS LIST

Resistors:
R1,R2 = 82kΩ
R3 = 560Ω
R4 = 220Ω

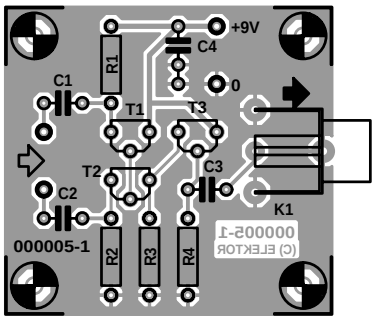
Capacitors:
C1,C2 = 2nF2 ceramic, raster 5mm
C3 = 100nF ceramic, raster 5mm
C4 = 100nF, raster 5mm or 7.5mm

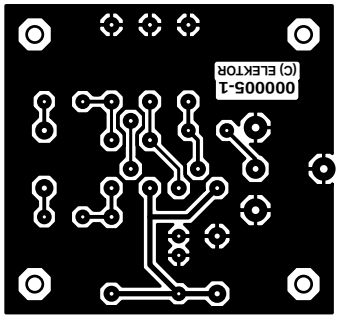
Semiconductors:
T1,T3 = BF494
T2 = BF451 (BF450)

Miscellaneous:
K1 = BNC or cinch socket
10 to 15 metres enamelled copper or hook-up wire

2

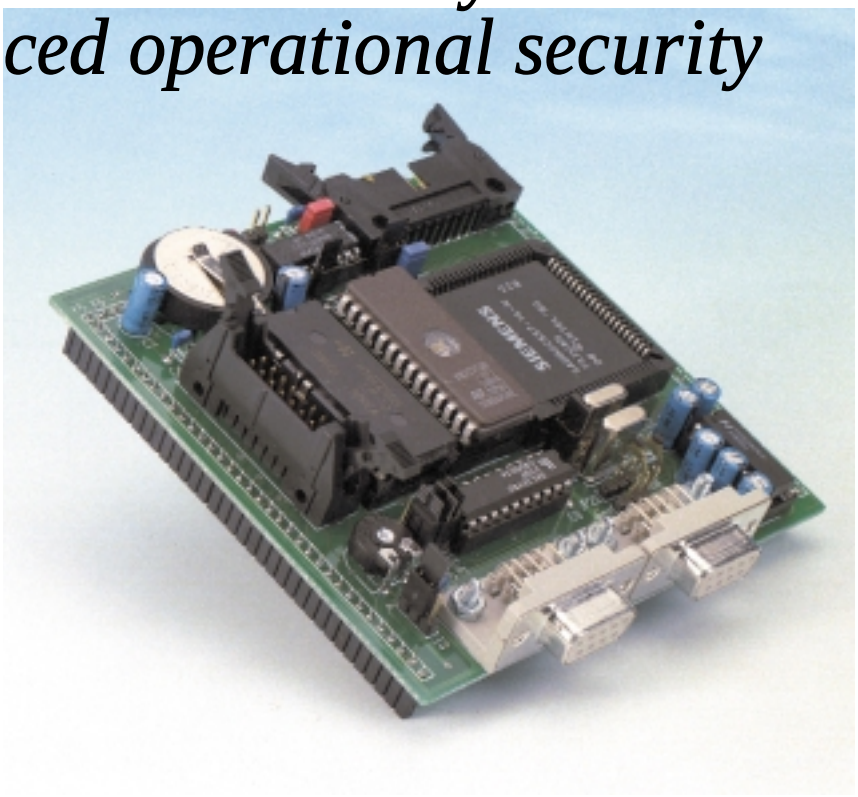
Figure 2. A simple design means that the printed circuit board can be delightfully compact.





537 'Lite' computer (1)

a high-performance, low-budget core microcontroller system with enhanced operational security



Siemens refers to its top-end 8051 derivative, the 80C537, as the '8-bit elephant'. In this article, *Elektor Electronics* presents a low-budget controller board using this microcontroller, as a companion to the previously-published design for a versatile teaching and training system based on the 537. The 'Lite' version is intended to be an easily-replaced core system that lends itself to a variety of applications in small to middle-sized projects, and which also satisfies special demands relating to operational security.

Design by Prof. B. vom Berg and Peter Groppe

Even a superficial glance at the data sheet of the 80C537 microcontroller shows that it truly has a lot of capabilities: an 8(10)-bit A/D converter with twelve inputs, three 16-bit timer/counters with compare/capture features, an arithmetic coprocessor unit, an on-chip watchdog timer, seven bi-directional digital I/O ports, two full-duplex serial interfaces, 14 interrupt sources, low-power modes and lots more. If you want to learn more about this interest-

537 'Lite' computer technical specifications

- *80C537 controller with selectable clock rate (12 or 16 MHz)*
- *enhanced operational security provided by a MAX807 microcontroller supervisory IC*
- *32 kB EPROM and 32 kB RAM, externally expandable to 64 kB each*
- *battery-backed RAM*
- *four free CS addresses (one reserved for LCD module if used)*
- *serial interface 1: RS232 levels; serial interface 2: selectable RS232 or TTL levels*
- *interface for direct connection to an alphanumeric LCD module, 1 x 16 to 4 x 20 characters, with contrast adjustment*
- *three different operating modes*
- *can be programmed in all 8051 languages (via Intel hex files) or using BASIC52 tokenised code*
- *extremely compact construction (90 x 100 mm)*
- *power: 5 V, 75 mA (12 MHz) or 80 mA (16 MHz), exclusive of LCD module*

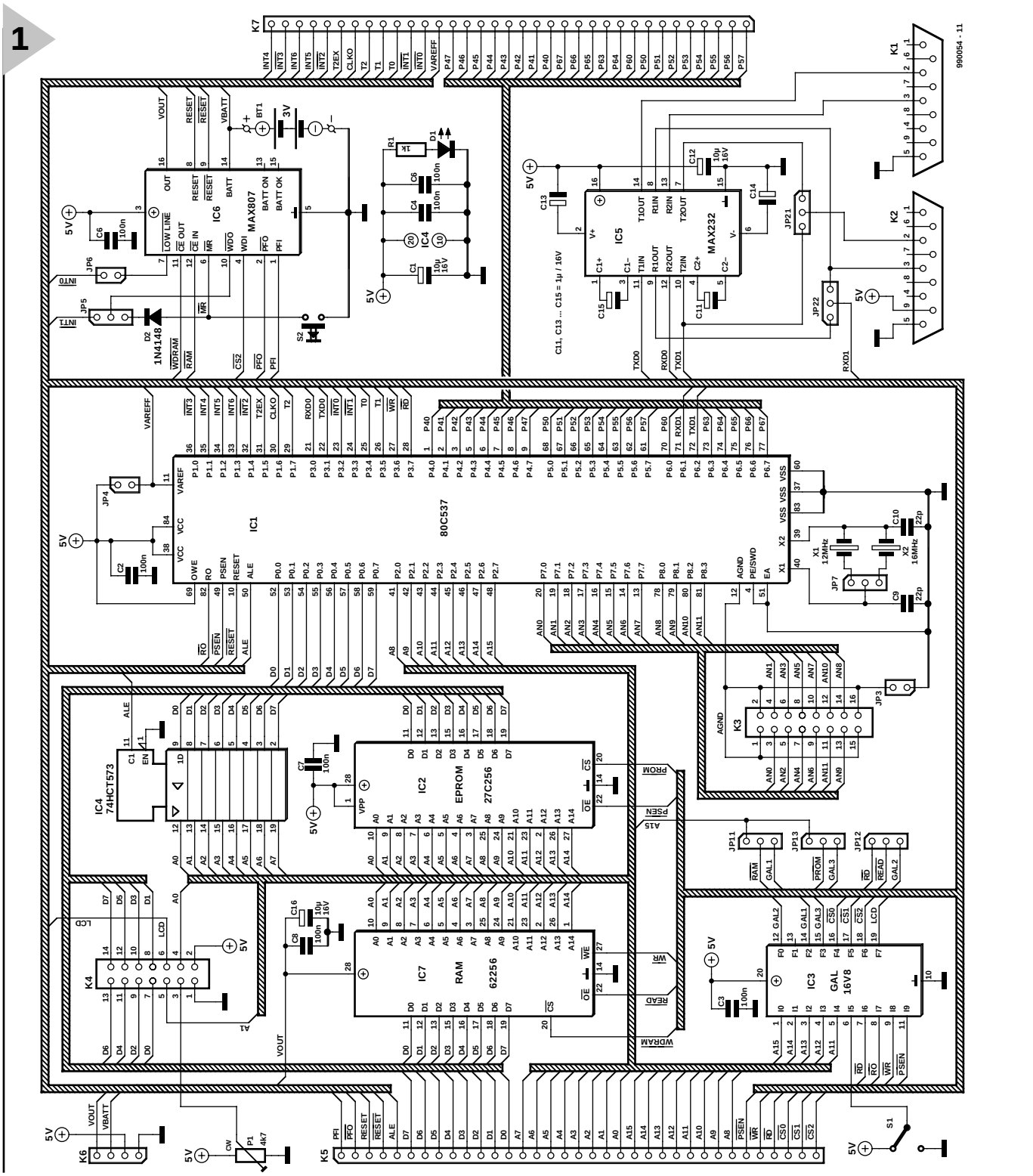


Figure 1. The SAB80C537 is an outstanding microcontroller that forms the heart of a high-performance board.

ing controller, refer to the literature listed at the end of the second instalment of this article.

537 'LITE' COMPUTER HARDWARE

The 537 'Lite' computer board, based on the SAB80C537 processor, represents an inexpensive, universally usable microcontroller system that can be plugged in to an application circuit just like a large IC. The main capabilities of the system are listed in the "537 'Lite' technical specifications" box. All important connections are made via headers or terminal blocks, so that they are available for extension as desired.

The hardware of the board, shown in the schematic diagram of **Figure 1**, can be divided into a number of functional groups.

Controller circuitry

The controller circuitry essentially corresponds to the basic circuit of an 8051-series microcontroller, with the minor difference that two different clock frequencies can be selected via jumper J7.

Digital I/O ports

The digital I/O ports are connected directly to header K7. The inputs do not have any protective circuitry, and the outputs are unbuffered.

Analogue input ports

The 80C537 has twelve multiplexed analogue inputs (0 to 5 V, 8 or 10 bit resolution). These are all connected to K3.

COMPONENTS LIST

Resistors:
R1 = 1kΩ
P1 = 4kΩ7

Capacitors:
C1,C12,C16 = 10μF 16V radial
C2-C8 = 100nF ceramic
C9,C10 = 22pF ceramic
C11,C13,C14,C15 = 1μF 16V radial

Semiconductors:
D1 = LED, 3 mm, red, high efficiency
D2 = 1N4148
IC1 = SAB 80C537-16-N
IC2 = EPROM 27C256 (order code **976510-1**)
IC3 = GAL 16V8 (order code **996515-1**)
IC4 = 74HCT573
IC5 = MAX232
IC6 = MAX807 NCPE
IC7 = 62256 70ns

Miscellaneous:
K1,K2 = 9-way sub-D socket, for PCB board edge mounting
K3 = 16-way boxheader with side latches, straight mounting
K4 = 14-way boxheader with side latches, straight mounting
K5,K7 = 35-way SIL pinheader
K6 = 4-way SIL pinheader
JP3,JP4,JP6 = 2-way pinheader + jumper
JP5,JP7,JP11,JP12,JP13,JP21,JP22 = 3-way pinheader + jumper
X1 = 12 MHz quartz crystal
X2 = 16 MHz quartz crystal
S1 = 3-way pinheader for connection to toggle switch
S2 = pushbutton, 1 make contact
BT1 = 3-V Lithium button cell, diameter 20 mm, with holder
PCB, order code **990054-1**
Disk, contains 537 Monitor program, commented, order code **976008-1**

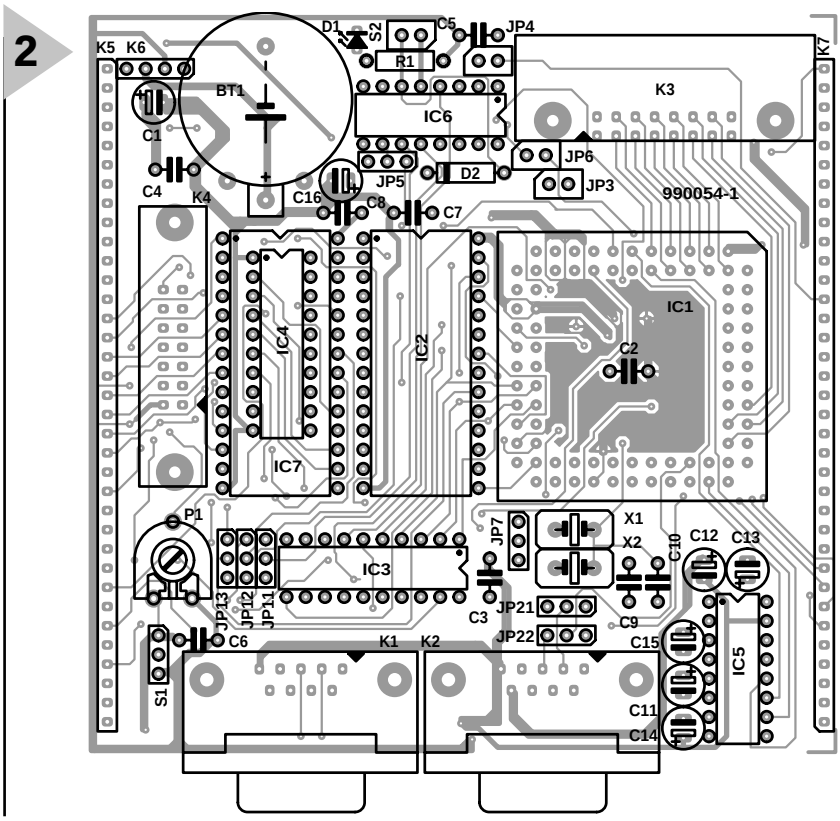


Figure 2. The relatively small printed circuit board can be installed in an application circuit just like a large IC.

Note that external protective circuitry and/or filters may be needed for these inputs. Jumpers J3 and J4 select the connections for the analogue earth (AGND) and the positive reference voltage.

Serial interfaces
The 80C537 has two serial interfaces (SS0 and SS1), which are connected to sub-D sockets K1 and K2. The SS0 interface always works as a V24 interface, using RS232 levels provided by the MAX232 level converter. The SS1 interface can be configured to operate with either V24 or TTL levels by means of jumpers J21 and J22.

Memory banks
The memory ICs (IC2, a 32 kB EPROM and IC7, a 32 kB RAM) are connected in the usual 8051-series manner via Port0 and Port2 of the controller, with the aid of a 74573 address latch (IC4).

Decoding logic
The necessary logic circuitry for encoding or decoding control signals is housed in a GAL 16V8 (IC3) in order to save space.

LCD module
An alphanumeric LCD module can be connected directly to connector K4, as long as the module has a Hitachi HD44780 controller or a compatible type (which is true of 80% of all such modules). This allows the 537 'Lite' board to be used directly with display modules ranging from one row of 16 characters to four rows of 20 characters. The display contrast can be adjusted using preset P1.

Microcontroller supervisory IC MAX807
The outstanding operational security of this microcontroller system is provided by the special-purpose MAX807 IC. This IC supports seven important and essential monitoring, reporting and alarm functions in the microcontroller system, as follows:

- automatic power-on reset generation;
- manual reset generation via a pushbutton switch;
- watchdog timer;
- chip-enable protection in case of system voltage failure;
- system voltage monitoring;
- power supply voltage monitoring;
- battery-backup connection with battery management.

A detailed description of this IC may be found in the datasheets to be published in next month's magazine.

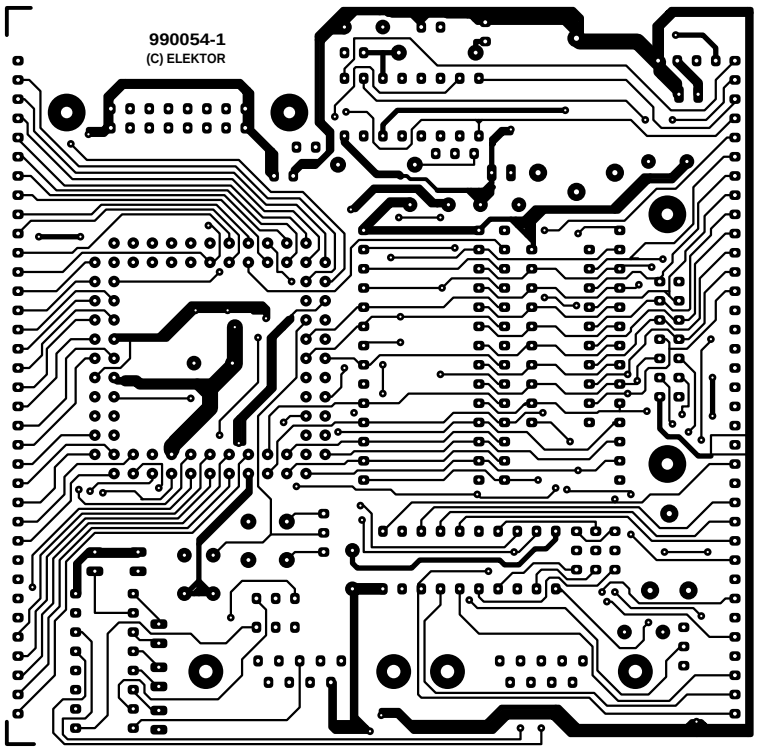
CONSTRUCTION AND INITIAL OPERATION — IT'S EASY!

