THE ELECTRONICS & COMPUTER MAGAZINE

SEPTEMBER 1999
£ 2·85

ELEKTOR ELECTRONICS

PC TOPICS:

• processor upgrading on a budget •
• hexadecimal keypad for PCs •
• structured assembly-code programming • with SXp

digital PID control

stepper motor control

BASIC Stamp course

design your own robot

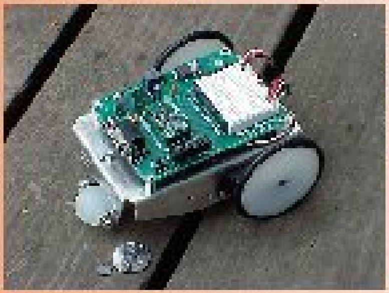
CAN bus

Data comms for car, home and industry

September 1999 Number 280

Elektor Electronics
Volume 25
ISSN 0268/4519

CONTENTS

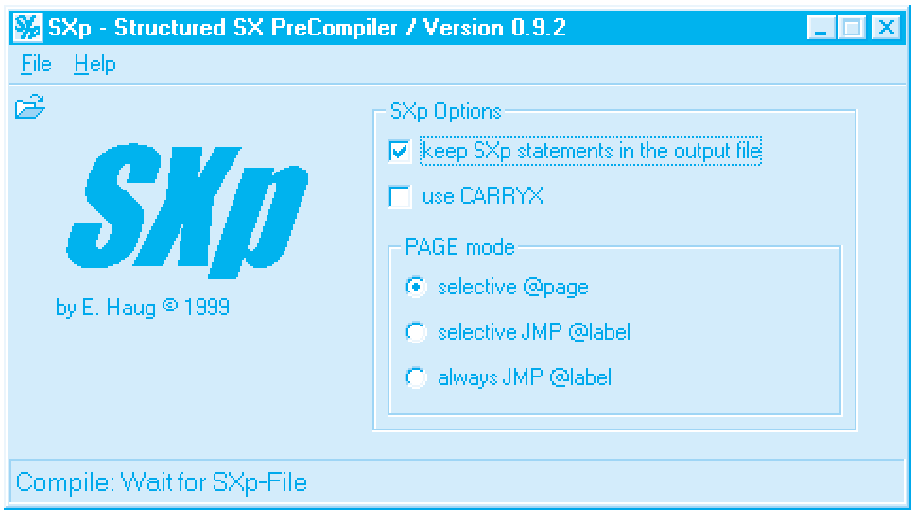


Higher programming language or assembly code? Because of market developments and ever-shortening design cycles, today's software developers tend to employ higher programming languages like C or C+ for microcontroller applications. In many cases, this choice is more or less forced by requirements like program documentation, product support and deadlines. However, higher-level programming may not be possible when memory resources are limited, or when the application is time-critical. In such cases, an assembler supporting structured programming offers the best of both worlds, with high program speeds and clever memory usage thrown in as useful extras.

By E. Haug

SXp precompiler

structured assembly-code programming for Scenix™ SX microcontrollers


SXp - Structured SX PreCompiler / Version 0.9.2

SXp Options

- [✓] keep SXp statements in the output file
- [ ] use CARRYX

PAGE mode

- (●) selective @page
- ( ) selective JMP @label
- ( ) always JMP @label

Compile: Wait for SXp-File

Based on examples from the past, including R-D. Klein's STRUKTA (1983), a CPM-compatible precompiler for Z80/8080 code, the author developed the *SXp* precompiler described in this *Elektor Electronics* PC Topics article. As indicated by its name, *SXp* supports the microcontrollers from the 'SX' family supplied by Scenix [1, 2]. Statements for structured programming of interrogation routines, branching and loops allow the high processing speed and optimum memory usage of assembly-code programming to be linked with those clearly laid out, easily created structures many of you have learned to value of higher programming languages like C.

Structured programming

To begin with, a frequently made error should be eradicated: control statements for *structured programming* are not the same as *conditional assembly*, a feature available in almost any assembler package. Remarkably, many experts are not aware of this small but essential difference ("IF-THEN-ELSE, yes well my assembler does that too").

Control statements as they are made available by *SXp* allow structured programming in SX assembler language, with the following advantages:

simplified program development and simplified debugging;
vastly improved legibility of the program and its documentation;
shorter program development cycles, resulting in higher productivity;
improved program reliability and maintenance;
more fun in writing SX programs.

SXp offers the programmer a set of statements to help him/her realize routines ('situations') performing these functions: interrogation, branching or loops, as they occur in almost any program. Using assembly code programming, such standard routines are not easily made, and invariably subject to programming errors and doubts (like "is that carry flag set or not?").

The negative effect of the *SXp* statements on the code efficiency of the program is negligible if at all existent. In other words, both processor performance and memory usage are comparable to what may be obtained with straight assembly code programming. When using *SXp*, all advantages of assembly code programming remain available simply because all non-*SXp* statements represent the regular assembly language commands. That, in a nutshell, is how it's done!

SXp — what's it for?

SXp — for the Windows 95/98/NT platforms — is a precompiler capable of converting statements for structured programming (the so-called *SXp* statements) into assembly language code for the Scenix SX family of microcontrollers.

The actual SX assembler commands are scattered between the *SXp* statements, and they remain unaffected by the *SXp* Precompiler. Of course, routines may be nested by inserting any number of additional *SXp* statements. The code generated by *SXp* has to be converted into the final SX object code by a Scenix compatible assembler. Note that only one-word basic commands of the SX are used.

The assembler source code program created by SXp (that is, the output file) may be used for simulation and debugging sessions (using, for example, SX-KEY™ from Parallax). Optionally, the *SXp* statements may be retained as comment in the assembly code file. This will be particularly useful with debugging.

A further option is the inclusion of DEVICE CARRYX of the SX processor. This flag represents a useful speciality of the SX, although it must be said that it is also a troublesome feature. The CARRYX flag is used because the SX has no separate commands for addition and subtraction with or without inclusion of the carry/borrow bit.

An essential feature of SXp is the exclusive use of the working register W, and, if necessary, the specified loop register. No other SX registers are employed or modified (except, of course, the program counter and, occasionally, the status register). The SX stack is not used either by *SXp*. Consequently, the SX program creator has full and exclusive control over all SX registers.

Depending on the programming method, *SXp* should achieve roughly the same code efficiency as pure assembly-code programming — but at the same time much better documentation and program legibility is guaranteed, the upshot being that in the end you are having more fun!

Depending on the memory configuration of the computer on which *SXp* is installed, all *SXp* structures may be nested to any depth. Because *SXp* does not employ the (relatively small) 8-level SX stack for its return address information, jumping is allowed at any location in the SX program, even into other *SXp* structures. With the previously mentioned STRUKTA program, great care was required in do doing so because this precompiler executed the return jump from structures via the stack.

What it can't do

SXp does not generate the SX object code directly from the *SXp* source program. Because *SXp* is not aware of absolute program addresses, it is not capable of detecting page boundaries (remember, the addressable memory range of the SX is organised as 'pages'). In such cases, one of the *SXp* Page Mode Options should be employed [4].

Furthermore, *SXp* is unable to check if defined and actually used symbols and data are within their allowable range. Running this check remains the exclusive task of the assembler.

SXp statements

The statements implemented in version 0.9 of *SXp* are summarized in Table 1. Extensive descriptions of all statements may be found in [3]. Below, some basic information is given regarding the *SXp* syntax.

Instructions represents any number (including zero) of SX assembler instructions and/or *SXp* statements. *SXp* statements may be nested to any depth.

THEN is only used in combination with PAGE indicators, and may be omitted in all other cases.

CONDITION represents comparisons (at least one SX register), register bit interrogations or status flag interrogations.

REGISTER ASSIGN indicates the number of loop iterations. The specified register may have been loaded beforehand, or its assigning is part of the command. The loop register may be filled with a constant, the contents of another SX File Register, or the W register.

ADDRESS stands for a non-local address at any location in the *SXp* program (take care, however, with jumps from subroutines). EXITIF-TO is the only non-structured *SXp* statement. It allows a code-saving and 'quick way out' of an *SXp* structure. An unconditional jump is simply programmed using the assembler command JMP.