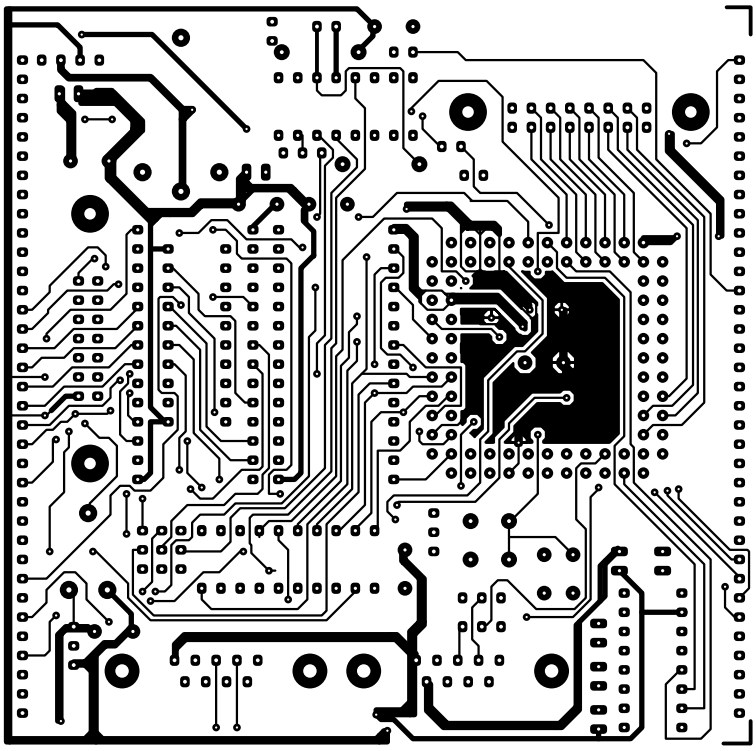
When constructing the circuit board, as shown in **Figure 2**, follow the well-known rules: fit the small components first and then the larger ones, use high-quality sockets for all ICs, and carefully inspect the board after completing the soldering. The backup battery should only installed at the very end, after initial operation has been successfully completed.

The memory ICs IC4 and IC7 are mounted 'piggy back' in order to save space. This mounting method is not as unusual as it may seem. Since sockets are used for both ICs, either a set of extra socket strips or an additional 28-pin socket with the middle portion cut away must be used to elevate the RAM IC (IC7).

After performing a final inspection of the components and installing the monitor EPROM IC2, set the basic configuration of the jumpers for operating mode 2, Program Download, as follows:

- set jumpers J11, J12 and J13 to position 2-3;
- set jumpers J21 and J22 to position 1/2;
- install jumpers J3 and J4;





- remove jumpers J5 and J6;
- set jumper J7 to position 1-2 (12 MHz clock);
- set switch S1 to the Load RAM position (grounded).

Now connect the supply voltage. *Tip:* during initial testing, do not connect the power supply leads directly to header K6 (that is, by soldering and subsequently unsoldering them), but instead supply the board with power via sub-D connector K2 (pin 5 = earth and pin 9 = +5 V). The operating voltage must be exactly 5 V, and it must not be connected with the wrong polarity! The current consumption should not exceed 80 mA.

After this, switch off the supply voltage and connect the SS0 interface on the board (K1) to a free COM interface of the PC. You can easily prepare the necessary cable yourself, as follows:

K1 (537-Board)	PC (AT)
Sub-D9	Sub-D9
TXD 2	RXD 2
RXD 3	TXD 3
GND 5	GND 5

If the board is to be connected to an old XT PC, the TXD and RXD connections must be swapped at the PC, and in addition the GND connection is on pin 7 of the 25-pin sub-D connector of the XT:

K1 (537-Board)	PC (XT)
Sub-D 9	Sub-D 25
TXD 2	RXD 3
RXD 3	TXD 2
GND 5	GND 7

Now you can again switch on the power for the 537 Lite board. The current consumption should not be any greater than before.

In order to establish communications with the board, go to the DOS level and insert Elektor diskette no. **976008-1** in drive A. First run the disk fingerprint check, `CHECK 1`. Now launch Windows and then HyperTerminal or another general-purpose communications program like ProComm or Telix. Use these settings: 9600, N, 8, 1, no flow control, and be sure to use a communication port that is still available for the 537 'Lite' Computer.

If you press the reset switch on the 537 'Lite' board, it will respond with the message `TFH 80C537er Monitor` ('TFH' stands for 'Technische Fachhochschule Georg Agricola in Bochum').

The link between the PC and the 537 board has now been established, and the first program may be downloaded. A reset, as just described, must be executed before every download of a new program. Once a program has been downloaded, press the reset button to return to the start of the program and then enter `g0` to start the program.

In order to download a program, press the F2 key (Input from file), enter `p1.hex` as the input file name, and confirm the entry by pressing <Return>. A number of columns of numbers will flit over the screen; these represent the Intel hex file of the `p1.hex` program. A hash sign will appear again to mark the end of the download.

After `g0` <Return> has been entered, the program starts at address 0. S1 should be in the Load RAM position. The microcontroller board will then greet you most cordially.

(990054-1)

Design editing: K. Walraven

Part 2 of this article, in the February issue, deals with programming the 537 'Lite' board. It also describes some possible extensions. Details on connecting the board to a CAN bus appear elsewhere in this issue.

Rhine Tower clock Mk II

a special design, with a circuit board shaped like the tower

The DCF-controlled LED clock design, published in the May 1998 issue of Elektor Electronics, is a radio-controlled electronic version of the clock mounted on the Rhine Tower in Düsseldorf. Inspired by the many positive reactions to this project, its designer has reworked the software to add new features. To make things even nicer, Elektor Electronics has designed a special circuit board in the shape of the actual tower in Düsseldorf.

Design by D. de Mülder

Building a circuit yourself is even more pleasant when the result is something unique. Those of you that have built the original DCF-controlled LED clock will certainly have experienced this first-hand. Anyone who sees such a clock for the first time will undoubtedly wonder what it is and how it works.

The clock that is integrated into Düsseldorf's Rhine Tower consists of a series of lights arranged in a vertical line. The 39 lamps that are used to indicate the time on the tower are replaced in this project by yellow LEDs. The outline of the Rhine Tower can be seen in the drawing of **Figure 1**. The nine LEDs at the very bottom display units of seconds (0 through 9). The four LEDs above them display tens of seconds. The minutes and hours are encoded in a similar manner. Finally, there are two

LEDs that display tens of hours. All of this can be clearly seen from the drawing. At 23:59:59 all lamps are illuminated, and at exactly midnight (00:00:00) all lamps are out. A new 24-hour cycle starts at this point.

In contrast to the previously published design, the beacon lights are also included in the present design. In addition to the four lamps that indicate the wind direction (N, E, S and W), there is also a beacon lamp at the top of the tower. The result is a faithful reproduction of the Rhine Tower (clock).

In addition to these cosmetic modifications, a number of new functions have been added in the software. It is thus no longer necessary to use a DCF receiver (although this simplifies operation and gives better results). The new clock can function without the radio time reference signal, although it will naturally not be as accurate over an extended period of time.

Since the DCF receiver is no longer essential, a number of pushbuttons have been added to allow the clock to be set.

An additional new function is the extraction of the day of the week from the DCF signal. It can also be manually entered using the pushbuttons. With this additional information, it is possible to use the clock as an alarm that takes the day of the week into account. Naturally, an electronic buzzer is also included to generate the wake-up signal.

In order to accommodate all these functions, it is of course necessary to use a somewhat bigger processor. This is the price that must be paid for the extra features.

CONTROLLING LAMPS WITH BITS AND BYTES

The schematic diagram of the LED clock is shown in **Figure 2**. A large part of this is the same as the schematic of the 1998 design. The same type of microcontroller is still used, as well as the same LED-driver IC and power supply. These parts of the circuit are all straightforward. The driver for the piezo buzzer is new. With the specified components, any buzzer that works with a 5 V supply voltage and a current not exceeding 50 mA can be used for Bz1. The three pushbutton switches are connected to processor inputs that were not used in the previous design. The extra LEDs fit into previously unused positions in the matrix.

As before, an additional LED that lights whenever the supply voltage is present can be connected via jumper JP1. Resistors R4 and R5 are provided to allow experiments with other displays to be carried out. Under normal circumstances, these resistors are not necessary, and in most cases they may be omitted (along with R7, which is connected to JP1).

It goes almost without saying that, while the hardware is almost the same as in the earlier design, the controller software has been extensively modified. In order to use the new functions, it is thus necessary to acquire a new controller. It is possible to use the new circuit board with the old controller, if you wish, but the additional LEDs will not function in that case.

CONSTRUCTION

The printed circuit board track layout and component layout are shown in **Figure 3**. As is immediately apparent, this is no ordinary circuit board. Not only does it have an unusual form, its size is also special, which is why the illustration is only 55% of the actual size. If you want to etch your own circuit board, you must first enlarge the layout drawing by 182%. Alternatively, you can find the full-scale layout drawing as a PDF document at Elektor's Internet site (http://www.elektor-electronics.co.uk). Of course, none of this matters if you buy the circuit board ready made. In addition, the ready-made circuit board is milled to the shape of the Rhine Tower by the board manufacturer, saving you the trouble of doing this yourself. If you make your own circuit board, a bit of exercise with a coping saw will be necessary to give the board its proper shape.

If you wish, you can colour the board by applying a thin coat of spray paint to the component side of the circuit, board before fitting the components (but take care to avoid clogging the holes!).

The foot of the tower consists of two circuit board sections that can be separated from the rest of the board if desired. The bottom-most section accommodatees the pushbuttons, while the middle section holds the processor that drives the whole circuit. It is a question of taste whether to leave everything in one piece or divide it into two or three separate boards. It's up to you to decide, and the choice will depend in part on how the finished unit will be set up.

Since the three portions of the printed circuit board are electrically separate, a number of interconnections must be made, regardless of the final configuration. You should first fit the wire jumpers and all necessary pins. It's a good idea to used sockets for IC1 and IC2. When fitting the LEDs, pay good attention to the polarization. With so many LEDs, it's easy to make a mistake.

After all the components have been fitted, the three portions of the circuit board must be connected to each other using a number of bits of wire and a piece of flatcable, even if they are not

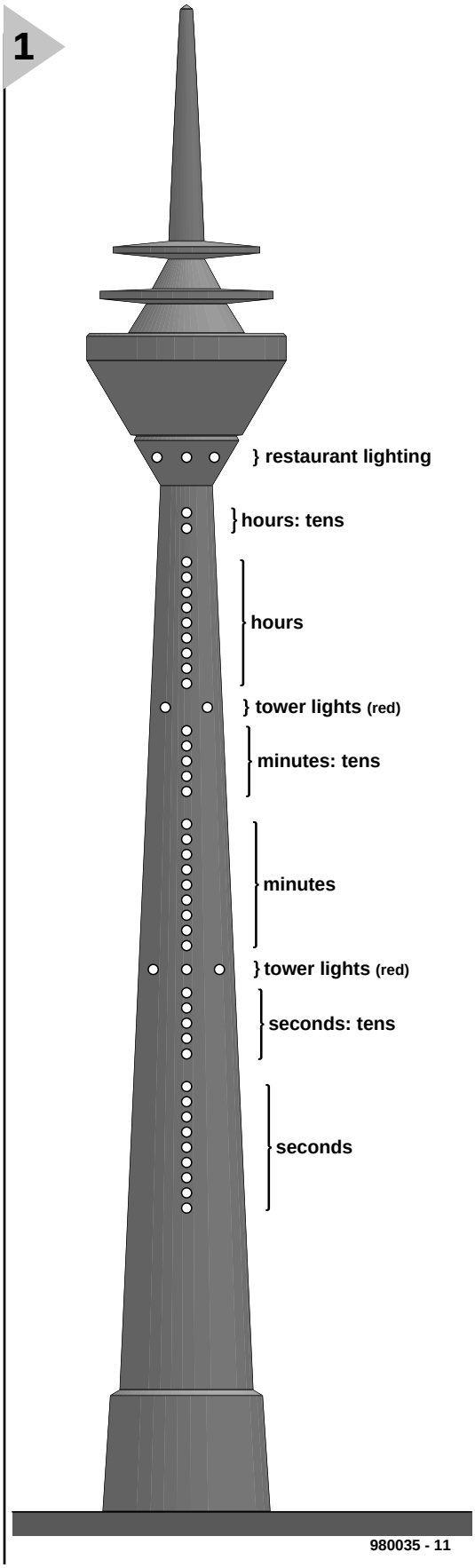


Figure 1. This drawing shows the configuration of the clock on the Rhine Tower in Düsseldorf, Germany.

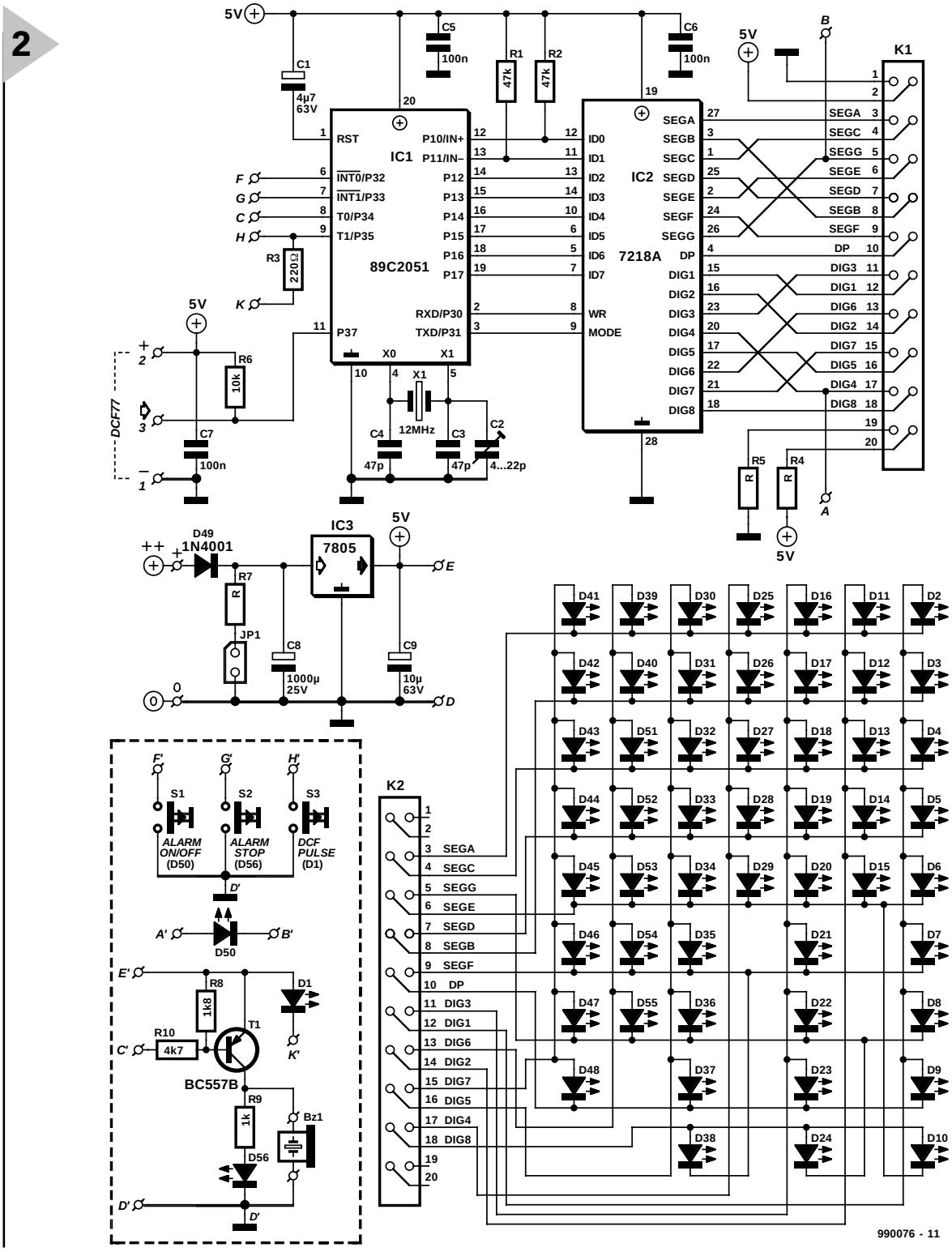


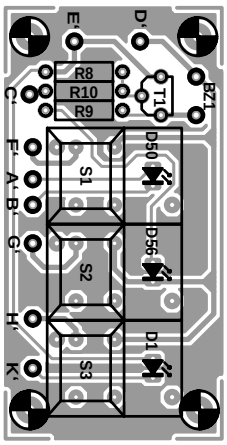
Figure 2. The schematic diagram of the clock. The similarity to the original DCF-controlled LED clock circuit is readily apparent.

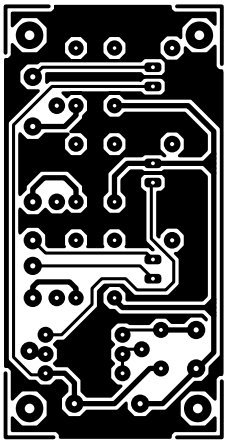
separated. Connect corresponding points to each other (A to A', B to B' and so on, finishing with K to K'). After this, connect a length of 20-lead flatcable between K1 and K2. The optional DCF receiver can be connected to PC1, PC2 and PC3. The CPU generates interference that can affect the operation of the DCF receiver, so the receiver should be located 20 to 30 cm away from the CPU. Use a piece of screened cable for the connection.

A wall adapter that can deliver around 250 mA at a d.c. voltage between 8 and 12 V can be used to supply power to the circuit.

Once all interconnections have been made and all components installed, the mains adapter can be plugged in. If a DCF module is used and the signal reception is adequate, LED D50 should flash once per second. This is also true for some of the beacon lamps. If signal reception is good (as indicated by D50 flashing), the exact time should be displayed after two to three minutes.

DCF77 reception should be possible within a radius of 1500 km (just under 1000 miles) from the transmitter location in Mainflingen, Germany. From previous projects employing the DCF77 signal we know that reception is just about adequate in South-Eastern parts of England, South Scandinavia and most of Central Europe. If you live in a fringe area, remember that the radio signal is not required all the time! Also remember that DCF77 transmits CET (Central





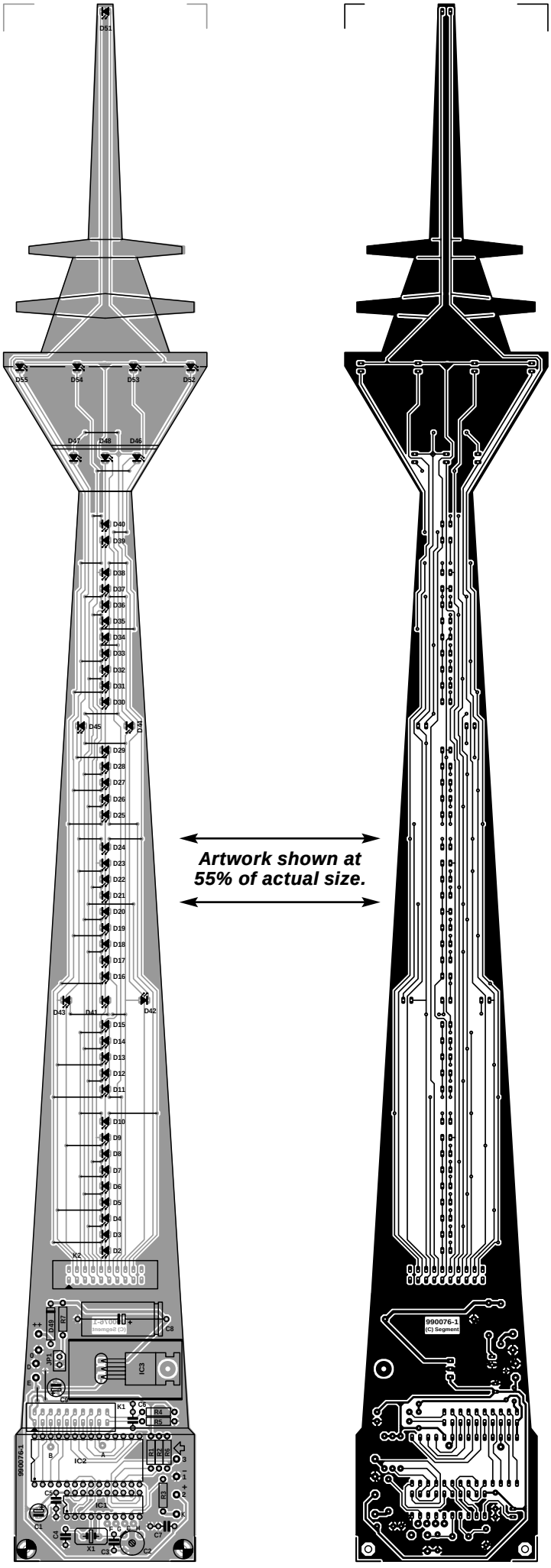


Figure 3. This may not be the largest circuit board ever offered by Elektor Readers' Services, but it is certainly one of the most attractive.

European Time).

None the less, if you do not use a DCF module, you will have to set the time manually and initialize the non-DCF operating mode. To do this, press S3 at the same time as power is applied. The restaurant lamps will come on, while all other lamps will remain off. Release S3 and start with setting the time. Use S1 to set the hour. The hours count is increased by one each time S1 is pressed. After this, use S2 to set the minutes. Note that there is an automatic 'rollover' from the minutes setting to the hours setting.

Use S3 to set the day of the week. It has a double function in this regard. Press it briefly (0.1 to 1 second) to set the day of the week; the count increases by one each time the button is pressed. While the day of the week is being set, the first seconds LED indicates Monday, the second LED Tuesday and so on. After Sunday (all seven LEDs on), the day automatically

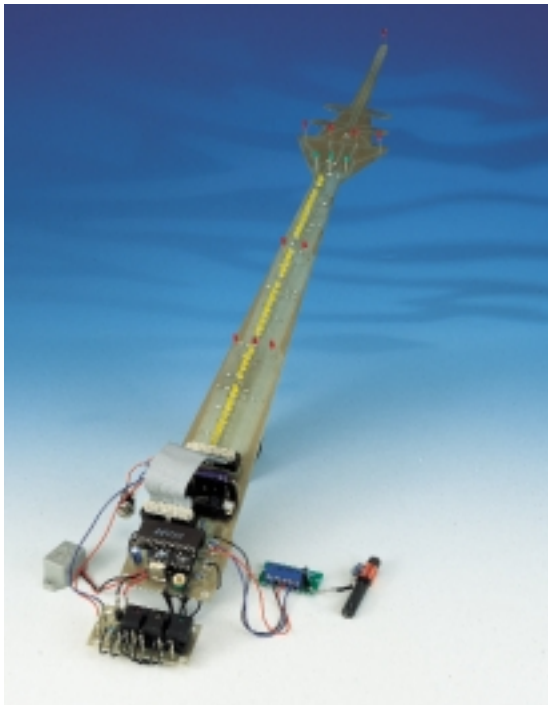


Figure 4. The assembled prototype of the clock.

returns to Monday (one LED on).

After everything has been set, you can start the clock using S3. If S3 is held pressed for longer than one second, the clock will start, and it starts exactly on the minute. You should therefore press S3 at the 59th second of the minute, so that the clock will start exactly when the minute changes.
After a bit of practice, you should find the operation of the clock very simple. Those of you that use a DCF module need not be concerned about all this, since clock synchronization is fully automatic with a DCF module!

SETTING THE ALARM

After the clock is set and is running, the alarm time can be set. The clock continues to run in the background while the alarm is being set. However, the DCF module is temporarily not used as long as the alarm is being set. This is not a problem, since the clock is crystal-controlled and thus runs quite accurately using its own internal timing. (Note that many battery-operated DCF clocks synchronize themselves with the DCF time only once per day, in order to save energy. This is usually sufficient to maintain the desired level of accuracy.)

To set the alarm time, first press S3. Either all LEDs will go out, or they will indicate the last-entered setting for the alarm time (including the day of the week).

All three pushbuttons have double functions for setting the alarm time, depending on how long they are pressed — either briefly (up to 0.6 second) or longer (more than 0.6 second). The alarm time is specified in hours and minutes. The functions of the switches are as follows:

S1 short: increment the hour
S2 short: increment the minute
S3 short: increment the weekday
S1 long: switch the alarm on or off
S2 long: program the weekday
S3 long: return to the time display

Each time S1 is pressed, the alarm time in increased by one hour. S2 sets the minute of the alarm time in a similar manner, and S3 the day to which the alarm time applies. The first time S3 is pressed, the LED belonging to Monday starts to flash. If the alarm should only be active on this day, press S2 until the flashing weekday lamp stays on continuously. To program more than one day in the week, use S3 to increment the weekday count and then S2 to select the day. If the weekday count rolls over from Sunday to Monday, all daily programming is erased and you will have to start over with selecting the day(s).

The last position of the cursor is saved when the clock is returned to the time display mode. This makes it possible to add other days to the program at some later time. Finally, press and hold S1 to activate the alarm function. The lamp built into S1 will illuminate. To terminate the alarm setup procedure and return to the normal time display, press and hold S3. When the alarm goes off, you can silence it by pressing S3.

The alarm output itself is a logic-level output that is connected to a switching transistor and a d.c. beeper. If desired, a sound-effects generator, a relay or some other type of circuit could be connected to this output. In some cases it may be necessary to connect a buffer to the output. There are lots of ways you can experiment with this unusual clock.

(990076-1)

Text (Dutch original): H. Steeman
Design editing: K. Walraven

COMPONENTS LIST

Resistors:
R1,R2 = 47kΩ
R3=220Ω
R4,R5,R7 = see text
R6 =10kΩ
R8 = 1kΩ8
R9 = 1kΩ
R10 = 4kΩ7

Capacitors:
C1 = 4μF7 63V radial
C2 = 4-22 pF trimmer
C3,C4 = 47pF
C5,C6,C7 = 100nF
C8 = 1000μF 25V
C9 = 10μF 25V radial

Semiconductors:
D1,D46,D47,D48 = LED, high eff., green
D2-D40,D56 = LED, high eff., yellow
D41-D45,D50-D55 = LED, high eff., red
D49 = 1N4001
T1 = BC557B
IC1 = AT89C2051-12PC (order code **996519-1**)
IC2 = ICM7218**A** IJI
IC3 = 7805

Miscellaneous:
JP1 = 2-way jumper
K1,K2 = 20-way boxheader
Two 20-way sockets and a piece of flatcable
S1,S2,S3 = pushbutton ,type 'Digitast' with integral LED (ITT Schadow)
X1 = 12 MHz quartz crystal
Bz1 = dc buzzer, 5 or 6 V
Heatsink for IC3 (15 K/W, e.g., ICK35)
PCB, order code **990076-1**
Optional: DCF-module (Conrad Electronics order code 64 11 38-55)

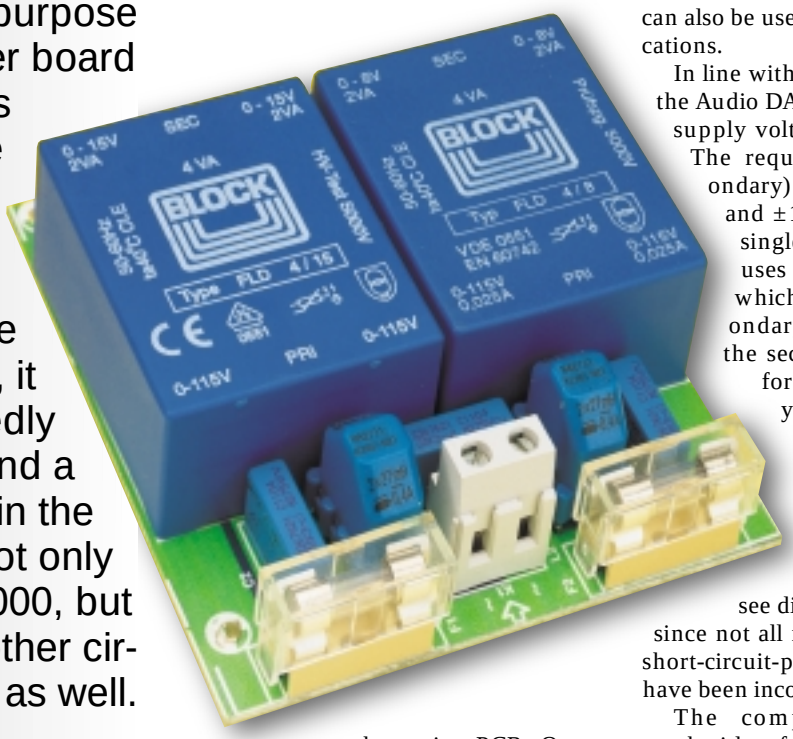
The Rhine Tower clock on the Internet

If you are interested in the Rhine Tower and its clock you may find a lot of interesting information on the Internet. For example, a Windows screensaver based on the Rhine Tower can be found at http://www.duesseldorf.de/tourist/download/index.html.
A site that is based on the theme of clocks can be found at http://www.hsp.de/~wiegels/programm/uhren.htm. Naturally, software that emulates the Rhine Tower clock is available at this site.
If you want to know more about the Rhine Tower itself, the site 'Höhe Türme' ('high towers') is certainly something for you. Go to http://www.hsp.de/~wiegels/tuerme/hoch.htm to see all the information that is available regarding this high tower (and others).

transformer board

for the Audio DAC 2000 and many other circuits

The general-purpose transformer board described in this article has space for two small mains transformers, two mains filters and two fuse holders. As such, it will undoubtedly prove a boon and a practical aid in the construction of not only the Audio DAC 2000, but in that of many other circuits as well.



Design by T. Giesberts

Many readers will have noted (and some have indeed commented) that over the past few years the projects published in this magazine seldom provide space on the associated printed-circuit board for a mains transformer. One of the reasons for this is safety. Working with mains voltages is and remains a risky undertaking and not-too-careful constructors, especially the younger ones, run the risk that the potentially lethal mains voltage appears at places where it should not. Much of this risk is obviated by the use of discrete mains adaptors, as has been the case in most recent mains-operated projects.

Where, for one reason or another, the use of a mains adaptor is precluded, modern practice is nonetheless to avoid placing the mains transformer on the mains PCB. One reason for this is that it ensures that the board does not become unduly large — and expensive — and another is that many constructors like to solve their mains power supply in their own way.

Another important reason for not placing the mains transformer on the mother board is that it is frequently desirable to keep the transformer as far away from the signal circuits as possible. This prevents problems with hum and other interference.

TWO TRANSFORMERS

Although the reasoning in the introduction is sound and practical, it is an unfortunate fact that many suitable mains transformers, particularly the smaller ones, are available only with terminals for soldering on to a board — they have no other fixing facilities.

It is for this reason that the present transformer board has been designed. Although this is intended primarily for use with the Audio DAC 2000, it can also be used for many other applications.

In line with many modern circuits, the Audio DAC 2000 works from two supply voltages: +5 V and ±12 V. The requisite transformer (secondary) voltages are 8 V (or 9 V) and ±15 V respectively. For the single supply, the prototype uses a 2×8 V transformer of which only one of the secondary windings is used — the second remains available for any other purposes as yet unforeseen.

The transformers are flanked on the board by mains filters L1-C2 and L2-C3 and an additional decoupling capacitor, C1 – see diagram in **Figure 1.** Also, since not all mains transformers are short-circuit-proof, two fuse holders have been incorporated.

The component layout and track side of the board are shown in **Figure 2**.

So as to enable the finished board to be fitted in a compact enclosure, low-profile transformers are used in the prototype. These are available in a 4 VA or 6 VA format from manufacturers such as Hahn, Block, and Monacor/Monarch. Hahn also has 3 VA, 10 VA, and 16 VA models in their catalogue. Although these models differ in height, their width and depth are the same. The board is suitable for use with all these. The components list gives model numbers for transformers suitable for use with the Audio DAC 2000.

The specified ratings for the fuses (32 mA, delayed action) apply to the non-short-circuit-proof 4 VA transformers from Hahn and Monacor/Monarch. When other transformers are used, these ratings should be altered accordingly.

The enclosure should be chosen with a view to the appropriate safety regulations. For instance, since the

Figure 1. The board has space for not only two mains transformers, but also two mains filters, and two fuse holders.

mains-carrying fuses are located at the edge of the board, this should be at least 6 mm away from the metal enclosure.

Also, the regulations require the minimum distance between mains-carrying tracks and ground to be at least 10 mm. This means that the board must be mounted on 10 mm long nylon or polythene spacers.

Finally, if the board is used in conjunction with the Audio DAC 2000 and a 2×9 V (instead of a 2×8 V) transformer is used, the dissipation of IC17 in the DAC rises sufficiently to make it desirable for the device to be mounted on a small heat sink.