SXp only uses the W register in the SX, plus the selected loop registers. Furthermore, the program counter and the status register are subject to modifications (the latter depending on

Table 1. SXp Directives (SXp version 0.9.x)

IF	*CONDITION*	**[THEN]**	; First interrogation
	Instructions		
ELSEIF	*Condition*	**[THEN]**	; More interrogations, multiple is optional
	Instructions		
ELSE			; Last branch, optional
	Instructions		
ENDI			; End of IF structure
LOOP			; Endless loop
	Instructions		; Exit LOOP only by means of
			; EXIT, EXITIF, JMP, RET, Interrupt or SX reset
ENDL			; End of LOOP
WHILE	*Condition*		; Interrogation at start of loop
	Instructions		
ENDW			; End of WHILE loop
REPEAT			; Start of loop
	Instructions		
UNTIL	*Condition*		; Interrogation at end of loop
FOR	*RegisterAssign*		; Loop with register initialisation
	Instructions		
ENDF			; End of FOR loop
EXIT			; Leave one structure level
EXITIF	*Condition*		; Conditional exit from one structure level
EXITIF	*Condition* **TO** *Address*		; Conditional jump from one structure level

SXp program listing 1

```
;       Square Root, SXp-Version by E. Haug, 27-JUN-1999
;       8-bit result equals INT(SQRT(x)), therefore -0.99... accuracy

;       employs CARRYX für 16-bit arithmetic!

;       device  pins28,pages4,banks8,turbo,stackx,optionx,carryx,oscxt
        device  SX28L,turbo,stackx_optionx,carryx,oscxt2      ; A-type silicon

        id      'ROOT'

        reset   start           ; Reset start address

        freq    1_000_000       ; takes <10 sec for all 64K possible SQRTs
;       freq    5_000_000       ; takes about 2 sec for all 64K possible SQRTs
;       freq    50_000_000      ; takes about 0.2 sec for 64K SQRTs, equals
                                ; approx. 3 µsec on average per SQRT @50 MHz

; Variable

                org     $10     ; all required registers in the same bank

input1  ds      1
input2          ds      1
root_mask       ds      1

lsbyte  ds      1       ; Loop counter for MSByte
msbyte  ds      1       ; Loop counter for LSByte

;***************************************************************************
                org     0       ; Program starts at address $000 (not strictly required)
start

; the following two FOR loops are for estimating runtime only

FOR msbyte = #256               ; 256 ...
        FOR lsbyte = #256       ; * 256 times, calculates all possible 64K SQRTs
                mov     w,msbyte
                mov     input2,w        ; because input2 is modified
                mov     w,lsbyte
                mov     input1,w        ; because input1 is modified
                call    sq_root ; W = INT(SQRT(input2_1))
        ENDF
ENDF

LOOP    ; Simple delay loop (smallest SXp program!)
ENDL    ; You could insert a break point here for runtime measurement


;***************************************************************************
; SQ_ROOT
; This routine calculates the square root of a 16-bit unsigned number.
; on entry:     input1 – low byte of number, destroyed afterwards
;               input2 – high byte of number, destroyed afterwards
; on exit:      W register returns 8-bit result: W = INT(SQRT(input))
;***************************************************************************

sq_root
        mov     root_mask,#$c0  ; Initialise root_mask
        mov     w,#$40          ; Initialise result
        REPEAT
                stc                             ; 16-bit arithmetic (CARRYX)!
                sub     input2,w                ; Subtract result developed so far
                IF NC                           ; Undo subtraction when carry cleared
                        add     input2,w        ; Carry not set here
                ELSE
                        or  w,root_mask         ; Set current bit
                ENDI
                LOOP                            ; Carry may be set or not at this point
                        rl      input1          ; Shift number to the left by one position
                        rl      input2
                        rr      root_mask       ; Copy MSBit from input2
                        EXITIF NOT root_mask.7; End of LOOP when MSBit not set
                        snc                     ; SKIP more effective here than IF-structure
                        retp                      ;  Done when Carry set
                        clrb      root_mask.7
                        xor       w,root_mask
                        stc                       ;  16-bit arithmetic (CARRYX)!
                        sub       input2,w
                        or        w,root_mask     ;  set current bit
                ENDL
                xor       w,root_mask
        UNTIL C                                   ;  End of loop when LSBit of root_mask
                                                  ;  shifted into Carry, else repeat
        sub       input2,w
        IF C
                IF NZ                             ;  If result <> 0 ...
                        xor       w,#1            ;  ... correct LSBbit
                ELSEIF input1.7                   ;  ... or if = 0 and MSBit set
                        xor       w,#1            ;  ... correct LSBit
                ENDI
        ENDI
        retp                                      ;  Done!

        end                                       ;  End of program
```

interrogations).

SXp employs the optional (and often wayward) DEVICE CARRYX bit.

The risk of collisions with conditional assembler statements is avoided by *Sxp*: in the *SXp* source code file, it places a question mark (?) immediately before colliding statements (which may occur in a few assemblers only, including the Parallax SX-KEY package). The ? is also automatically removed again by *SXp* without taking any further action in the relevant line, so that all subsequent statements are available to the actual code-assembly operation.

SXp statements may be written in capitals, lower-case characters or a mixture of these.

If *SXp* structures extend across SX page boundaries, the tools required to deal with such cases are available. However, these tools are not available in the demonstration version (incidentally, that is the only difference with the full version).

Example

After such a vast amount of theory to wade through we'd say it's time for an example of an *SXp* source code program. The one to be discussed computes the square root of a 16-bit number [5]. To be precise, the function is written as

SQ_ROOT = INT(SQRT(NMBR))

As a result of the 8-bit format of the output word, the rounding off performed by the integer function may produce results that are up to 0.99 too low (just think of the square root of 65,535 = 255.998…). With some extra effort, the result could be expanded into a 9-bit number, which would have a maximum error of ± 0.5.

For clarity's sake all *SXp* statements are printed in bold capitals. The SX assembler commands appear in lower-case characters (see Listing 1).

The *SXp* source code program is written with any word processor capable of producing ASCII output format. The file is saved with the extension '.SXP'. With *SXp* up and running, this *SXp* file is then compiled after a number of options have been selected (in this case, the option 'use CARRYX' is enabled although the example does not include any *SXp* statements having comparisons; it has flag interrogations only).

During the compilation process, *SXp* generates an SX assembler source code program with the extension '.SRC'. This program may also contain error reports with error pointers to line numbers of the *SXp* program. These reports allow you to quickly trace and eliminate programming errors.

After compiling the SX assembler program may be assembled using SX-KEY™ or indeed any other assembler for Scenix SX micros, provided, of course, the mnemonics and syntax rules set out in the SX datasheets [1] are fully supported. DOS assemblers should also be suitable!

The same SX assembler source code program may also be subjected to a debugging session using, for example, SX-KEY™. Optionally, all *SXp* statements may be retained as comment in the SX assembler source code file. This makes debugging much easier (for a better overview and checking the structure level is also produced). Thanks to the Windows operating system, there's no need to quit *SXp*. It may remain in memory and quietly sit in the taskbar during the entire 'design session'. In this way, you can quickly change back and forth between the text editor, the *SXp* precompiler and an assembler/debugger utility.

Code efficiency

The currently available statements of *SXp* allow programmers to generate very efficient object code for SX micros. As compared with pure assembly-level programming, less efficient code is obtained when an *SXp* statement contains just <u>one</u> single-word assembler instruction (no macro!). This occurs twice in the last 'IF' structure of the above example.

This is caused by the SX assembler allowing <u>one</u> SX single-word instruction to be jumped over using (conditional) skip instructions. These single-word commands are then not recognized as such by SXp, which substitutes inefficient jump instructions instead of a skip instruction. If, however, more than one SX single-word instruction is found within *SXp* structures, there will be no differences in respect of efficiency.

Separately compiled, the IF structure mentioned above produces the SX assembler code shown in Listing 2. To keep the overview uncluttered, it is produced without the option 'keep SXp Statements in output file'.

For the above example, a nifty SX assembler code file [5] provides the result shown in Listing 3.

Instead of the IF structure, this compact but difficult to analyse piece of assembler code could have been used in the *SXp* source code file instead of just in the IF structure. This would have resulted in the same efficiency for the *SXp* program. A single conditional SKIP instruction should not affect program legibility too much (see also the example). However, we should repeat our warning: be careful with macros!

SXp program listing 2

```
    SC
    JMP    _L1
    SNZ
    JMP    _L3
    xor    w,#1      ; ... Correction of LSBit
    JMP    _L4
_L3
    SB     input1.7
    JMP    _L5
    xor    w,#1      ; ... Correction of LSBit
_L4
_L5
_L2
_L1
```

SXp program listing 3

```
        sc                        ; As SXp
        retp                      ; Already finished here!
        snz                       ; As SXp
        snb     input1.7          ; Combined interrogation:
        xor     w,#1              ; NZ OR (Z AND input1.7)
```

SXp program listing 4

```
        REPEAT                    ; wait ...
        UNTIL Port.5              ; ... for Port Bit 5 to be set
```

SXp program listing 5

```
        ;01//  REPEAT             ; wait ...
_L1
        ;01//  UNTIL Port.5       ; ... for Port Bit 5 to be set
        SB      Port.5
        JMP     _L1
_L2
```

With pure delay loops, by the way, the REPEAT-UNTIL structure is more efficient than the WHILE-ENDW structure because an interrogation is performed at the end of a REPEAT loop, resulting in one less jump instruction — see Listing 4.