[000001]

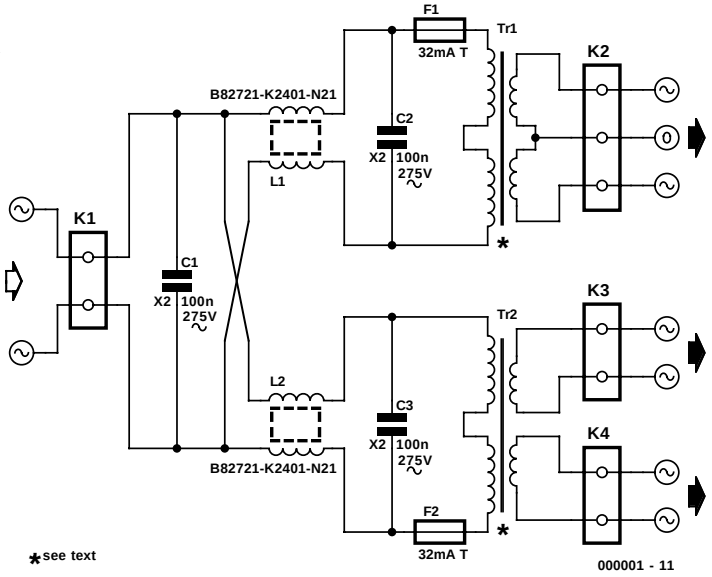
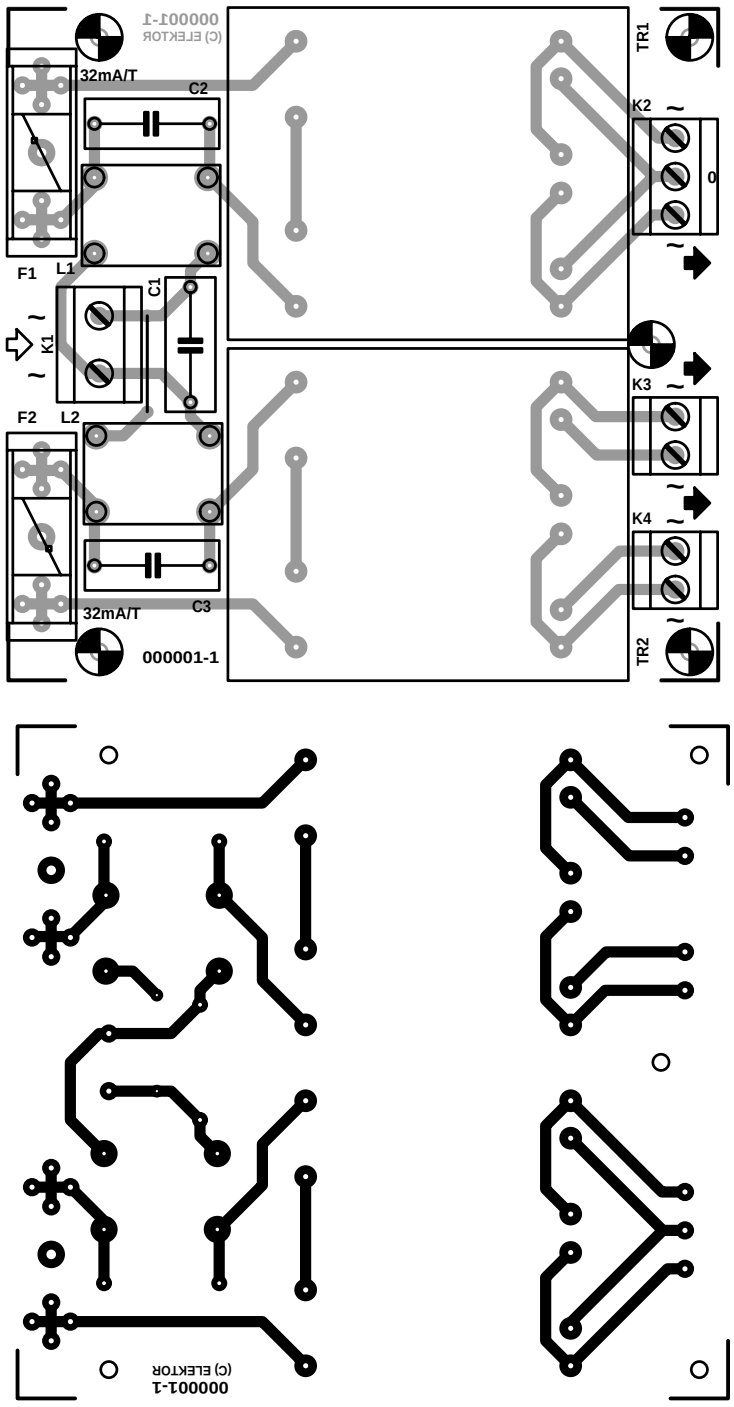


Figure 2. The board layout is designed primarily for use with low-profile transformers, although taller ones can also be used.



COMPONENTS LIST

Capacitors:
C1,C2,C3 = 100nF 275V_{AC} class X2, raster 15mm

Inductors:
L1,L2 = 2 x 27mH/0.4 A e.g., Siemens type B82721-K2401-N21

Miscellaneous:
K1 = 2-way PCB terminal block, raster 7.5mm
K2 = 3- way PCB terminal block, raster 5mm
K3,K4 = 2- way PCB terminal block, raster 5mm
F1,F2 = fuse 32 mAT (slow), with PCB mount holder
Tr1 = PCB mount mains transformer 2 x 15 V/4 VA, e.g., Block FLD4/15; Hahn BV UI 302 0165; Monacor FTR-415. *
Tr2 = PCB mount mains transformer 2 x 8 (or 9)V/4 VA, e.g., Block FLD4/8 or FLD4/9; Hahn BV UI 302 0161; Monacor FTR-49.
PCB, order code **000001-1**(see Readers Services pages)

*) Hahn and Monacor series not short-circuit resistant. Block series short-circuit resistant.

BASIC Stamp programming course (5)

Part 5: Remote Rover

This month we explore how the BoE-Bot vehicle can be made to respond to commands received from an infra-red remote control.

20

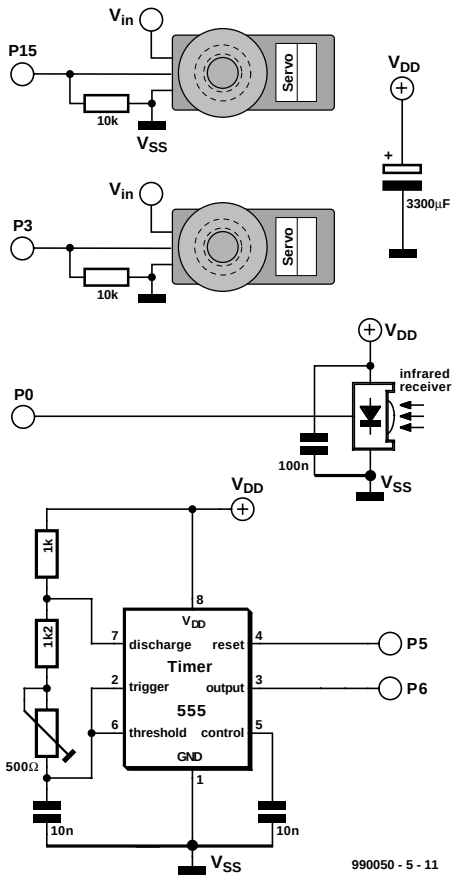


Figure 20. The Remote Rover schematic. Only one IR receiver is used (Panasonic type 4602).

By Al Williams

How many remote controls do you own? If you are like me, the number is embarrassing. Every piece of consumer electronics seems to have its own remote these days. When you lose or break an original remote, you have to keep buying new remotes — none of which completely control the device in question.

Last month, I showed you how to add infra-red (IR) object detection to your BoE-Bot. IR components are plentiful because remote controls generally use IR to send commands back to their parent device. So why not make the IR detector on your robot pick up commands from a remote control? If you could communicate with the robot remotely, you could command it to do your bidding without leaving your chair! Along the way you'll find out how the Basic Stamp measures pulses and handles arrays.

DETAILS

The robot circuitry is simple (see **Figure 20**). This is just about the same circuit you used last month, except there is only one IR sensor and no LEDs. The LEDs are in the remote control unit. With some experimentation, you could probably get any remote control to work. I used a Sony unit because the SIRCS (Serial Infra Red Control Signal; sometimes called Control S) protocol is well documented on the Internet. If

you don't have a Sony remote, get a cheap universal remote and program it to operate a Sony TV.

There are several common protocols that remotes use (see *http://www.hut.fi/Misc/Electronics/docs/ir/ircodes.html* for some ideas). Most use some form of pulse width modulation. For the Sony protocol, the remote sends a start bit (often called an AGC pulse) that is relatively wide (more than 2 ms). This allows the receiver to synchronize and adjust its automatic gain control (this occurs inside the IR detector module).

After the start bit, the remote sends a series of pulses. A 600 μs pulse is a 0 and 1200 μs pulse is a 1. There is a 600 μs gap between each pulse. Of course a variety of factors will affect the timing, so you should consider these times approximate.

PULSE MEASUREMENTS

The Stamp is adept at reading pulse widths (using the **PULSIN** command). **PULSIN** requires three arguments. The first specifies which pin you want to use to measure the pulse. The Basic Stamp will set this pin to be an input if it isn't already. The next argument tells the command if you are looking for a low to high transition (1) or a high to low transition (0). The final argument is a word-sized variable that holds the time duration of the pulse (if any). The Basic Stamp uses a 2 μs time base, so if the variable contains, for example, 100, the pulse was 200 μs wide. The command times out in 131 ms. If no pulse occurs during that time, the command sets the variable to 0. The largest possible pulse to measure is 65535 × 2 μs = 131 ms.

The **PULSIN** command only measures pulses if it finds the specified edge. Suppose you want to determine how long the user pushes a button. The button provides a zero on the input pin when pushed. If you start executing **PULSIN** after the button is providing a zero, you'll never measure the pulse. You must execute **PULSIN** *before* the edge of the pulse occurs. If you think about it, this makes sense because it prevents **PULSIN** from returning inaccurate results. The command always measures a full pulse.

You can use a byte variable if you are measuring pulses that will never exceed 510 μs. However, if a pulse does exceed this width you will get an erroneous result with no warning. Also, the command always uses a 16-bit timer internally, so using a byte variable does not alter the time-out period. With the IR sensor, you'll need to read pulses over 2 ms wide, so all the programs in this article will use a word-size variable.

Assuming the IR sensor is on pin 0, it is very easy to measure the width of an IR pulse:

```
IRREAD:
  PULSIN 0,0,W1
  IF W1=0 THEN IRREAD ' no pulse
  DEBUG ?W1
  GOTO IRREAD
```

You can try this simple program to verify that you get different widths as you press keys on the remote. Of course, you are not reading each bit, so the results are not meaningful yet. The Sony remote emits 12 data bits (13 if you count the start bit). The first seven bits comprise the function code (least-significant bit first) and the next five bits make up a device ID. Since you only want to know the function keys, you can ignore the device ID. Conceptually, you could read the word like this:

```
IRREAD:
  B0=0          ' byte read in
  B1=1          ' bit mask
  PULSIN 0,0,W5
  IF W5<1200 then IRREAD
                ' not a start bit
  FOR B2 = 1 to 7
    PULSIN 0,0,W5
    IF W5<400 THEN READZERO
    B0=B0+B1    ' set 1 bit
READZERO:
    B1=B1*2     ' bump mask up
  NEXT
```

This looks like a good piece of code, but it does not work. The principle, however, is sound. The first **PULSIN** reads the start bit and rejects any bit that isn't the right length. Then the code enters a loop, reading each bit in turn and setting the corresponding bit in **B0** when the length of the pulse is greater than 800 μs (this is much greater than a 0 at 600 μs; since a 1 is nominally 1200 μs, none should be as short as 800 μs).

The only problem with this code is that there is only a 600 μs gap between bits. This is not much time for the Stamp to recover from reading the last pulse. The Basic Stamp takes at least 470 μs to execute an IF statement (the average time to execute a command is about 330 μs — some commands take longer, some take less time). Even the quickest commands require more than 100 μs. With the commands between successive calls to **PULSIN** the BASIC Stamp misses some bits.

One option, of course, might be to switch to a Basic Stamp IISX, which is much faster than an ordinary Basic Stamp. However, with some clever programming, you can make the ordinary Stamp II read the IR pulses at this rate.

THE SOLUTION

To solve this problem, you must reduce the instructions between the **PULSIN** commands. In fact, to ensure proper operation, you must *eliminate* the instructions between the **PULSIN** commands. How is this possible? Simply store the raw **PULSIN** results and process them later when more processing time is available.

The problem is where do you store the raw times? Sure you could use variables, but you'd need at least eight word variables (one for the start bit and seven data bits). This could lead to some very ugly code. If you are familiar with other programming languages, you might be thinking of storing the counts in an array. That is a good idea and luckily, the BASIC Stamp supports arrays.

ARRAYS

An array is a way of grouping similar variables together using the same variable name, with an *index number* to differentiate them. For example, suppose you wanted to work with the odd numbers. You might write:

```
oddnums var byte(5)
oddnums(0) = 1
oddnums(1) = 3
oddnums(2) = 5
oddnums(3) = 7
oddnums(4) = 9
```

Now `oddnums(2)` refers to the third odd number (remember, the index starts at 0). If you reserve five elements, then you'll use 0 to 4 for the index. If you use a different number, you will write over memory that some other part of your program is using.

This is useful because you can work with arrays inside loops. To print all the odd numbers, for example, you might write:

```
I var byte
for I = 0 to 4
  Debug ?oddnums(I)
next
```

Of course, you still have to observe the Basic Stamp's limit on memory. Arrays don't give you any extra memory, they simply help you better use the memory you already have.

READING IR

Successfully reading the IR data stream requires a series of 13 PULSIN statements (or possibly 8 if you want to ignore the extra bits). You only need to store 8 of these counts. So your code could read:

```
irsense con 0
irstartlow con 1100
    ' minimum start bit width
irstarthi con 1300
    ' maximum start bit width
raw var word(7)
dummy var word
start var word
```

Listing 8. The Remote Rover Software

```
' Remote Rover by Al Williams
irsense con 0
irinput var in0
irthreshold con 450
irstartlow con 1100
irstarthi con 1300

value var byte    ' result

raw var word(7)
start var word
dummy var word


right_servo con 3 ' right servo motor
left_servo con 15 ' left servo motor
delay var byte      ' motor cycle time
center con 750
speed var word
i var byte


delay=10
speed=100

top:
  gosub read_ir
  if value=1 then forward
  if value=3 then left
  if value=5 then right
  if value=7 then back
  goto top


forward:

  for i=1 to delay*2
    pulsout left_servo,center-speed
    pulsout right_servo,center+speed
    pause 20
  next
  goto top

back:
 for i=1 to delay
    pulsout left_servo,center+speed
    pulsout right_servo,center-speed
    pause 20
  next
  goto top

left:
  for i=1 to delay
    pulsout left_servo,center-speed
    pulsout right_servo,center-speed
    pause 20
  next
  goto top

right:
 for i=1 to delay
    pulsout left_servo,center+speed
    pulsout right_servo,center+speed
    pause 20
  next
  goto top



read_ir:
' The problem here is that the gap between bits
' is about 500µS and the Stamp may miss bits
unless
' you read everything in one swoop. So you
can't
' read this in a loop or even test the start
bit
' until you are finished.
  if irinput=0 then noir
' Already in the middle of a pulse so skip it
  pulsin irsense,0,start
  pulsin irsense,0,raw(0)
  pulsin irsense,0,raw(1)
  pulsin irsense,0,raw(2)
  pulsin irsense,0,raw(3)
  pulsin irsense,0,raw(4)
  pulsin irsense,0,raw(5)
  pulsin irsense,0,raw(6)
' Could comment these out
  pulsin irsense,0,dummy
  pulsin irsense,0,dummy
  pulsin irsense,0,dummy
  pulsin irsense,0,dummy
  pulsin irsense,0,dummy
' test for good start bit
  if (start<irstartlow) or (start>irstarthi)
then noir
  value=0
  for dummy=6 to 0
    value=value*2
    if raw(dummy)<irthreshold then ir0
    value=value+1
ir0:
  next

  return
noir:
  value=-1
  return
```

```
read_ir:
  pulsin irsense,0,start
     ' read potential start bit
  pulsin irsense,0,raw(0)
  pulsin irsense,0,raw(1)
  pulsin irsense,0,raw(2)
  pulsin irsense,0,raw(3)
  pulsin irsense,0,raw(4)
  pulsin irsense,0,raw(5)
  pulsin irsense,0,raw(6)
  pulsin irsense,0,dummy
           ' device ID – ignore
  pulsin irsense,0,dummy
           ' device ID – ignore
  pulsin irsense,0,dummy
           ' device ID – ignore
  pulsin irsense,0,dummy
           ' device ID – ignore
  pulsin irsense,0,dummy
           ' device ID – ignore
```

Now you need only process the raw data. It is possible that the first bit the code read was not the start bit, so a simple test prevents you from reading a packet in the middle:

```
   if start<irstartlow or
start>irstarthi then noir
```

Unfortunately, you can't perform this test right after the start bit is read — you must read the entire packet and then decide if it was correct or not.

The next task is to convert the raw data into a binary number. Here, speed is not as critical:

```
  value var byte
  value=0
  for dummy=6 to 0
    value=value * 2
    if raw(dummy)<irthreshold
then ir0
    value = value +1
ir0:
  next
  return
```

This simply examines each value (in reverse order). If it the raw value is

greater than the threshold, the code adds 1 to the result. In any case, the code multiplies the value by 2 (a shift left) each time through the loop. (The Stamp has a shift left operator, so you could replace the multiply statement with:

```
value = value << 1
```

Now you can read the keys from the remote easily. The keys do repeat so you'll want to take that into account in your program.

Internet

http://www.parallaxinc.com – BASIC Stamp Manual Version 1.9, BASIC Stamp DOS and Windows Editor, program examples. International distribution sources.
*http://www.stampsinclass.com —BoE documentation, Robotics curriculum, BoE-Bot *.dxf and *.dwg drawing formats, discussion group for educational uses of BASIC Stamp.*
chucks@turbonet.com — creator of the BoE-Bot and author of this series. Technical assistance.
kgracey@parallaxinc.com — co-author of this article. Technical assistance and questions about the educational program.
http://www.milinst.demon.co.uk — UK distributor of Parallax BASIC Stamp.

Remote Rover

Armed with these IR sensor routines, it is easy to add remote control to your robot. You just need to know what values the keys on the remote return. This is easy enough to deduce by calling the ir_read routine and using **debug** to print out the **value** variable.

For the Sony's remote, The "1" key returned 0, the "2" key returned 1, and so on. The "0" key returned 10. I decided to make the "2" key move forward, the "8" key move backwards, and the "4" and "6" keys move left and right. With the layout of the numbers on the controller, this is a natural arrangement. It is a simple matter to test for any particular key and send the correct commands to the servos (just like the other motion commands from previous months). You can find the complete Remote Rover code in **Listing 8**.

I did run into one limitation, however. After playing with the Remote Rover code for a while, I thought about making the robot move forward when you pressed the "2" key and then continue to go until you pressed another command or the "5" key. This turned out to be a problem.

It is easy enough to set a flag to indicate forward motion. However, the problem is that when you try to read the IR signal from the remote, you have to wait for each **PULSIN** command to time out before control returns to the main loop. With 13 commands each timing out in 131 ms, this works out to almost 2 seconds of dead time between motion commands. That makes the robot's motion jerky. This doesn't occur as badly with the original scheme because the repeating codes from the remote end the **PULSIN** commands without requiring them to time out.

Of course, you can mitigate this somewhat by not reading the device ID codes. This reduces the number of time-outs, but it increases the number of times your robot will miss the start bit and have to resynchronize with the remote. In the end, I decided to leave the code as it was.

Another way to improve efficiency is to test the sensor before sensing the start bit. If the sensor is reading a 0, then there is a packet in progress and there is no way you could possibly read it so why bother? The code in Listing 8 implements this test.

Future Directions

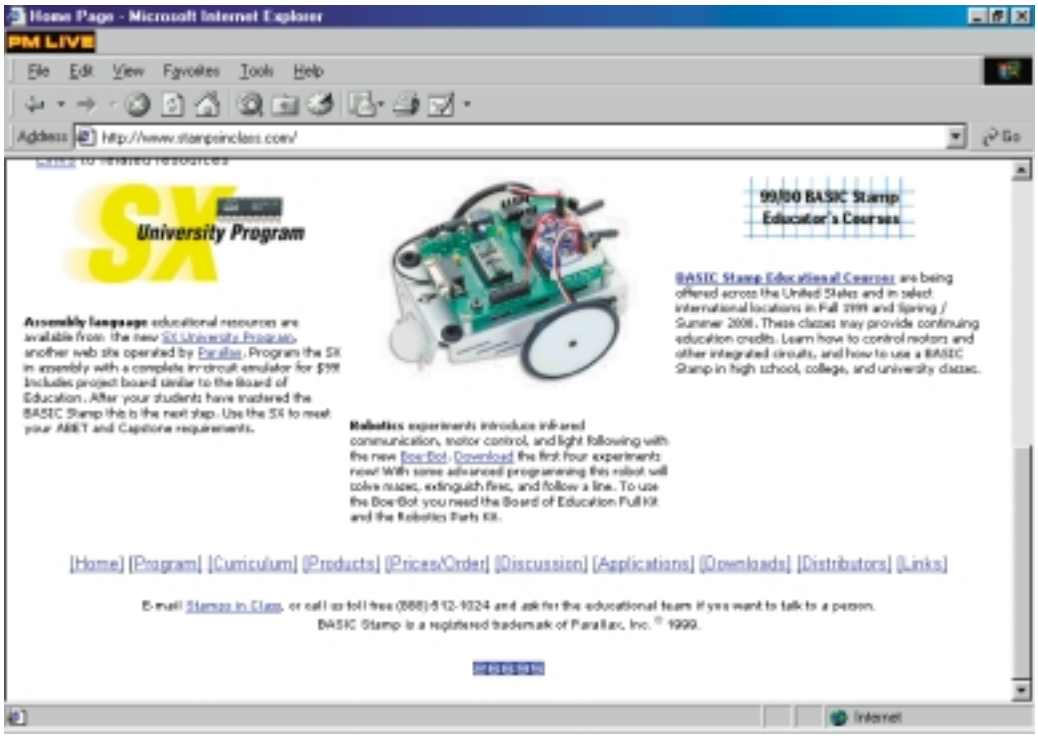
There are many simple modifications you could make to the program in Listing 8. For instance, it would be easy to program the volume and channel buttons to alter the **speed** and **delay** variables. Try it.

Once you can read remote codes, you can add many advanced features to your robot. For example, it would be trivial to make particular keys perform a predefined set of steps. With a bit more effort you could use the remote to program a sequence, store it in EEPROM, and then recall it later (like a macro, if you will). Once you understand the remote control's protocol, you could send the robot a series of commands for your television, dispatch it to another room and then have it repeat the commands when it is in sight of the television. You could even use these techniques to allow two robots to communicate with each other over modest distances.

Although an infrared remote uses very fast pulses, the BASIC Stamp can handle it if you write your software correctly. The **PULSIN** command made measuring the pulses simple and accurate. While not strictly necessary, arrays made the task much easier. With a little ingenuity, there is practically no limit to what the BASIC Stamp can do.

(990050-5)

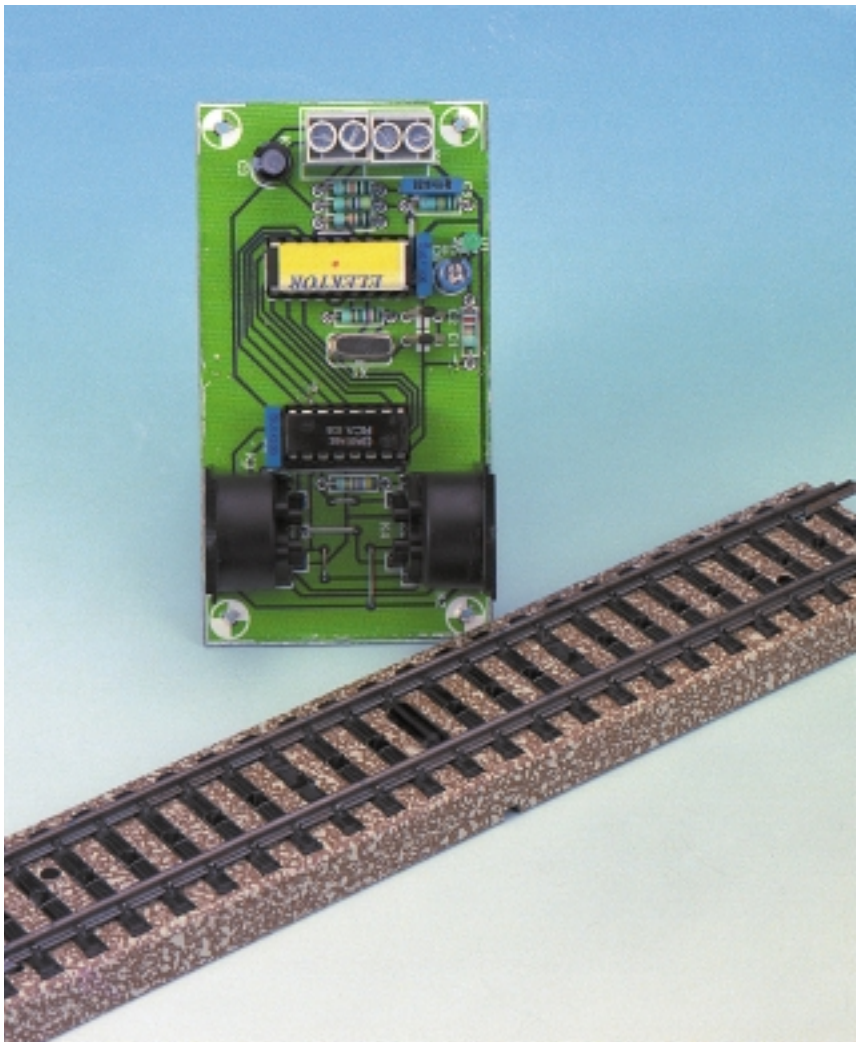
Article editing: J. Buiting
Design editing: L. Lemmens



PC-controlled model railway

EEDTS-Pro loco address return signaller

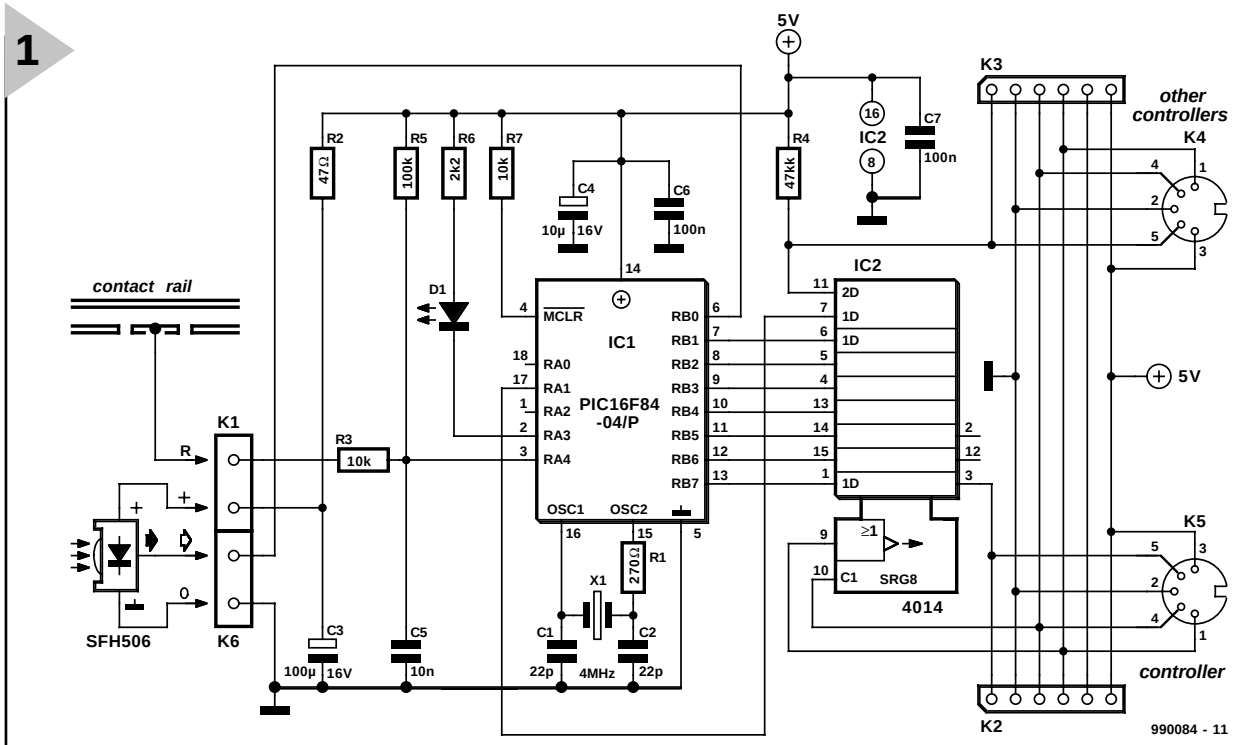
With the circuit described in this article, locos that pass the return signaller can be identified. Depending on the loco number, the software can then choose a predefined track or an alternative track, or wait until the selected track is free. Just as with the full-sized version, we can now (for example) have a local train enter the station and wait at a platform for a given time to allow passengers to board, while a freight train rumbles along the freight track and a TGV speeds along a passing track.



Specifications of the EEDTS Pro return signaller

- *IR reception range up to several metres*
- *can be used together with Märklin S88 and 'old' EEDTS return signallers*
- *occupies one half of an S88 address space*
- *includes one contact rail*

Design by H. J. Prince



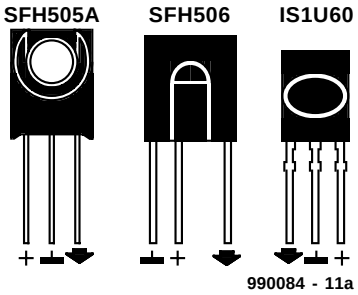


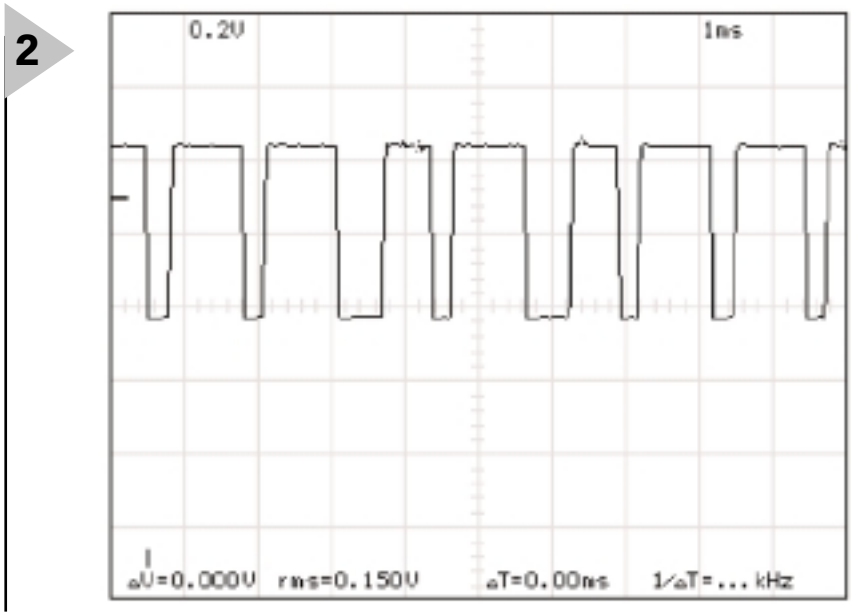
Figure 1. Schematic diagram of the loco address return signaller.

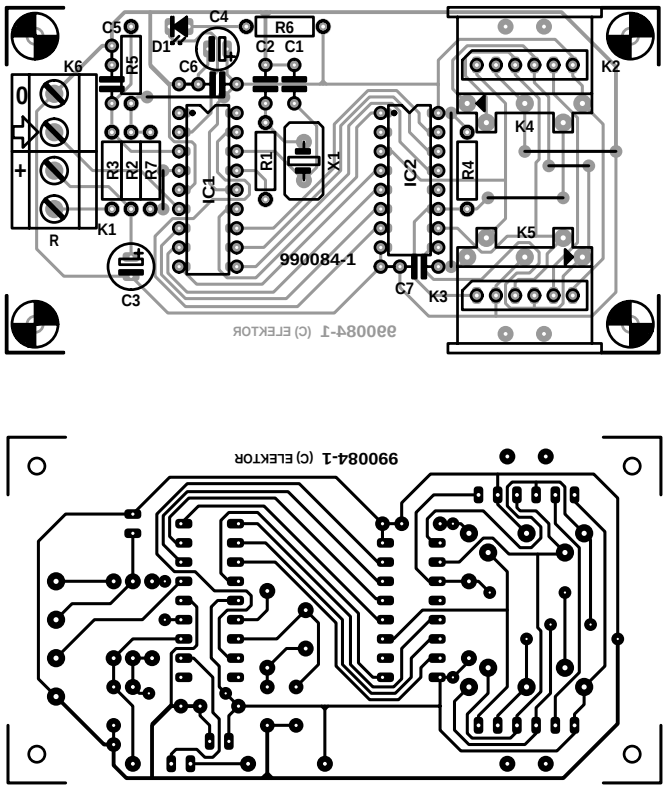
If you're operating a good-sized model train system, it's essential to know where every train is at any given time. Only then is it possible to properly carry out various activities, such as letting a train run along a specific track or stop at a platform. The new return signaller has been developed for this purpose. It utilizes an IR receiver that can be built into the railbed so that it is nearly invisible. Each loco transmits a unique number from its loco decoder, and this can be detected by the return signaller.

THE CIRCUIT

Just as with the loco decoder described last month, this circuit is built around a PIC 16F84 IC. At the left side of the schematic in **Figure 1**, we see the IR receiver (type SFH506). This module contains an IR receptor diode, an amplifier, a dynamic compressor and an HF demodulator. As soon as an IR signal is received from a loco that passes by the IR receiver, the signal is demodulated and appears in inverted form at the receiver output as an 8-bit pulse-code modulated signal containing the loco address. **Figure 2** shows a representation of this sig-

Figure 2. The signal at the output of the IR receptor.





COMPONENTS LIST

Resistors:
R1 = 270Ω
R2 = 47Ω
R3,R7 = 10kΩ
R4 = 47kΩ
R5 = 100kΩ
R6 = 2kΩ2

Capacitors:
C1,C2 = 22pF
C3 = 100µF 16V radial
C4 = 10µF 16V radial
C5 = 10nF
C6,C7 = 100nF

Semiconductors:
D1 = low-current-LED, red, 3 mm
IC1 = PIC16F84-04/P (programmed, order code **996523-1**)
IC2 = 4014
SFH 506

Miscellaneous:
K1,K6 = 2-way PCB terminal block, raster 5mm
K2,K3 = 6-way Märklin header
K4,K5 = DIN socket (female), 180 degrees, PCB mount
PCB, order code **990084-1**

Figure 3. The printed circuit board and component layouts for the return signaller (board available ready-made).

nal, in this case the binary form of the address '24'.

This signal goes to the interrupt input of the PIC, where it is temporarily stored and compared with the next 8-bit data word that is received. If the two words match, a valid reception is signalled by switching on LED D1. The moment the contact rail is closed (pin 3 of the PIC is pulled low), the decoded address appears at the PIC output. The 4014 IC converts the address back into a serial data stream that can be recognized by the EEDTS Pro controller. The loco address return signaller can be connected in series with standard signalling units.

CONSTRUCTION

Putting together the circuit board, as shown in Figure 3, should not present any problems. With so few components involved, this task is quickly completed. Two circuit board terminals are provided for connecting the sensor to the contact rail. For connecting the return signaller to other signalling units and the controller, two DIN connectors (for EEDTS Pro) or two headers (for Märklin) are provided, allowing you to install only the connectors that you need for your own system.

A FEW PRACTICAL POINTS

The receiver must be placed in the vicinity of the passing loco. The handiest solution is to place it between the rails in the bed, as shown in the photo. This minimizes possible interference from other trains running along neighbouring tracks. In this case, the IR LED that is connected to the loco decoder to transmit the loco code number must naturally be located on the underside of the loco.

If the train passes quickly over the return signaller, there is not much time for the reception of two successive data words. It is thus important that the beam of light emitted by the IR diode is adequately broad, and that it is not blocked by the adjacent undercarriage of the loco.

After a valid address has been received, the loco must close the contact rail. The PIC then passes the address through to its output, so that it can be registered by the EEDTS Pro controller.

Of course, it is possible to use a reed switch in place of the contact rail, or to employ a circuit that works via the power-takeoff slider. A correct address has to be accepted before the switching input of the PIC goes low. As soon as the loco leaves the block and the input goes high again, the output will be reset after a certain amount of time, whereupon the return signaller output is reset and the address '0' appears at its output. This means that if a loco with a conventional decoder passes the return signaller, the address '0' will be transmitted when the contact rail is closed. The programming rules of the EEDTS program can be used to deal with this.