As you can see from Listing 5, the SX code generated by the precompiler is hard to improve.

Comparisons with a constant ‘0’ enables *SXp* to generate optimised code. In the above example, the FOR loops with 256 iterations (9-bit number) are executed as they should. More tips of this type, as well as descriptions of typical programming errors may be found in the *SXp* manual [3].

Conclusion

The *SXp* precompiler described in this ‘PC Topics’ article offers its user the possibility to create structured assembler-language programs for the SX microcontrollers from Scenix™. As compared with higher programming languages, a modest amount of study is required to learn programming from scratch. This is mainly by virtue of the syntax which has been kept as simple as possible, clear and easy to remember.

The code generated by *SXp* is efficient by most standards while you, the programmer, do not alienate from assembly-language programming and its typical environment, despite the use of structures.

The efficiency of the precompiler described here could no doubt be enhanced further if only multiple logic combinations (AND-OR) and quantity requests (IN, as in Pascal) would be allowed. Both were available many years ago in STRUKTA. Even better efficiency would be obtained if statements like SWITCH-CASE, ON-GOTO were implemented in *SXp*. Mind you, they are not yet implemented. Similarly, the rudimentary FOR-NEXT loop could be expanded into the more powerful FOR-TO-STEP structure.

It would be desirable for the makers of SX assemblers to integrate at least the features of *SXp* into their products. Many years ago, there were intentions to do so for some microcontroller families (for example, 68000 assemblers from Motorola, Mostek and Quelo). Today the same happens with Mitsubishi and Zilog, although the resulting products are cumbersome to use while offering fewer features than *SXp*.

Finally, some C compilers also support structured assembly-code programming in the integrated assembler. A good implementation is found with IAR, for example. In most cases, however, the price tag of these products puts them well beyond the reach of hobbyists and occasional users.

(992037-1)

References:
[1] Datasheet SX18/20/28, Scenix™, downloadable via *www.scenix.com* or obtainable from authorized Scenix distributors.
[2] SX microcontroller evaluation system (1-4), *Elektor Electronics* February through May 1999.
[3] SXp — structured assembler programming for the Scenix™ SX, by E. Haug. Download *SXp* DEMO version and manual via *www.scantec.de*
[4] The various Page functions are not implemented in the demo version of *SXp*!
[5] Exact source of original program not known.

Most old computers just collect dust in the attic after being retired from active service. That's a pity, since there are still useful tasks that could be performed by these relics. In this article, we describe a very simple way to add an optical output to the CD-ROM drive of a cast-off PC. Such an output could for example be used to connect a MiniDisc recorder.

Design by K. Schlott

optical output for CD-ROM drive

a new life for old PCs

Regardless of make, model and price, there is one thing that all computers have in common: they start to age while they are still in the shop. The first wrinkles, which appear after two to three years, can be smoothed out with an update, but a couple of years after that there is not much more that can be done. Then we buy a stylish new model, and the old workhorse lands for good in the attic. Usually, some thought is given to selling the old computer, but if it turns out that little is to be gained by this, most of us would rather hold on to it as a back-up. As a rule, this back-up function is never actually used, and the old beast just sits there for years doing nothing.

This is truly a pity, since an old 386 or 486 PC can still be used quite well for a lot of things. For example, it could be used as the 'brain' of a model train control system, as an advanced roll-down shutter controller or a perfect thermostat for the central heating. The possibilities are almost endless!

If you don't have any particular application for the entire PC, you can still consider looking for new uses for its components. The CD-ROM drive, for example, is in principle ideally suited for playing back to a MiniDisc recorder

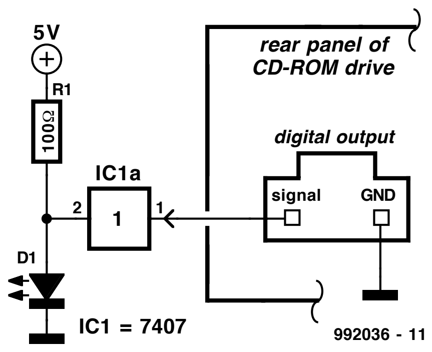


Figure 1. A 'standard' digital output can be converted into an optical output using only three components. The diameter of the LED (2.2 mm) is matched to fiber optic guide. The LED is order code 18 44 11-55 from Conrad Electronic.

Digital out, optical in

The only thing that stands in the way of a successful marriage between a CD-ROM drive and a MiniDisc recorder is that the former has only an electrical digital output, while the latter has only an optical digital input. However, this should not be an insurmountable problem for any self-respecting electronic technician. A quick examination of the digital output of a CDROM drive shows that it provides a trapezoidal biphase signal, with an amplitude of 5 V. With a suitable converter, this can easily be transformed into optical ones and zeros. The well-known TOSLINK modules from Toshiba are made to measure for this task. Still, it's more of a challenge to see if there's a simpler way to implement this conversion — perhaps with a few components that we just happen to have lying around. And in fact, this turns out to be possible. Fifteen minutes of tinkering was all it took to add a perfectly functional optical output to our sample CD-ROM drive, and only three components were needed: a standard TTL buffer, a resistor and a LED. Figure 1 shows how simple it all is.

Next comes the cable. Special optical interconnecting cables are not only fairly expensive, they are also terminated in connectors that only mate with matching components. For a lot less money, you can buy fibre-optic cable by the roll (from Conrad Electronic, for example). If a light-diode is used for D1, the fibre-optic cable can be inserted directly into the hole in the LED package that is provided for this purpose. All that then remains is the question of how to insert the cable into the optical input of the MiniDisc recorder. The cable fits into the opening easily enough — that's no problem — but somehow the signal from the cable seems not to reach the internal circuitry of the recorder. Very strange. After a thorough examination of the optical input connector, we find the culprit: a pesky little miniature switch, built into the connector, that controls the switchover between the analogue and digital inputs. Our cable is just a bit too thin to actuate this switch. The solution to this problem is easy: after a short piece of heat–shrink tubing is fitted over the end of the cable, everything works perfectly!

Remarks

Just to avoid any misunderstanding, it should be noted that the CD-ROM drive need not necessarily be located in an old PC. If you only have a new PC and you want to use its CD-ROM drive as a master for making recordings, you can easily build the circuit described here into the PC. If you fit the LED to on the front panel so that it is readily accessible, you will have no problem making an optical connection to other devices.

(992036–1)

Many computers and microcontrollers are programmed in hexadecimal, which represents numerical values using the numerals 0 through 9 plus the letters A through F. Entering the numerals is quite easy with the numeric keypad of a PC keyboard, but the letters are scattered over the entire alphanumeric portion of the keyboard. This article describes the construction of a keypad with 20 buttons, which includes all the necessary characters as well as several control characters.

Design by U. Reiser

hexadecimal keypad for PCs

problem–free programming

The hexadecimal keypad is connected between the actual PC and its keyboard. It also provides the necessary extension for the keyboard cable. The following functions are provided by this keypad:

- generating the lower–case letters a through f;
- generating the upper–case letters A through F when the Shift or Caps Lock key of the PC keyboard is activated. Activating the # key on the hex keypad will prevent characters from being output by the numeric/shifted character keys located in the alphanumeric portion of the PC keyboard);
- generating the numerals 0 through 9, corresponding to the numeric keys of the alphanumeric portion of the PC keyboard;
- generating the numerals 0 through 9, corresponding to the keys of the PC keyboard numeric keypad, when the # key of the hex keypad and the NUM key of the PC keyboard are activated;
- generating cursor control characters corresponding to the keys of the PC numeric keypad, if the # key of the