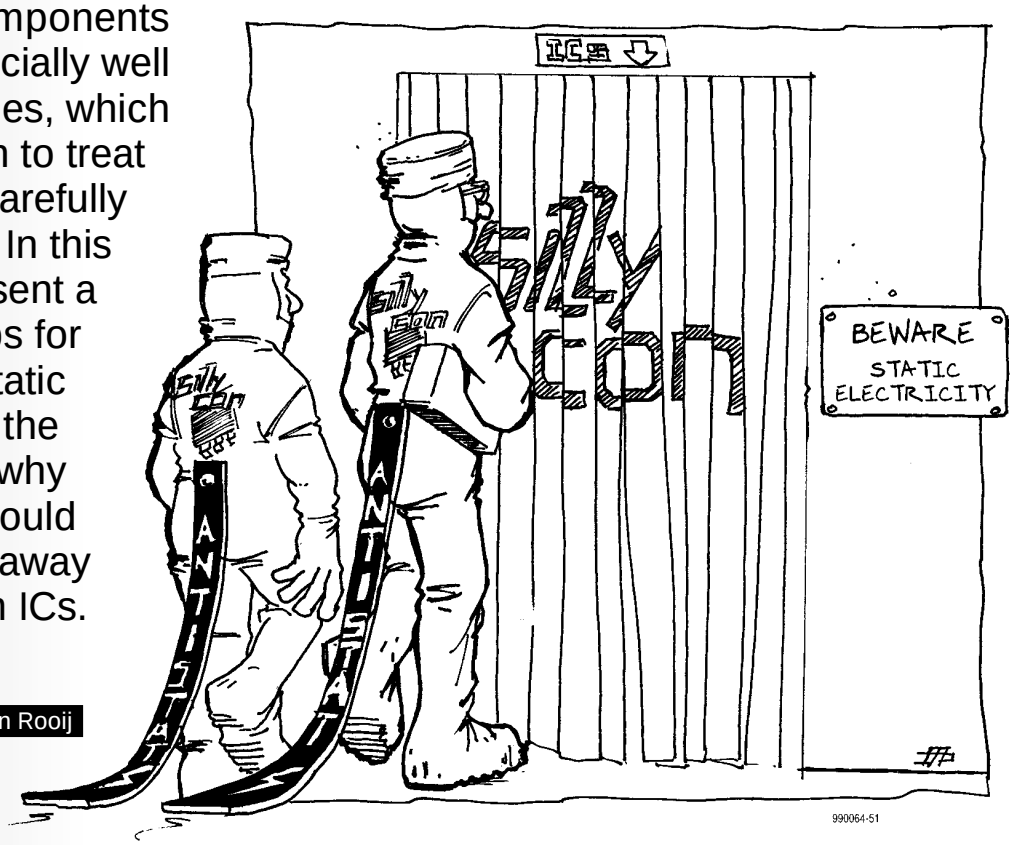
During the start-up phase, the EEDTS Pro software determines the addresses that are in use for the return buttons, which include the loco address return signaller. As soon as an address is missing, the system assumes that the last recognized address is the last one of which data may be requested. Consequently, the stream of addresses may not contain gaps. In case return signaller units are used that occupy two addresses, they always have to be assigned to at least one button. If this is not done, the next return signaller will not be recognized.

(990084)

static charges

all the shocking facts

Electrical components cannot deal especially well with static charges, which is a good reason to treat such charges carefully and with respect. In this article, we present a few practical tips for dealing with static charges, and in the process reveal why elephants should keep their trunks away from ICs.



By K. Walraven & S. van Rooij

Commonly accepted ideas about static charges still exhibit a lot of misconceptions, and in this regard they are very similar to the ideas that arose with coming of the first (steam) trains in the last century. At that time, many persons (including famous scientists) thought that the human body would spontaneously disintegrate at speeds above 100 km/h. In the meantime, we know better. It is not the speed that matters, but the change in the speed. Simply driving a car at 100 km/h presents no problems, but if you run into a wall at this speed, your speed changes very quickly from 100 km/h to nothing, and you do in fact have a serious problem.

The situation with static electricity is similar. We have all at some time charged ourselves by petting a cat or walking over a synthetic carpet. The charge is built up gradually, so that you do not notice it. Only when you touch a doorknob, or another person that has a different charge, will you have a literally electrifying experience. When this happens, the charge changes rapidly, with a painful shock as a consequence. What you feel in this case is not the potential difference, but the discharge current that flows from your body. The larger this current is, the more painful the shock. By the way, you can keep the level of the discharge current low by holding one lead of a 10 kΩ or 100 kΩ resistor and touching the other lead first to the other object. This trick guarantees that you will not feel anything, even though the charges will still be equalized (it just takes a bit longer).

VULNERABLE COMPONENTS

Electronic components are even more sensitive to static (dis)charges than we humans. Most of them are so delicate that they can be irreparably damaged during charge equalization. Nowadays, we know that it does not matter if there is a charge of 1000 volts or more on a component, as long as the component's immediate surroundings have the same charge, so that there is no potential difference. If there is a potential difference, the internal insulation of the component can be damaged (punctured), or a current can flow to equalize the different charges. This current can also destroy the component.

Based on this knowledge, we can set up a sort of 'code of behaviour' for the safe handling of electronic components. What it all comes down to is that before you pick up a component, you must always make sure that there is no potential difference between your body and the component.

For example, suppose that you have bought new memory modules for your PC and you want to install them in the

PC. The bag in your pocket that contains the memory modules can have any arbitrary potential. This bag should be made of conductive plastic, which guarantees that there cannot be any potential difference between the individual memory modules or between the modules and the bag, so that the ICs are protected. Naturally, your own body also has an arbitrary potential with respect to the bag, and thus the first thing that you should do is to eliminate any potential difference between your body and the bag. If you are not afraid of a (small) shock, this is very simple: simply take hold of the bag with your fingers. Only after you have done this can you remove the modules from the bag without risk of static damage. Since there is no potential difference, you do not have to worry about touching the modules. In fact, you should touch them, since this creates a conductive path to your body, which prevents any new potential difference from arising. Once again, for good measure: *it is better to take the modules out of the bag with your bare hands than to use an insulated pair of pliers!*

The next step is to install the modules in the PC. Once again, the PC can also have an arbitrary potential with respect to your body. You should thus touch the enclosure of the PC with your other hand (not with the one holding the modules) on a conductive place, such as a bare screw (not the plastic or a painted part). This can also result in a brief, small shock, but the result is that there is now no potential difference between the bag containing the memory modules and the PC. This is because your body acts as a conductor that has eliminated the potential difference. Keep your one hand in contact with the PC while you use the other hand to install the memory modules in their sockets on the motherboard.

WHAT ABOUT EARTHING?

You may ask, is it not much better to first earth the modules? After all, the earth potential is zero, so nothing should go wrong if the modules are earthed. Let's look at what's involved here.

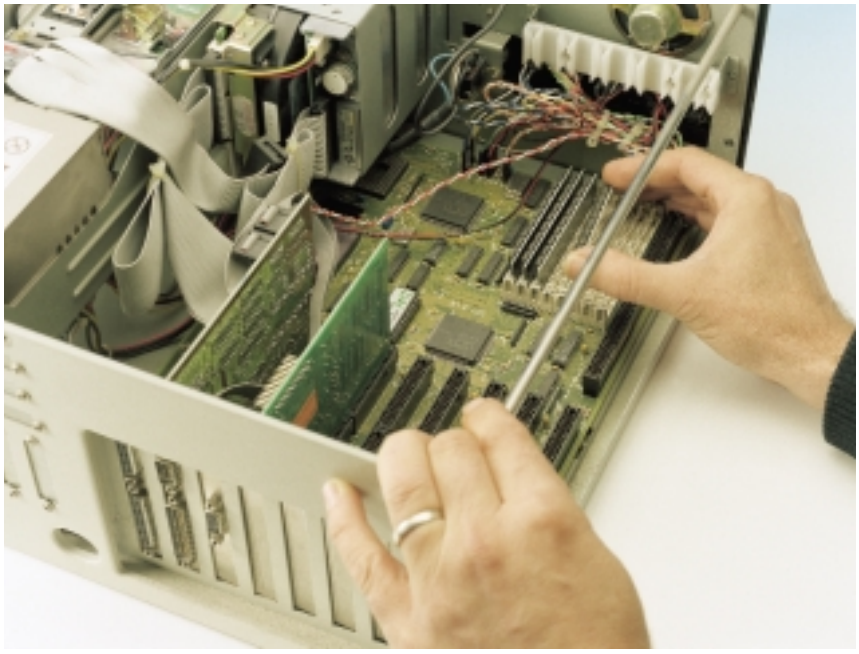
Just as you can feel completely comfortable at the top of the Eiffel tower (as long as you do not jump or fall), it does not matter to the component whether its potential is zero (earth) or some other value, as long as the potential is stable. Earth potential is thus actually not at all important, although it does form a handy reference. If you know in advance that the PC and the bag containing the modules are both at earth potential, then you do not need any measurements or other complicated procedures to be confident that there is no potential difference and thus no danger.

PEOPLE, NOT ELEPHANTS

Now comes the question, what can an IC actually tolerate? Let's consider the modern logic families. Manufacturers of ICs have noticed that people are careless. In spite of all the warnings, many ICs are damaged in daily life because people pick them up without taking special precautions. These manufacturers have thus provided their products with protective circuitry that can withstand an 'average' static discharge from the human body.

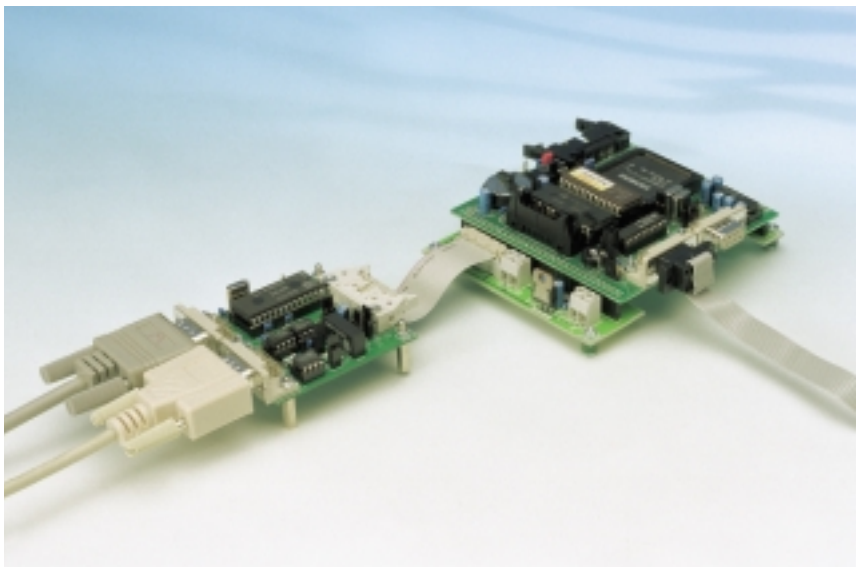
To give you an idea, a logic IC in the HC(T) family can tolerate voltages up to 2000 V (!). As a rule of thumb, we can assume that the human body has a capacitance of 100 to 150 pF with respect to its surroundings. This capacitance is discharged when a person touches an IC. If an elephant (which has a larger body and thus holds more charge in its body) were to pick up the same IC, the IC would not survive the contact. It would also not survive contact with you if your body were charged to more than 2000 V. However, practical experience has shown that the above-mentioned values are reasonable. The current that flows during a static discharge is not insignificant; we're talking about several hundred milliampères for a few microseconds!

(990064-1)



controller area network (CAN) (5)

Part 5: a CAN interface using BASIC537



If you are new to the CAN bus, look forward to some hard work before seeing the first usable results. After all, at least two microcontroller systems have to be hooked up to CAN bus controllers, and a bus-based data link implemented based on the use of talk/listen programs. Once you have data travelling over the bus, all further expansion is really simple. This article attempts to make your first practical experiments with the CAN bus as easy as possible.

Hardware & software design
by B. Kainka

The CAN bus interface described in *Elektor Electronics* November 1999 may be controlled using the BASIC537 higher programming language. BASIC537 is an EPROM version of the well-known Intel MCS51 BASIC, specially adapted and extended for the 80C537 microcontroller. Many of you will be familiar with MCS51 BASIC because it was the subject of several articles in *Elektor Electronics*. Originally, this BASIC interpreter was developed for the (now obsolete) 8052AH-BASIC microcontroller. When stored in an external (E)PROM, however, it is also great for other controllers from Intel's 80xx series and second sources (see also the 80C32 BASIC Control Computer described in *Elektor Electronics* February and March 1998). The 537 'Lite' Computer is described elsewhere in this magazine — it too is capable of running BASIC537. The ancestor of this computer, a full-blown 80C357 microcontroller system, was published in our June 1997 magazine.

The new, smaller and cheaper 537 'Lite' Computer is employed here in

combination with the CAN bus interface. The 537 'Lite' is prominently featured on this month's cover.

HARDWARE

An adaptor board was developed to implement a simple connection between the 537 'Lite' Computer (fitted with a BASIC537 EPROM) and the CAN interface board. The circuit diagram of the adaptor is given in **Figure 1**, the copper track layout and component mounting plan, in **Figure 2**. As you can see on the photograph, there are no wiring problems because the 537 'Lite' board is plugged directly on to the adaptor board. The link with the CAN interface is then made using a length of flatcable (see introductory photograph).

To solve the power supply problem in a simple way, a 5-volt voltage regulator and supply reversal protection are accommodated on the adaptor board. In this way, the adaptor board can supply +5 V to the two other boards. The upshot is that any low-cost wall adaptor supplying 9-12 volts unregulated at about 300 mA may be connected to PCB connector K2. If you already have a stabilized 5-volt line available, you may omit IC1, D1, C1 and C2 from the adaptor board, and connect the 5-V supply voltage directly to the K1 terminals on the adaptor board.

To keep cost down, the adaptor board is much smaller than the 537 'Lite' Computer board to be plugged on it. If you cut the adaptor board in two along the line indicated on the PCB overlay, and fit the two sub-boards at the right distance above a carrier (e.g. aluminium sheet), the 537 'Lite' Computer is easily plugged on to this assembly. The only wire connection to make (if necessary) is $\overline{INT2}$ between pin 12 (K3) and pin 32 (K6) (see the photograph showing the 537 'Lite' Computer board with the two adaptor sub-boards). The two adaptor sub-boards have connecting pins for the $\overline{INT2}$ link, which is also shown as a wire link on the component mounting overlay (Figure 2).

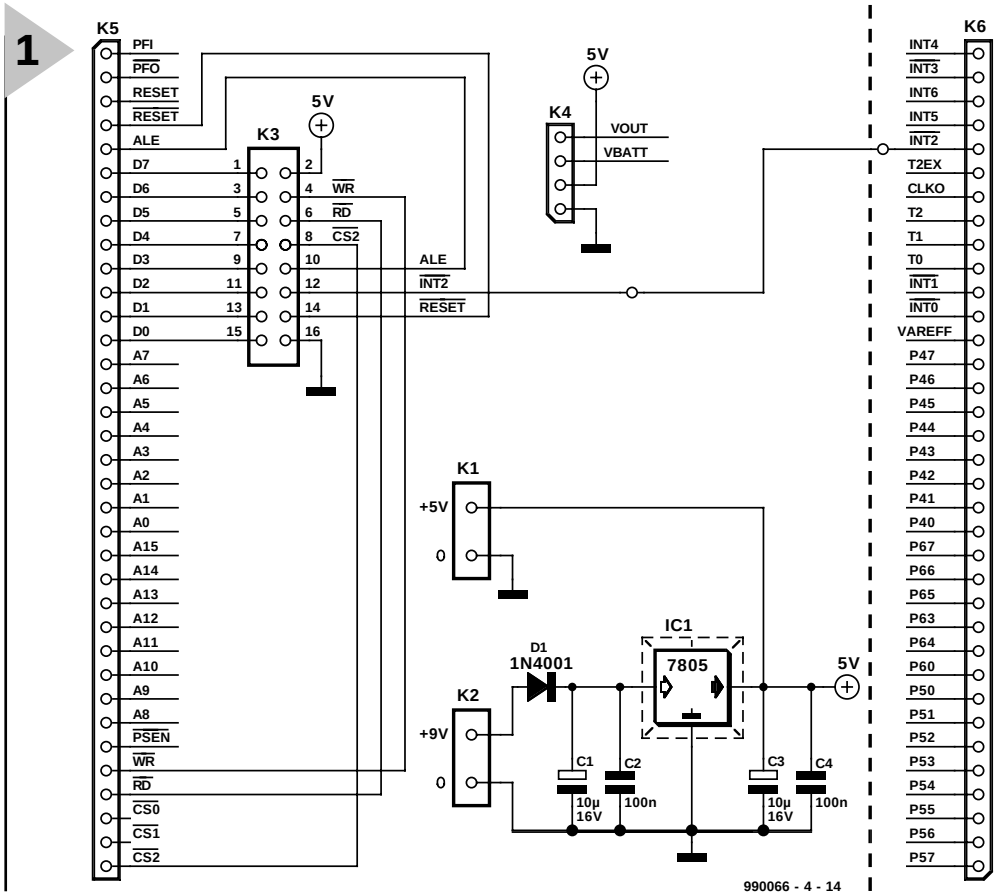


Figure 1. Circuit diagram of an adaptor board that creates a simple link between the 537 'Lite' Computer board and the CAN bus interface.