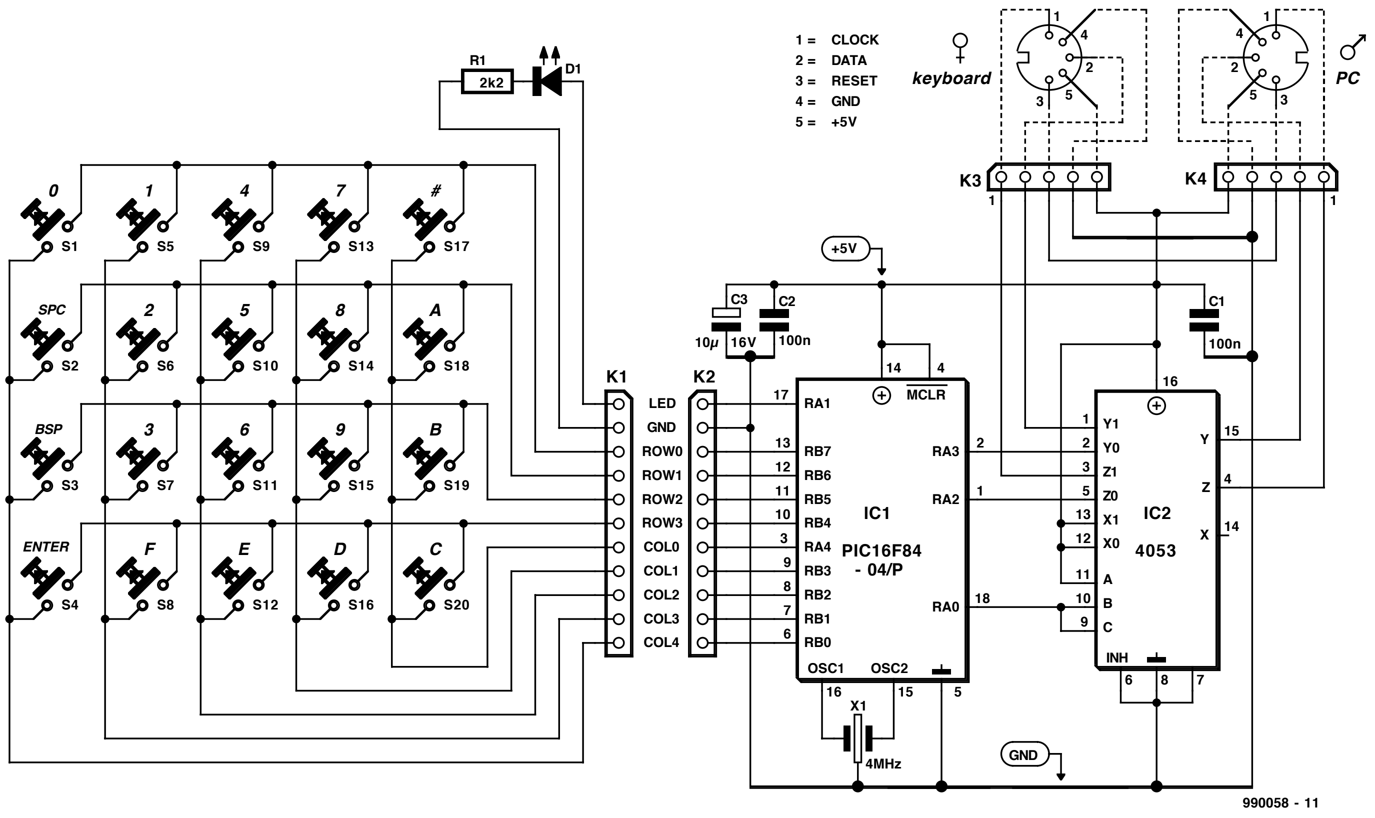


Figure 1. Circuit diagram of the hexadecimal keypad.

hex keypad is activated and the NUM key of the PC keyboard is deactivated;

- generating the Space, Backspace and Enter characters (in all modes).

- In addition, it is possible to utilize the control functions of the numeric keypad. This requires that the NUM key of the PC keyboard is not active, which is indicated by the NUM LOCK LED of the keyboard not being illuminated.

The schematic diagram of the hex keypad, as shown in **Figure 1** shows that only a few components are needed in addition to the pushbutton switches. The operation of the hardware is easily explained. The PIC controller (16F84) constantly polls the 20 switches and sends the code corresponding to an activated switch key to the PC. The second IC, a 4053, is a bidirectional multiplexer that connects either the PC keyboard or the hex keypad to the PC. The keypad is connected to the PC only when its controller has detected a keypress, and the multiplexer immediately reconnects the keyboard to the PC after the key code has been transmitted by the hex keypad. It is at least theoretically possible for the hex keypad to interrupt signals being sent from the PC keyboard. The PC supplies the 5 V operating voltage for the hex keypad (and also for the PC keyboard).

Software

You can see how the program works from **Figure 2**, as well as from the assembly-language listing on the *Elektor Electronics* Readers Services diskette (order code **996021-1**). In the first section of the program, labelled Initialisation, the input and output functions of ports A and B are set up and the internal pull–up resistors for port B are enabled. The subsequent power–on self test flashes the LED of the hex keypad twice. The on and off intervals are both set to 109 ms by TIME4. This function is intended to indicate to the user that the basic PIC processor is operational.

The main program of the controller starts with polling the key matrix (S1 through S20). This is performed by setting one of the five columns from High to Low (COL??) and waiting for 5 ms (BOUNCE), after which all four rows are polled for the presence of a Low level passed through by any activated switch. Based on the arrangement of the switches in rows and columns, the switch number is selected and the program branches to the appropriate subroutine (SW??). There it waits 5 ms, using

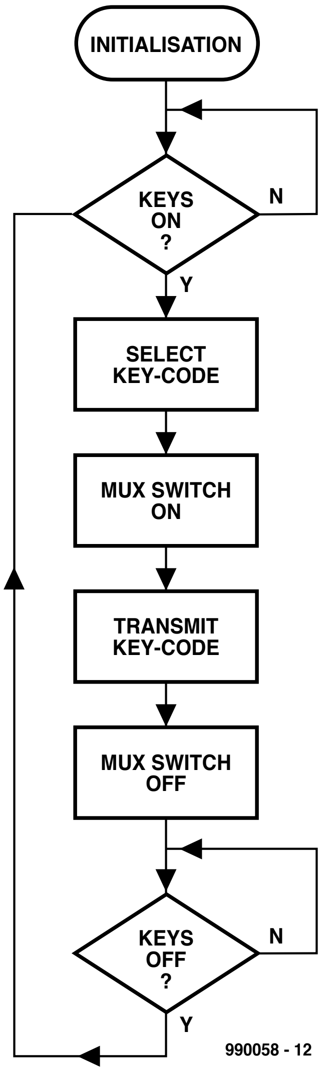


Figure 2. Software flow chart.

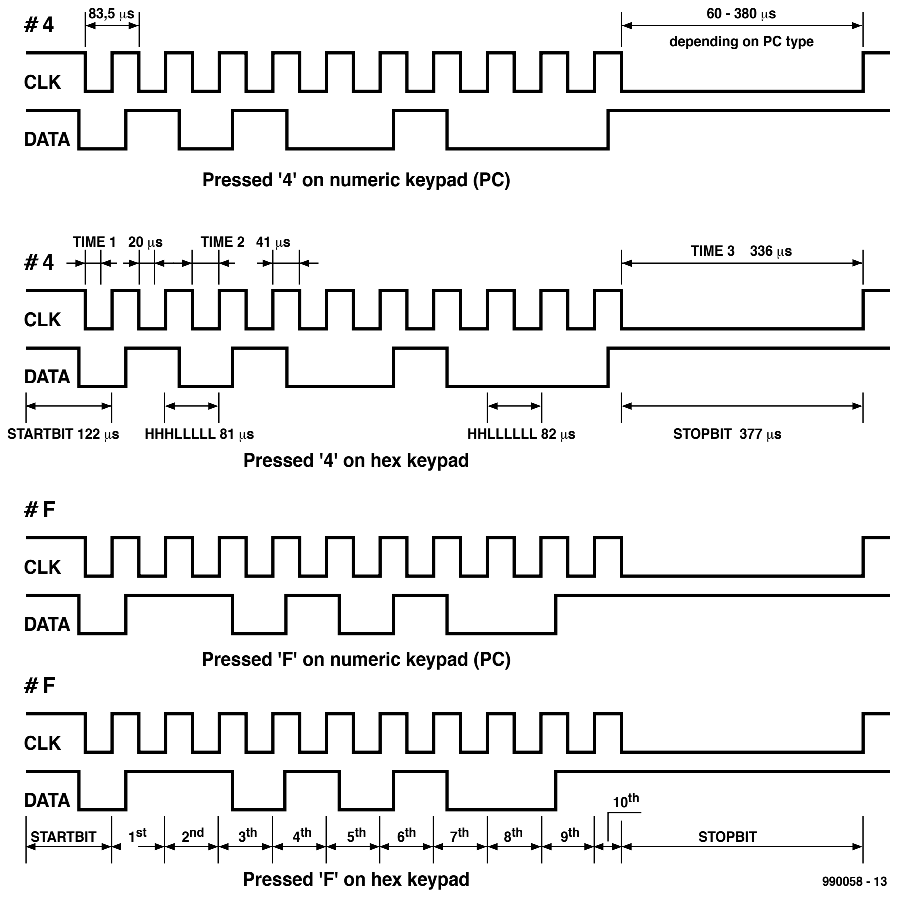
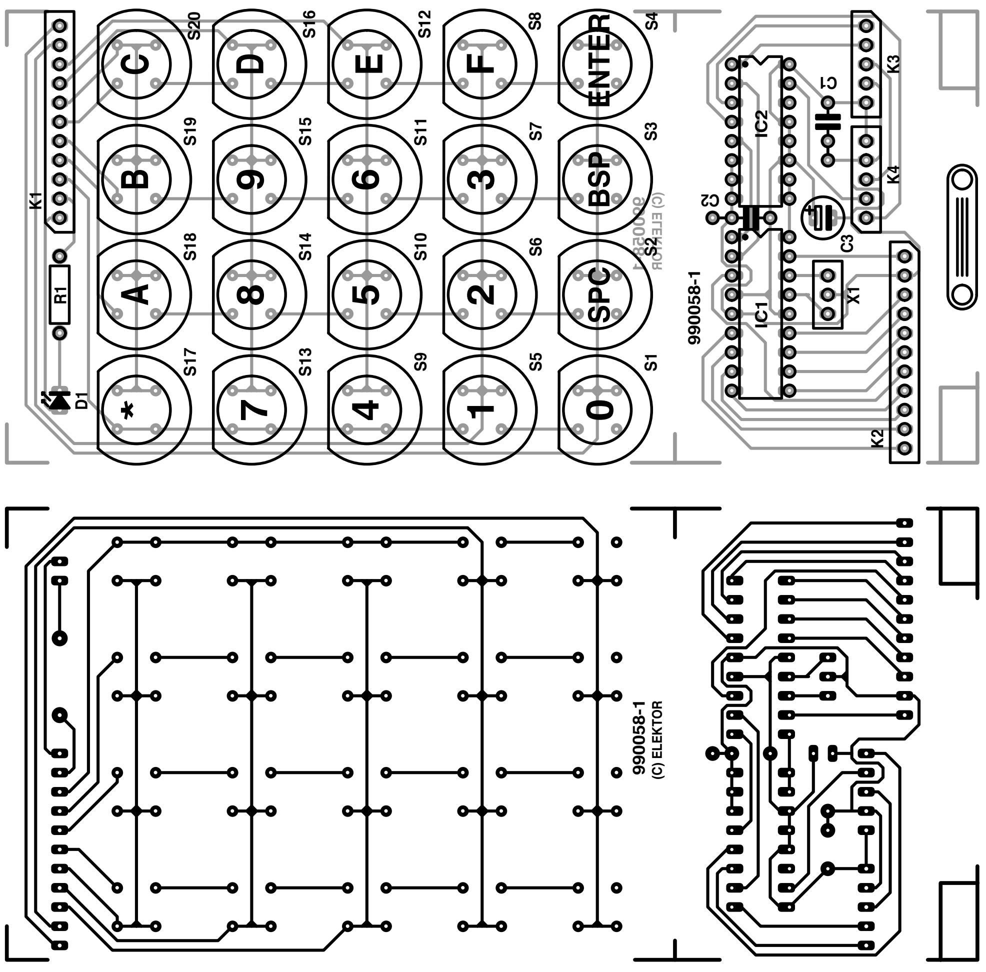


Figure 3. Key code pulse sequences.

CALL BOUNCE, and then again polls the already detected switch. If the result is negative, an error is assumed (due to keybounce, for example). GOTO COLUMN? repeats the polling of the same column. If on the other hand the second polling confirms the first result, the LED is polled (in the case of the numeric keys) and CALL RELAISON (call multiplexer-on) sets the B and C control inputs of the 4053 to Low, which causes the data and clock signals of the hex keypad to be connected to the PC. In order to assure reliable switching of IC2, program execution is delayed for an additional 5 ms by again calling the timer loop BOUNCE.

The bit sequence for the key code is generated by calling twelve subroutines. Following this, the instruction BSF PORTA,REL causes the 4053 multiplexer to be switched back to its prior state. To avoid repeated key code generation for a pressed key, the routine GOTO KEYSOFF waits for all keys to be released. This is checked by setting all columns to Low and polling all four rows in turn. Searching for a new key press is

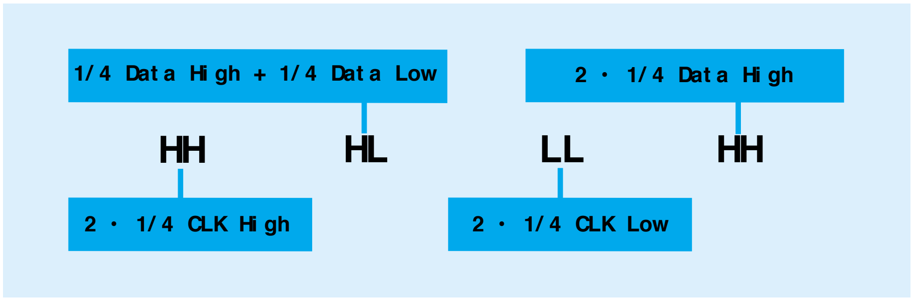
Figure 4. Printed circuit board layout and component mounting diagram.



initiated by the instruction GOTO COLUMN4 only after no Low level is detected.

Key codes

Figure 3 shows two pulse sequences that represent the signals on the clock and data lines for the keys 4 and F. The clock frequency is approximately 12 kHz. Level changes on the data line, when required, occur halfway through the High interval of the clock signal. Since the beginning and end of a pulse train are always the same, their generation is relegated to the separate subroutines STARTBIT and STOPBIT. The data and clock signals are generated by setting and clearing RA2 and RA3 as appropriate. The pulse length is determined by the two timing loops TIME1 and TIME2. If a change in the level of the data signal is required (HHHL and HHLH), TIME1 generates an interval of 20 μs, which is a quarter of the clock interval. After this, the level changes, and a second quarter–clock interval completes the High portion of the clock signal. If no level change occurs (HHLL and HHHH), the High portion of the signal (half–clock interval) is completely produced by TIME2 (41 μs). Since a level change never occurs during the Low portion of the clock signal, this portion is always generated by TIME2. Altogether, this procedure results in a pulse period of 81 to 82 ms, which corresponds to a frequency of 12.345 to 12.195 kHz. Since the pulse sequences for the individual keys are made up of several, partly identical subsequences, these are implemented as subroutines (HHLLLLLL, HHHLLLHH and so on). The meaning of the arrangement of the letters H and L in the names of the subroutines is as follows:



COMPONENTS LIST

Resistor:
R1 = 390Ω

Capacitor:
C1 = 100nF

Semiconductors:
IC1 = PIC16F84 (ready-programmed, order code 996516-1)
IC2 = 4053
D1 = LED, red

Miscellaneous:
S1-S20 = presskey type D6-R-RD (Conrad Electronics # 70 84 88)
K1,K2 = 22-way pinheader
K3,K4 = 10-way SIL pinheader
5-way DIN plug (or mini-DIN) for chassis mounting
5-way DIN socket (or mini-DIN) for chassis mounting
Enclosure type Pult-box 25 x, 109x70x50mm (Conrad Electronics # 52 24 49)
Strain relief grommet (Conrad Electronics # 52 46 20)
2 PCB standoffs, M3x10mm
Flatcable, 11-way, length approx. 10 cm
PCB, order code **990058-1**
Disk, source code file, order code **996021-1**.

Hardware

The dimensions of the circuit boards (see **Figure 4** and the layout of the components) are designed to fit the enclosure recommended in the list of components. To best utilize the available space, the keypad portion (with the LED and series resistor) and the processor portion (with the PIC and the multiplexer) are mounted on separate printed circuit boards. These are connected by an 11-lead flat cable via connectors K1 and K2. You should use the switches recommended in the list of components, since they internally connect certain circuit board tracks. If other types of switches are used, it may be necessary to fit wire jumpers so that every switch connects two sets of contacts when it is activated, as is already the case with S9, S13 and S17.

The two circuit boards are fitted one on top of the other using 10-mm standoffs. The connections to the PC and the keyboard are provided by K3/K4 and a DIN or mini-DIN plug and jack set. The two 5-lead cables are attached to the processor circuit board by strain–relief clamps.

Make openings for the 20 switches and the LED in the enclosure, and then fit the circuit-board sandwich into the enclosure. Now cut two strips of circuit board material (without copper overlay) 65 mm long and 9 to 10 mm wide (depending on the type of switch used). Fit one of these into the guide slots of the enclosure at the level of switches S1 through S4, and the other one over the processor circuit board (both crossways). These prevent the switches from being pushed down into the enclosure when they are pressed. On the front side of the enclosure, cut a rectangular hole to allow the two cables to pass to the outside.