CONTROL IT ALL IN BASIC

In essence, all you need to be able to control the CAN interface board is a program that looks after a bank of registers starting at address F000h in the CAN controller SJA1000. The XBY operator is employed for all access to addresses in the external RAM area and peripherals.

To make the introduction as easy going as possible, the following paragraphs describe the simplest case of a data link between two 80C537 systems. The essential settings are supplied as defaults by the program. The communication runs at 20 kbits/s. Message are sent without the RTR bit — in other words, no reply is requested. The two systems should fulfil the following functions:

System 1 regularly sends messages with Identifier 300 and containing eight bytes. The data originates from the first eight channels of the A-D converter. The system continuously performs measurements on eight analogue inputs. The messages it transmits may be received and processed by any other system connected to the bus.

System 2 receives all messages on the bus and copies them to the PC by way of the RS232 interface. In fact, this is a simple CAN monitor that allows you to pick up and examine all data traffic on the CAN bus.

A block diagram of the system configuration is given in **Figure 3**. A special cable is not required to link the two systems. Our first experiments in the lab indicated that a simple two-wire link between pins 4 and 8 of the CAN plugs is adequate if the total length does not exceed about 1 metre. With such a short cable, no difference was noted between the termination resistors being present or not.

TRANSMIT PROGRAM AND TEST

The transmit program for Controller 1 is given in **Listing 1**. The CAN controller SJA1000 is addressed by the 80537 system via base address 0F000h. The address range is defined in line 95 (BA = 0F000h). If you use a different system, all you have to do (initially) is modify BA accordingly. The initialisation is carried out as described in the article on the CAN bus interface hardware. In lines 110 and 200, the results of the register programming are requested. The program then waits for a register bit to go to a specific state. In case the controller is not found on the bus, or does not function correctly, the program will 'hang' at this point. If everything is successful, however, you are greeted with these messages:

```
Reset OK
Init OK
```

To start with, it is sufficient to execute the initialisation routine up to line 200. An initial check may be made by looking for a rectangular signal at the test pin on the controller board. Whereas this pin supplies 8 MHz before the initialisation, you will then find 2 MHz. If this is okay so far, you may safely

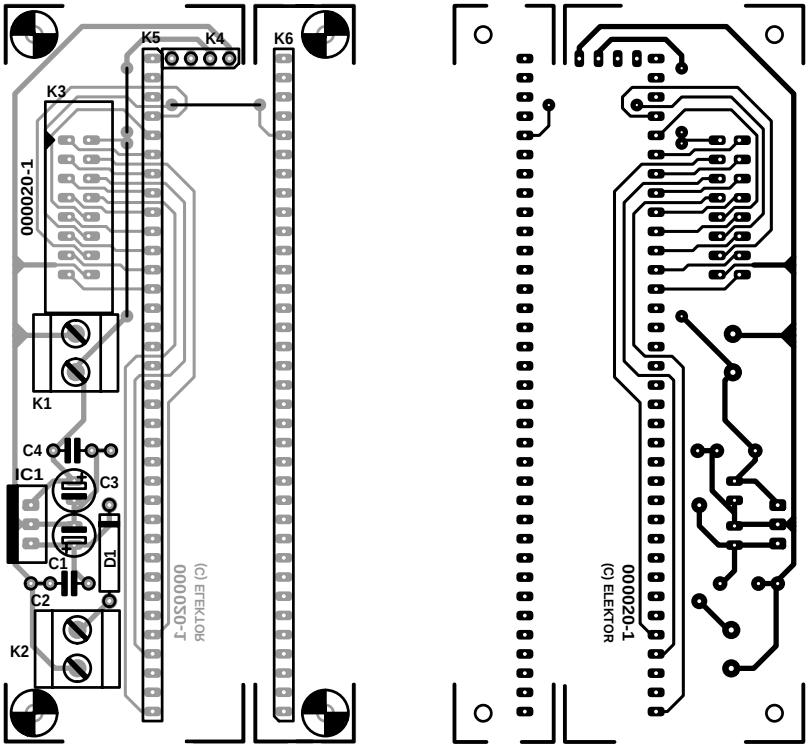


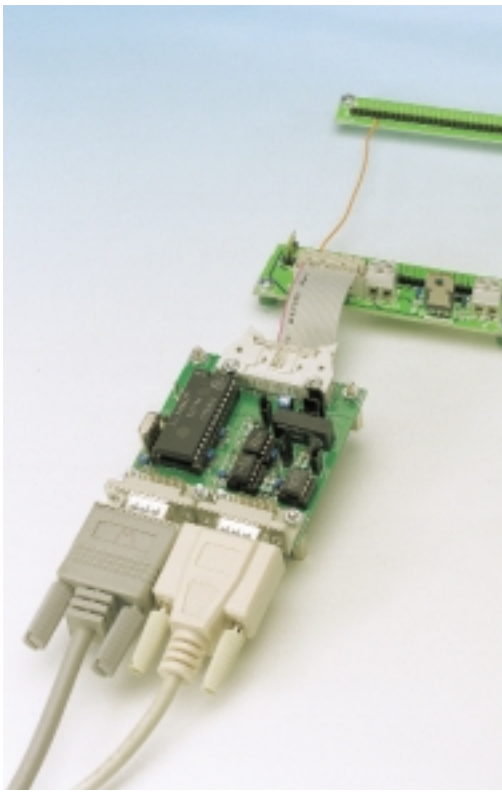
Figure 2. Layout and component mounting plan of the adaptor board.

COMPONENTS LIST

Capacitors:
C1, C3 = 10µF 16V (radial)
C2, C4 = 100nF (ceramic)

Semiconductors:
D1 = 1N4001
IC1 = 7805

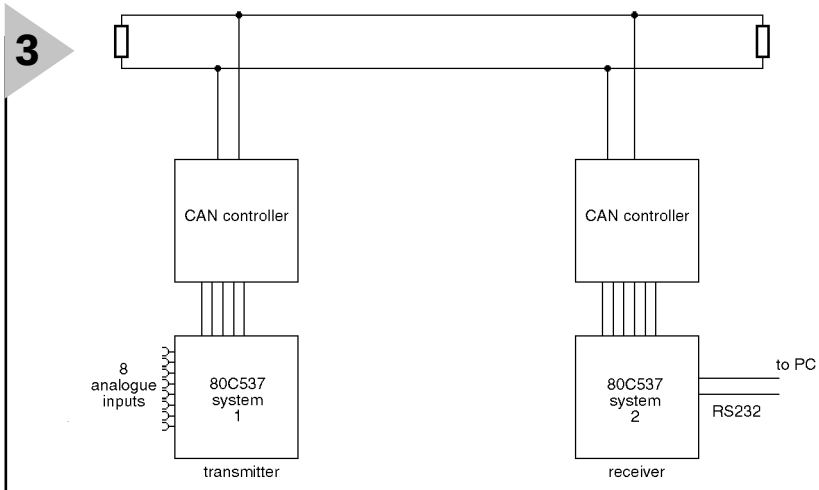
Miscellaneous:
K1 = 2-way PCB terminal block, raster 5mm
K2 = 2-way PCB terminal block, raster 5mm
K3 = boxheader, straight, 16 pins
K4 = pin header, 1 row, 4 pins
K5, K6 = pin header, one row, straight, 35 pins



assume that the controller is driven with the proper signals.

Now the complete program may be loaded and executed. Experienced as you are, you will no doubt have an oscilloscope ready to observe data traffic on the bus. Without a connection having been made to a second system, you will be able to find signals on the data lines. After a hardware reset and without an initialisation you will not be able to find the 'inactive' level of 2.5 V on the two wires. As soon as the transmit program is started, data is easily recognized as rectangular signals with a level of 1 V. The shortest logic levels are just 50 µs long, which means that the transmission rate is indeed 20 kBits/s. However, you will not fail to see that there is a quasi-steady datastream with 2-ms pauses, rather than short data packets as would be expected. Not to worry, however, this is the normal behaviour of the controller as long as it has not detected any intelligence on the CAN bus. Yet, it is not sufficient to connect the second controller via the two-wire link, because this, like system 1, has to be initialised. By the way, the transmitting station will continue to send a quasi-continuous datastream even if the BASIC program is terminated.

Figure 3. Block diagram showing how the CAN bus is linked to the two 80537 systems programmed in 537 BASIC.

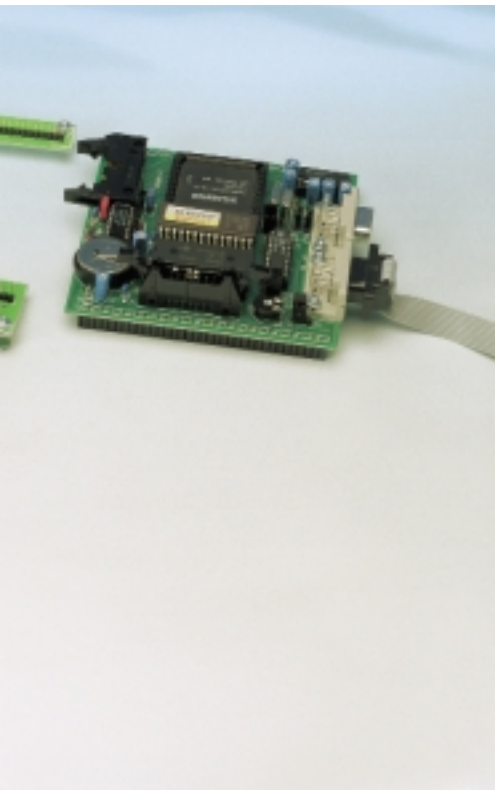


AT THE RECEIVER SIDE

Now we are ready to 'deploy' the receiver program as given in **Listing 2**. This program is for system number 2. As you can see from the listing, the initialisation is the same as that used for the transmitter. As soon as the initialisation is done and the message "Init OK" has appeared on the PC display, the transmitting controller will com-

mence its normal operation. From then on, short data packets with a length of just over 5 ms will start to appear on the CAN bus. Finally, the CAN bus functions as you, the interested reader, would like to see it: data packets being sent back and forth over the bus without any indication of their being read anywhere at all!

The actual receiver program starts at line 500 and waits for a message which is announced by controller status bit zero. As soon as a data packet has arrived, the program may read ten bytes from the controller. The first two contain the message ID. It is recovered from two bytes in line 570 and then sent to the display. As expected, it is the 'ID', 300, which was arbitrarily defined in the transmitter program.



The user data are read in a loop and displayed in line 610. There , you (finally…) get the measured values on the eight analogue inputs of the first controller system. **Figure 4** shows the received data in a terminal window.

FINALLY: THREE ON THE BUS

Of course, the results up to now could have been obtained with a rather simpler RS232 interface. The CAN bus however will typically not unleash its power until more than two devices are connected to the bus. To end the 'lonely' existence of the two systems discussed so far, a third 'CANable' device should be added. The program CAN3.BAS shown in **Listing 3** (without initialisation!) performs the following functions:

Listing 1. Transmitter program CAN1.BAS.

```
90    REM Init CAN Controller
95    BA=0F000H
100   XBY(BA+00H)=01H :  REM Reset Mode
110   IF (XBY(BA+00H).AND.1)<>1 THEN  GOTO 110
111   PRINT ″Reset OK″
120   XBY(BA+1FH)=43H :  REM CDR, 2 MHz
130   XBY(BA+04H)=0 :  REM ACR
140   XBY(BA+05H)=0FFH :  REM AMR, Acceptance Mask, all
150   XBY(BA+06H)=53H :  REM BTR0, 20 Kbit/s*
160   XBY(BA+07H)=2FH :  REM BTR1
170   XBY(BA+08H)=1AH :  REM OCR;
180   XBY(BA+01H)=0EH :  REM CMR, end sleep mode
190   XBY(BA+00H)=0 :  REM CR, end reset mode
200   IF (XBY(BA+00H).AND.1)>0 THEN  GOTO 200
201   PRINT ″init ok″
500   REM ************* Main Loop ***************
501   REM Send 8 Bytes of AD-Data in message 300
510   FOR N=0 TO 7
520   XBY(BA+0CH+N)=AD(N) :  REM fill TB1..TB8
530   NEXT N
540   ID=300 :  REM Message Identifier
550   DFL=8 :  REM 8 Bytes
560   GOSUB 1000 :  REM Send Massage
570   FOR T=1 TO 1000 :  NEXT T
580   GOTO 500
1000  REM ************* Send CAN Telegram *************
1010  IF (XBY(BA+02H).AND.4)=0 THEN  GOTO 1010 :  REM SR
1020  XBY(BA+0AH)=INT(ID/8) :  REM IDT1
1030  XBY(BA+0BH)=(ID-8*INT(ID/8))*32+DFL :  REM IDT2
1040  XBY(BA+01H)=0DH :  REM CMR, start transmission
1050   RETURN
```

Listing 2. Receiver program CAN2.BAS

```
90    REM Init CAN Controller
95    BA=0F000H
100   XBY(BA+00 H)=01H :  REM Reset Mode
110   IF (XBY(BA+00H).AND.1)<>1 THEN  GOTO 110
111   PRINT ″Reset OK″
120   XBY(BA+1FH)=43H :  REM CDR, 2 MHz
130   XBY(BA+04H)=0 :  REM ACR
140   XBY(BA+05H)=0FFH :  REM AMR, Acceptance Mask, all
150   XBY(BA+06H)=53H :  REM BTR0, 20 Kbit/s*
160   XBY(BA+07H)=2FH :  REM BTR1
170   XBY(BA+08H)=1AH :  REM OCR;
180   XBY(BA+01H)=0EH :  REM CMR, end sleep mode
190   XBY(BA+00H)=0 :  REM CR, end reset mode
200   IF (XBY(BA+00H).AND.1)>0 THEN  GOTO 200
201   PRINT ″Init OK″
500   REM ******* Receiver Main Loop *************
510   SR=XBY(BA+02H) :  REM Status Register
520   REM Error Detection and Clear Data Overrun
530   if (SR .AND. 2) = 2 then XBY(BA+01H)=8: :Goto 510
540   REM Get Receive Status
550   if (SR .AND. 1) =0 then goto 510
560   REM Read received message
570   ID=XBY(BA+14H)*8+INT(XBY(BA+15H)/32) :  PRINT ID
580   DFL=XBY(BA+15H).AND.15 : rem   Data Length
590   RTR=(XBY(0FE15H).AND.16)/16 :  REM RTR not used
600   FOR N=0 To 7
610   PRINT N ,XBY(BA+16H+N)
620   NEXT N
630   XBY(BA+01H)=0CH :  REM Release Receive Buffer
640   GOTO 510
```

Listing 3. Receiver & Transmitter program CAN3.BAS without initialisation.

```
500 REM ************ Main Loop ***************
505 REM ************ Receiver ****************
510 SR=XBY(BA+02H) :  REM Status Register
520 REM Error Detection and Clear Data Overrun
530 IF (SR.AND.2)=2 THEN XBY(BA+01H)=8 :  GOTO 510
550 IF (SR.AND.1)=0 THEN  GOTO 510
560 REM Read received message
570 ID=XBY(BA+14H)*8+INT(XBY(BA+15H)/32): Print ID
580 DFL=XBY(BA+15H).AND.15 :  REM   Data Length
590 RTR=(XBY(0FE15H).AND.16)/16 :  REM RTR not used
600 IF ID<>300 THEN  GOTO 660
610 PORT=0
620 IF XBY(BA+16H+0)>100 THEN PORT=PORT+1
630 IF XBY(BA+16H+1)>100 THEN PORT=PORT+2
640 IF XBY(BA+16H+2)>100 THEN PORT=PORT+4
650 WRSFR 0E8H,PORT :  REM Port 4 Output
660 XBY(BA+01H)=0CH :  REM Release Receive Buffer
800 REM ******** Send AD-Data ***********
810 FOR N=0 TO 7
820 XBY(BA+0CH+N)=AD(N) :  REM fill TB1..TB8
830 NEXT N
840 ID=500 :  REM Message Identifier
850 DFL=8 :  REM 8 Bytes
860 GOSUB 1000 :  REM Send Message
870 FOR T=1 TO 1000 :  NEXT T
880 GOTO 500
1000 REM ******* Send CAN Telegram *************
1010 IF (XBY(BA+02H).AND.4)=0 THEN  GOTO 1010 :  REM SR
1020 XBY(BA+0AH)=INT(ID/8) :  REM IDT1
1030 XBY(BA+0BH)=(ID-8*INT(ID/8))*32+DFL :  REM IDT2
1040 XBY(BA+01H)=0DH :  REM CMR, Start Transmission
1050 RETURN
```

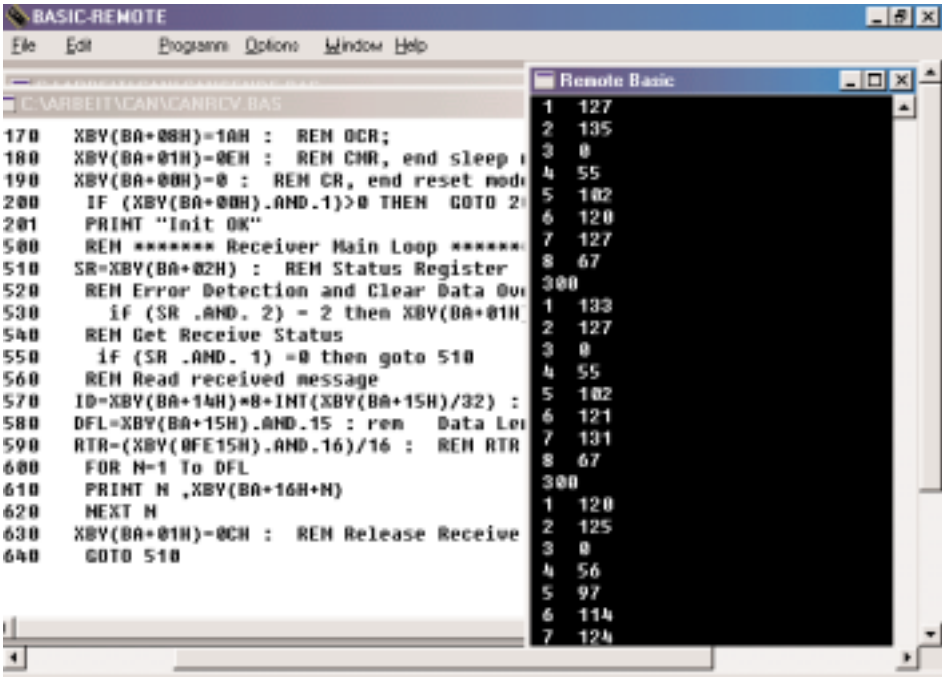


Figure 4. Received data in the terminal window of BASIC537.