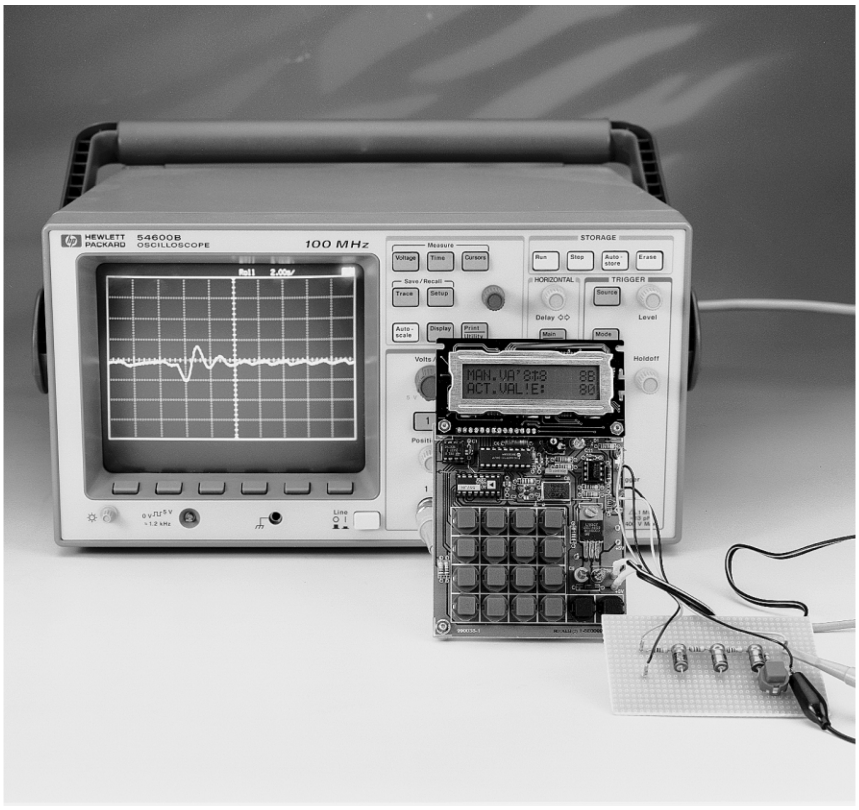
Assuming that all the components have been properly fitted, it is not necessary to specifically test the operation of the keypad. Since the keypad draws its power from the PC, you should try it out with a different power source before putting it into service, just to be safe. The LED should flash twice after the 5 V supply voltage is connected, to indicate that the PIC is operational. Switching the LED on and off with the # key can also be used as a simple functional test. If a switch is permanently activated, possibly due to its being jammed by the enclosure or incorrectly fitted, the circuit will not be able to do anything except to flash the LED twice! You can use an oscilloscope or logic probe to check the signals on the data and control outputs. When a key is pressed, pulse sequences similar to those already shown should be seen on the clock and data outputs, and a single pulse on the control output (RA1).

(990058–1)

digital PID controller

closing the loop with a PIC

The term 'PID controller' normally brings to mind an analogue controller circuit with a couple of opamps. This article shows how such a controller can be implemented in digital form. In this universal controller circuit, based on a PIC and a D/A converter, all control parameters can be entered via a keyboard to adapt the controller to the task at hand.



The digital PID controller described in this article, based on a PIC microcontroller, is on the one hand very well suited to studying how the controller operates with various parameter values, using simple control loops. On the other hand, it can of course also be used for specific control tasks. A complete control loop requires an appropriate actuator and a suitable sensor, in addition to the controller itself.

The core of the circuit is an inexpensive PIC 16C71 microcontroller, whose built-in A/D converter digitizes the measured value (actual value) and provides the result to the controller program. The program determines the control error relative to the set value (target value) for each actual value and, based on this, calculates the control value that is output to the external D/A converter. The control parameters, which are entered by a keypad, are stored in a non-volatile memory (EEPROM).

In order to manage with the program memory of the PIC, which is only 1 kB, the parameters are entered as hexadecimal codes. They are however displayed on the LCD as text, which is easier to understand.

Specifications:

➤ *controller type:*	*PID*
➤ *parameter entry:*	*hexadecimal keypad*
➤ *parameter display:*	*alphanumeric LCD*
➤ *adjustable parameters:*	*- set value*
	- initial control quantity value
	- upper limit
	- lower limit
	- sampling interval (100 ms to 25 s)
➤ *parameter storage:*	*EEPROM*
➤ *A/D and D/A conversion:*	*8 bit*
	(measured value, control quantity)
➤ *microcontroller:*	*PIC 16C71*

Design By D. Kohtz

Basic theory

Control circuits are encountered very often in electronics. For instance, a voltage regulator is present in almost every circuit. More elaborate controllers for dealing with other physical quantities, such as rotational speed, velocity, temperature, pressure, flow and so on. Control technology deals with finding solutions for such problems. Let's look at the basic principles of this technology.

In control theory, the controlled activity is referred to as the **process**. The job of the controller is to maintain the value of the **controlled quantity** at a set-point or **target value**, in spite of the effects of external *interference*. In order to do this, the controller must measure an **actual value y**, calculate the **control error e** by comparing the actual value to the **target value w**, and output a *control value* **u** that depends on the value of the control error and which achieves the desired effect. (Note: the official term for the control value is 'manipulated variable'.)

A characteristic feature of every control system is **closed loop operation**, as illustrated in **Figure 1**. It is necessary that a given input signal produces a specific output signal, for both the controller and the process. In general, the relationship between the input an output signals is *time-dependent*.

The step response is an essential characteristic of both processes and controllers. It reflects how the output signal responds when the input signal **u** changes instantaneously from one level to another level. The waveform shown in **Figure 2** illustrates a step response that is typical of many processes. For example, if a given DC voltage is applied to a resistive heating element, the graph of its temperature as a function of time will be similar to this curve.

A controller is also a transfer element that produces a control value (output signal) in response to a control error (input signal). Depending on the way in which the controller responds, it can be classified as a proportional, integral or derivative controller. In a proportional controller (*P* controller), the transfer element is in principle nothing more than an amplifier. Its output signal changes in proportion to its input signal, with the gain being an adjustable parameter. As the names suggest, the transfer element of an integral controller (*I* controller) is an integrator, and the transfer element of a derivative controller (*D* controller) is a differentiator. With the latter types of controllers, both the time constant and the gain are adjustable parameters.

A controller that combines all three types of response, called a PID controller, has proven its worth as a sort of 'universal' controller. As shown in **Figure 3**, the control value consists of the sum of the outputs of the three types of controllers, which can be assigned individual weighting factors according to the desired controller response.

1

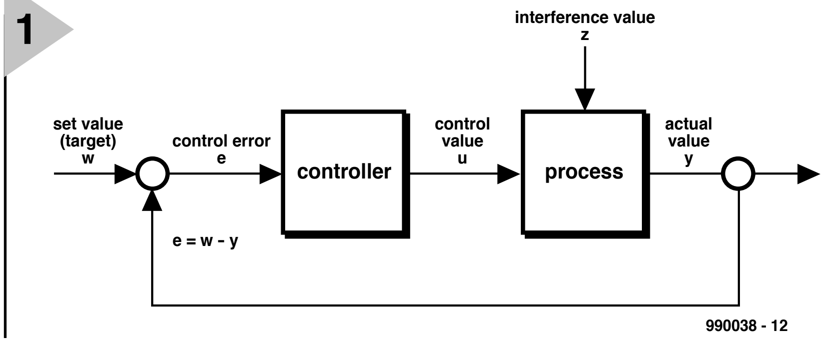


Figure 1. A closed loop is applied to control a process.

The mathematical relationship between the control error and the control value is called the **control algorithm**. For a PID controller, this is a differential equation.

The primary difference between a digital controller and an analogue controller is that with a digital controller the actual value is not measured continuously, but is instead periodically sampled at some fixed interval T_0. In the time between two successive samples, the control error must be determined and the control value calculated using the control algorithm.

The following equation can be used as the control algorithm of a PID controller (see the sidebar 'Digital PID controller design' for the derivation of this equation):

$$u_k = u_{(k-1)} + q_0e_k + q_1e_{(k-1)} + q_2e_{(k-2)}$$

In this equation, the index k represents the running count of the individual measurements. This means that $e_{(k-1)}$ is the control error for the previous measurement. The control parameters q_0 through q_2 can be approximately determined from the characteristic parameters of the process step response using the following formulas:

$$q_0 = [1.5T_G/(K_pT_u)](1 + T_u/2T_0)$$

$$q_1 = [1.5T_G/(K_pT_u)](T_0/2T_u - T_u/T_0 - 1)$$

$$q_2 = 3T_G/4K_pT_0$$

The step response of the process must be measured experimentally to obtain these values. The parameters must also satisfy the following conditions:

$$q_0 > 0$$
$$q_1 < 0, |q_1| > q_0$$
$$|q_0 + q_1| < q_2 < q_0$$

As a guideline for the value of the sampling interval T_0, it should chosen to be around one tenth of the time required for the process step response to reach 95% of its final value.

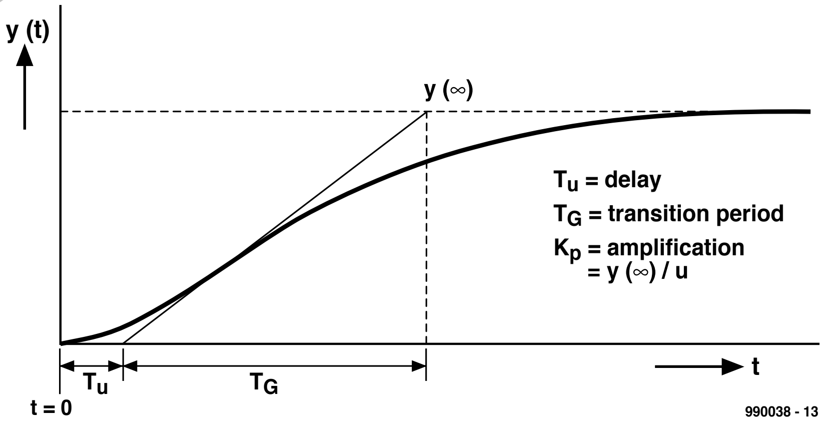
Hardware and Software

Figure 4 shows the circuit diagram of the controller. Its input is designed for voltages between 0 and 5 V, which can come from any desired type of sensor. The input voltage is passed to the internal A/D converter of the PIC 16C71 via opamp IC5a, which acts as a buffer. The low-pass filter formed by R8 and C8 at the output of the opamp keeps high-frequency interference from reaching the A/D input (RA0 of the PIC).

The controller output provides a

2

Figure 2. Example of a jump reply.



3

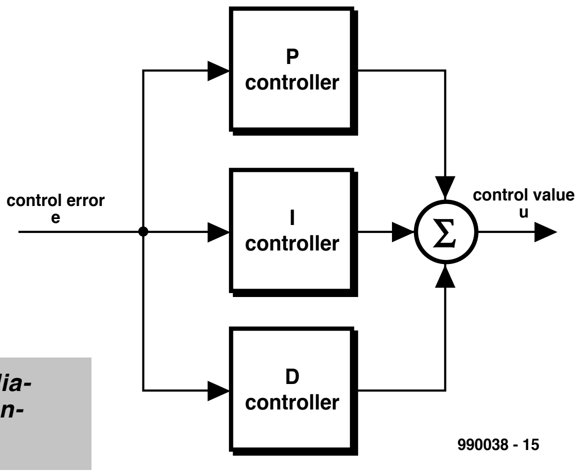


Figure 3. Block diagram of a PID controller.

voltage that also ranges from 0 to 5 V. The output current of opamp IC5b is certainly adequate for use with the previously-mentioned circuit that simulates the process to be controlled. For controlling a real process, you will naturally have to provide a suitable power stage that can be driven by the maximum load current (5 mA) of the controller output.

IC5b matches the voltage range of the D/A converter to that of the A/D converter of the PIC. Its gain is set to 1.935 by R10 and R11.

An 8-bit DAC IC (Analog Devices AD577) is used for the D/A converter. It can be operated from a single 5 V supply and does not need an external reference. Its external circuitry is limited to C7, which is necessary to decouple the supply voltage. In addition, pull-up resistors R2 and R3 are connected to the clock and data pins of the EEPROM IC (24C01).

Furthermore, only a small amount of extra hardware is needed around the PIC itself. In addition to the clock circuitry with a 4 MHz crystal, there is a reset circuit which requires only a few passive components. The power-on reset is provided by R5 and C5, while a manual reset is provided by R6 and S1.

The alarm LED D1 is driven directly from the RA4 port output, and the D/A converter chip and the LCD module are also connected directly to the output port pins of the PIC. In the input mode, the same port pins are used for scanning the contacts of the parameter-input switches. Except for S2, these are arranged as a 4×4 matrix, with four port connections on each side. S2 is used as the SET pushbutton for confirming input values; it connects RB0 to earth when actuated.

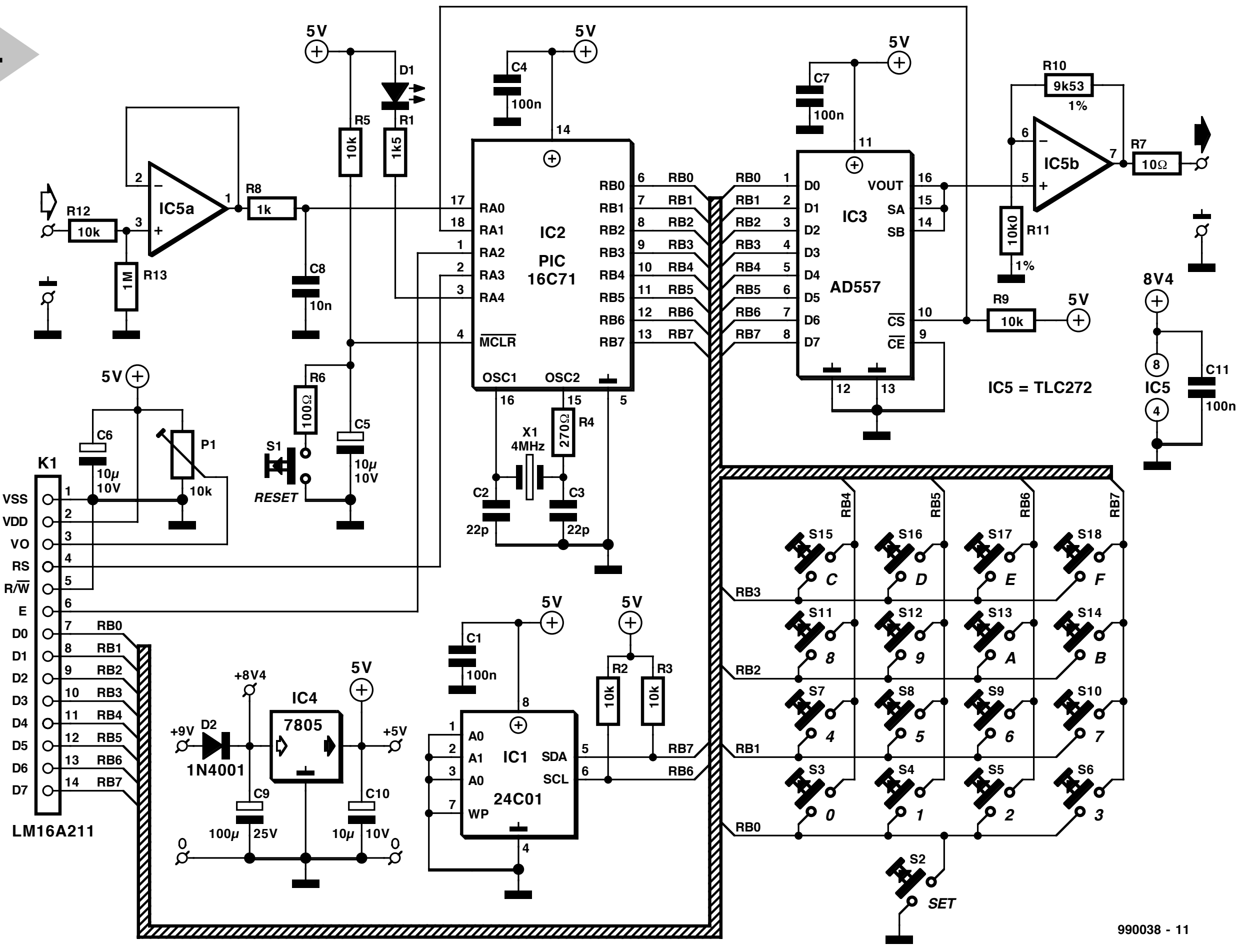
The power supply is just as simple as the rest of the circuit. It consists of a 5 V regulator, two electrolytic capacitors and a reverse-polarity protection diode at the input connector for an external 9 V mains adapter.

As far as the software is concerned, the tables for the predefined display text and the instructions needed to control the serial EEPROM both take up quite a bit of storage space. The mathematical routines for calculating the control quantities also take up a certain amount of space. The actual controller program is modest by comparison.