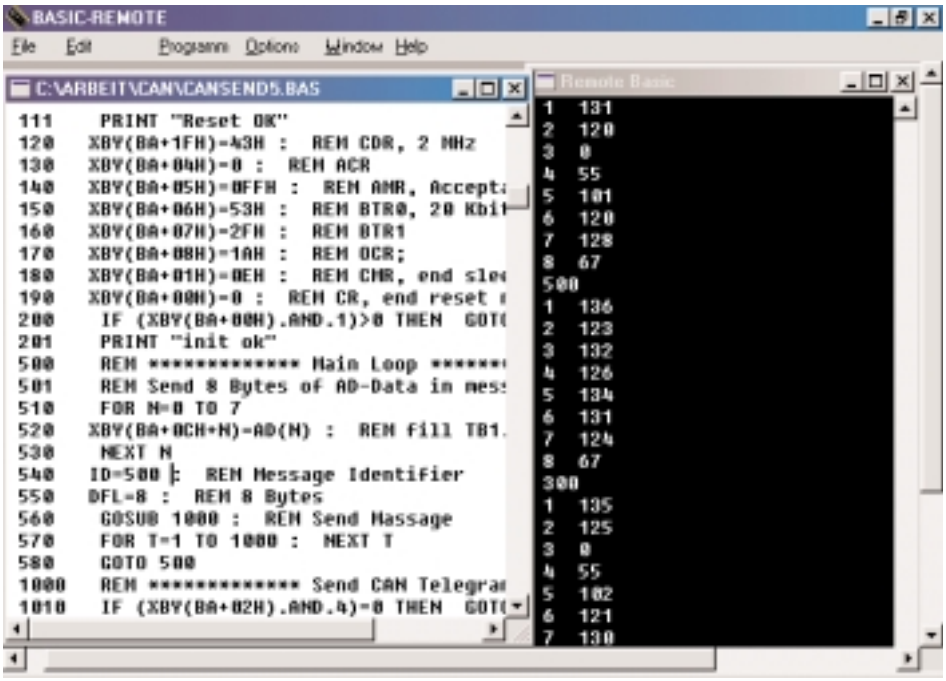


Figure 5. Reception of messages with Identifiers (IDs) '300' and '500'.

It receives all messages but only processes the ones with the ID '300'. The first three transmitted measurement values are compared with certain extreme values and switch on three lines on Port 4 if a particular extreme is exceeded.

After processing of the received message, a message with the ID '500' is returned, where all A-D channels are measured and transmitted again. As soon as the third system is connected to the bus, System 2 will also supply data with ID '500' to the terminal (see **Figure 5**).

990066-4

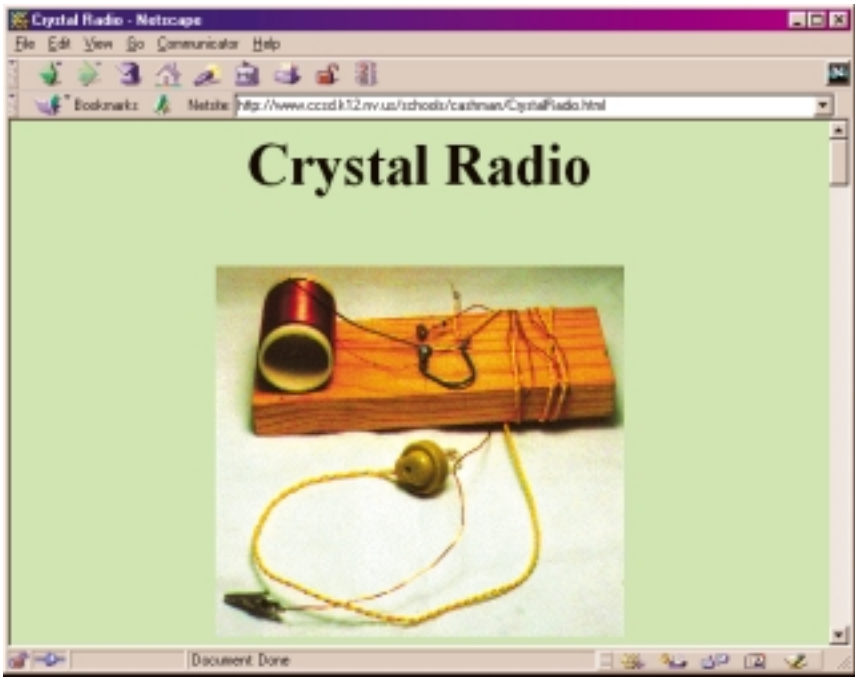
Note: the three program listings discussed in this article are available for downloading from the Elektor Electronics website at *www.elektor-electronics.co.uk*

Article editing (German original): E. Krempelsauer
Design editing: K. Walraven

build a crystal radio

back to the early days

Anyone with a basic knowledge of electronics should know what a crystal radio is: the first and most rudimentary model of a radio receiver, consisting of no more than a diode (the 'crystal'), an antenna, an inductor and an earpiece. Many designs of crystal sets may be found on the Internet, often with extensive construction details and nice pictures.



In today's ultra-modern high-tech society it is 'normal' for extremely complex electronics to be applied in many different equipment and appliances. Come to think of it, electronics has not been with us that long, has it? Many electronics hobbyists (and professionals) have fond memories about their first experiments in electronics like building a simple radio with just one 'active' element.

The fact that many Internet websites may be found covering the subject 'crystal radio' is proof that many electronics enthusiasts do more than just ruminate halcyon days.

The websites mainly contain simple crystal radios. Although many of the designs are heavily educational, being aimed at beginners in schools and colleges, the information is also of interest to hobbyists in general. Some addresses:

Building a basic AM broadcast crystal set *(www.midnightscience.com/project.html)*
Crystal Radio *(www.ccsd.k12.nv.us/schools/cashman/CrystalRadio.html)*
Crystal radio project *(www.datasync.com/~ignatz/electro/crystal/crystal.htm)*

The designs we came across also vary in complexity. Although genuine crystal receivers of course contain just one germanium diode as the active element, there are also more extensive designs where transistors and even integrated circuits have been added. Here, too, three examples:

High power crystal set *(http://hibp.ecse.rpi.edu/~john/xtal.html)*
A "convertible" crystal radio *(www.glynn.k12.ga.us/~opool/XTAL/radio.htm)*
Short wave radio project *(http://www.datasync.com/~ignatz/electro/receive/receive.htm)*

An exceptionally clear description of a crystal shortwave receiver may be found at **Radio Wrinkles**. This particular radio design has been dubbed "EconOceanic". It is capable of receiving the medium-wave band and most shortwave bands between 1.7 and 17 MHz. If you are interested, have a look at *www.antiqureradio.org.econmain.htm*

A fine overview of various crystal radio designs may be found on the **Crystal Set Design** pages. The presentation starts with a simple diode detector and ends with a two-inductor design. It's all there at *www.webex.net/~skywaves/xtalset102/xtalset102.htm*

There are also a number of crystal radio clubs (interest groups) on the Internet. Of these we should mention the **Xtal Set Society** (*http://www.midnightscience.com/*) and the **Crystal Set radio club** (*http://clubs/yahoo.com.clubs/thecrystalsetradioclub*). Both clubs exclusively cover building and experimenting with simple radio receivers.

Al Klase on his website shows some fine pieces of home wrought radio. A professional electronics engineer, In his spare time Al obviously takes pleasure in building crystal radios. His designs are marked by sophisticated mechanical and electrical designs and constructions. Don't take our word for it, take a look at the photographs and diagrams at *www.webex.net/~skywaves/HP002/HP-002.htm*

A nice collection of antique crystal radios may be seen at a number of sites including **Scott's Crystal Radios**. The pictures really make you want to go back in time and once again build your own crystal set. (*http://members.aol.com/scottswim/*)

(005005-1)

Text: H. Baggen

Hardware mode controls		
PIN NAME	**PIN NUMBER**	**DESCRIPTION**
RESV	13	Reserved, Not Used
LRIP	12	LRCIN Polarity LRIP = H: LRCIN= H = Left Channel, LRCIN= L = Right Channel LRIP = L: LRCIN= L = Left Channel, LRCIN = H = Right Channel
CKO	11	CLKO Output Frequency CKO = H: CLKO Frequency = XTI/2 CKO = L: CLKO Frequency = XTI
MUTE	15	Soft Mute Control: H = Mute Off, L = Mute On
I^2S IW0 IW1	3 4 5	Input Data Format Controls **I^2S IW1 IW0 INPUT FORMAT** L L L 16-Bit, Standard, MSB-First, Right-Justified L L H 20-Bit, Standard, MSB-First, Right-Justified L H L 24-Bit, Standard, MSB-First, Right-Justified L H H 24-Bit, MSB-First, Left-Justified H L L 16-Bit, I^2S H L H 24-Bit, I^2S
SRO	27	Digital Filter Roll-Off: H = Slow, L = Sharp
OW0 OW1	19 20	Output Data Word Length Controls **OW1 OW0 OUTPUT FORMAT** L L 16-Bit, MSB-First L H 18-Bit, MSB-First H L 20-Bit, MSB-First H H 24-Bit, MSB-First
SF0 SF1	17 18	Sample Rate Selection for the Digital De-Emphasis Control **SF1 SF0 SAMPLING RATE** L L 44.1kHz L H Reserved, Not Used H L 48kHz H H 32kHz
DEM	16	Digital De-Emphasis: H = On, L = Off

DF1704

Integrated circuits
Special Function, AF

ELEKTOR ELECTRONICS

DATASHEET 1/2000

DF1704
Soundplus™ Stereo, 24-bit, 96kHz 8× Oversampling Digital Interpolation Filter Digital-to-Analog Converter

Manufacturer

BURR-BROWN®

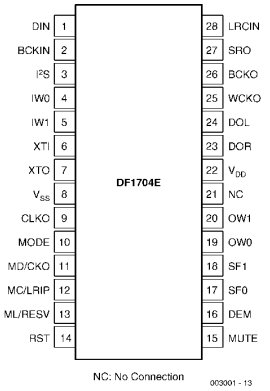
Burr-Brown, P.O. Box 11400, Tucson, AZ 85734,
U.S.A. Tel. (520) 746-1111, fax (520) 889-1510.
Internet: http://www.burr-brown.com/

Features

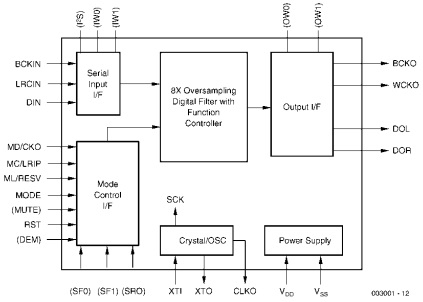
- COMPANION DIGITAL FILTER FOR THE PCM1704 24-BIT AUDIO DAC
- HIGH PERFORMANCE FILTER:
  - Stopband Attenuation: –115dB
  - Passband Ripple: ±0.00005dB
- AUDIO INTERFACE:
  - Input Data Formats: Standard, Left-Justified, and I^2S
  - Input Word Length: 16, 20, or 24 Bits
  - Output Word Length: 16, 18, 20, or 24 Bits
  - Sampling Frequency: 32kHz to 96kHz
- SYSTEM CLOCK: $256f_S$, $384f_S$, $512f_S$, $768f_S$
- ON-CHIP CRYSTAL OSCILLATOR
- PROGRAMMABLE FUNCTIONS:
  - Hardware or Software Control Modes
  - Sharp or Slow Roll-Off Filter Response
  - Soft Mute
  - Digital De-Emphasis
  - Independent Left/Right Digital Attenuation
- +5V SINGLE-SUPPLY OPERATION
- SMALL 28-LEAD SSOP PACKAGE

Application Example
Audio DAC2000, *Elektor Electronics* November 1999 – January 2000.

Description
The DF1704 is a high performance, stereo, 8× oversampling digital interpolation filter designed for high-end consumer and professional audio applications. The DF1704 supports 24-bit, 96kHz operation and features user-programmable functions, including selectable filter response, de-emphasis, attenuation, and input/output data formats.
The DF1704 is the ideal companion for Burr-Brown's PCM1704 24-bit audio digital-to-analog converter. This combination allows for construction of very high performance audio systems and components.



Pin configuration.



DF1704 bock diagram.

Programmable Functions
The DF1704 includes a number of programmable features, with most being accessible from either Hardware or Software mode. The table below summarizes the user programmable functions for both modes of operation.

FUNCTION	SOFTWARE (MODE = H)	HARDWARE (MODE = L)	RESET DEFAULT (Software Mode)
Input Data Format Selection	O	O	Standard Format
Input Word Length Selection	O	O	16 Bits
Output Word Length Selection	O	O	16 Bits
LRCIN Polarity Selection	O	O	Left/Right = High/Low
Digital De-Emphasis	O	O	OFF
Soft Mute	O	O	OFF
Digital Attenuation	O	X	0dB, Independent L/R
Sample Rate for De-Emphasis Function	O	O	44.1 kHz
Filter Roll-Off Selection	O	O	Sharp Roll-Off Selected
CLKO Output Frequency Selection	O	O	Same As XTI Input
Legend: O = User Programmable, X = Not Available.			

Hardware Mode Controls
With MODE = L, the DF1704 may be configured by utilizing several user-programmable pins. The following is a brief summary of the pin functions. The table below provides more details on setting the hardware mode controls.

Pins I^2S, IW0, and IW1 are used to select the audio data input format and word length.

Pins OW0 and OW1 are used to select the output data word length.

The DEM pin is used to enable and disable the digital de-emphasis function. De-emphasis is only available for 32kHz, 44.1kHz, and 48kHz sample rates.

Pins SF0 and SF1 are used to select the sample rate for the de-emphasis function.

The SRO pin is used to select the digital filter response, either sharp or slow roll-off.

The MUTE pin is used to enable or disable the soft mute function.

The CKO pin is used to select the clock frequency seen at the CLKO pin, either XTI or XTI ÷ 2.

The LRIP pin is used to select the polarity used for the audio input left/right clock, LRCIN.

Finally, the RESV pin is not used by the current DF1704 design, but is reserved for future use.

DF1704

Integrated circuits
Special Function, AF

ELEKTOR ELECTRONICS

DATASHEET 1/2000

Pin Assignments			
PIN	**NAME**	**I/O**	**DESCRIPTION**
1	DIN	IN	Serial Audio Data Input (3)
2	BCKIN	IN	Bit Clock Input for Serial Audio Data (3)
3	I^2S	IN	Input Audio Data Format Selection (2, 4)
4	IW0	IN	Input Audio Data Word Selection (2, 4)
5	IW1	IN	Input Audio Data Word Selection (2, 4)
6	XTI	IN	Oscillator Input /External Clock Input
7	XTO	OUT	Oscillator Output
8	VSS	—	Digital Ground
9	CLKO	OUT	Buffered System Clock Output
10	MODE	IN	Mode Control Selection (H: Software, L: Hardware) (1)
11	MD/CKO	IN	Control Data Input/Clock Output Frequency Select (1, 5)
12	MC/LRIP	IN	Control Data Clock/Polarity of LRCK Select (1, 5)
13	ML/RESV	IN	Control Data Latch/Reserved (1, 5)
14	RST	IN	Reset. When this pin is LOW, the digital filter is held in reset. (1)
15	MUTE	IN	Mute Control (1, 4)
16	DEM	IN	De-Emphasis Control (2, 4)
17	SF0	IN	Sampling Rate Select for De-emphasis (2, 4)
18	SF1	IN	Sampling Rate Select for De-emphasis (2, 4)
19	OW0	IN	Output Audio Data Word and Format Select (2, 4)
20	OW1	IN	Output Audio Data Word and Format Select (2, 4)
21	NC	—	No Connection
22	V_{DD}	—	Digital Power, +5V
23	DOR	OUT	Rch, Serial Audio Data Output
24	DOL	OUT	Lch, Serial Audio Data Output
25	WCKO	OUT	Word Clock for Serial Audio Data Output
26	BCKO	OUT	Bit Clock for Serial Audio Data Output
27	SRO	IN	Filter Response Select (2, 4)
28	LRCIN	IN	L/R Clock Input (f S) for Serial Audio Data (3)

NOTES: (1) Pins 10-15; Schmitt-Trigger input with pull-up resistor. (2) Pins 3-5, 16-20, 27; Schmitt-Trigger input with pull-down resister. (3) Pins 1, 2, 28; Schmitt-Trigger input.
(4) Pins 3-5, 15-20, 27; these pins are invalid when MODE (pin 10) is HIGH.
(5) Pins 11-13; these pins have different functions corresponding to MODE (pin 10), (HIGH/LOW).