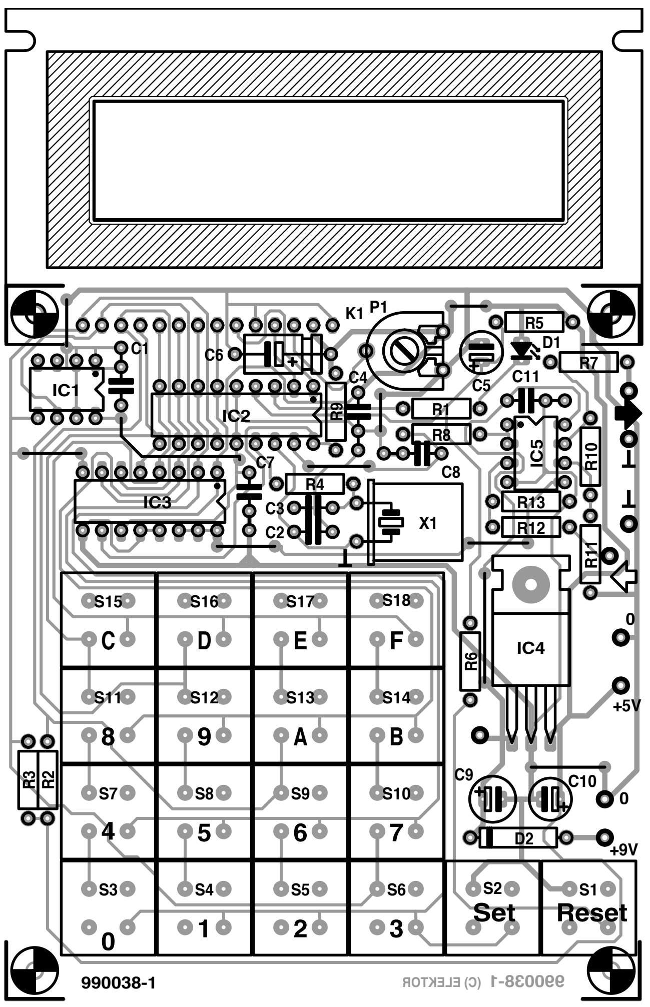
The program is protected by a watchdog timer, which in case of a failure triggers a reset and an alarm (blinking LED D1) and also sets the control quantity to zero. If this happens, the controller must be restarted, but it is not necessary to re-enter the control

Figure 4. Circuit diagram of the PIC-based digital PID controller.

4



5



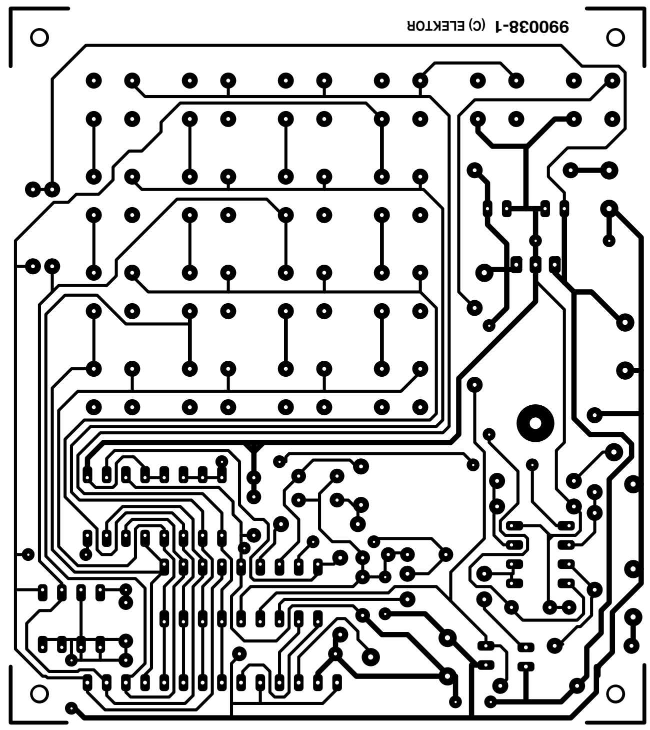


Figure 5. PCB track layout and component mounting plan (board available ready-made).

COMPONENTS LIST

Resistors:
R1 = 1kΩ5
R2,R3,R5,R9,R12 = 10kΩ
R4 = 270Ω
R6 = 100Ω
R7 = 10Ω
R8 = 1kΩ
R10 = 9kΩ53 1%
R11 = 10kΩ 1%
R13 = 1MΩ

Capacitors:
C1,C4,C7,C11 = 100nF ceramic
C2,C3 = 22pF ceramic
C5,C10 = 10μF radial
C6 = 10μF 10V
C8 = 10nF MKT (Siemens)
C9 = 100μF 26V radial

Semiconductors:
D1 = LED, red, high efficiency
D2 = 1N4001
IC1 = 24C01
IC2 = PIC16C71, order code **996504-1**
IC3 = AD557JN (Analog Devices)
IC4 = 7805
IC5 = TL272CP

Miscellaneous:
X1 = 4MHz quartz crystal
K1 = LM16A211 LCD module,2 x 16 characters (Sharp)
S1-S18 = pushbutton, PCB mount, ITC type D6-R-RD
PCB, order code **990038-1**
Disk, contains PIC source code, order code **996003-1**

parameters, since the values that were previously stored in the EEPROM can be used again.

Since the program needs a very large number of registers, some registers must be used for more than one purpose. In principle, this is not a problem, since (for example) the registers used for performing calculations are never used during parameter input. The only disadvantage is that the registers that are used more than once do not have meaningful names.

If you want to carefully study or modify the program, you will find the source code on a diskette available through our Readers Services.

Construction and operation

Constructing the circuit on the printed circuit board does not require any unusual skills. Since the circuit board is single-sided, a few jumpers are unavoidable. Do not overlook them when stuffing the board.

The keypad module specified in the list of components must be used for the parameter input pushbuttons, due to the internal connections. An unregulated 9 V mains adapter can be used for the power supply, as long as it can supply at least 200 mA.

When setting up the board, first adjust P1 to set the contrast of the display for optimum readability.

A simple circuit, shown in **Figure 7**, has been designed for testing the controller and experimenting with it. This circuit simulates the process to be controlled. The step response of the process is determined by the time constants of three RC low-pass circuits

6

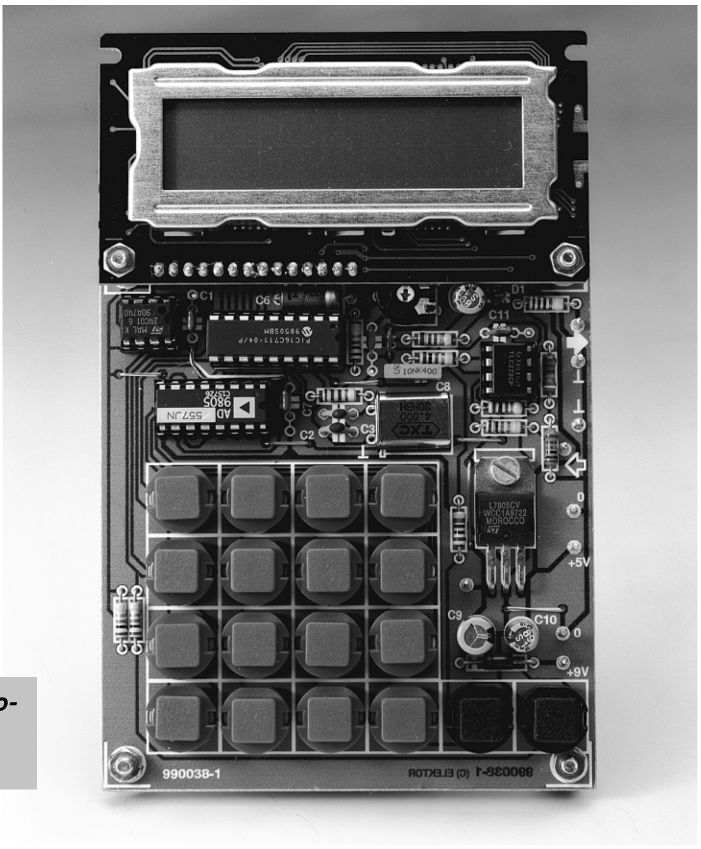


Figure 6. Finished prototype of the digital controller.

connected in series. The time constants of this 'electronic process' can thus be easily changed, and interference can be easily simulated by loading the output with the switchable resistor. To use the simulation circuit, connect its input to the output of the controller and its output to the input of the controller.

Control parameter values can be entered only immediately after the circuit is switched on or directly following a reset. They are entered in hexadecimal form by means of the keypad. Two characters will be displayed after two keys have been pressed in turn. The SET button must be pressed for the entered value to be accepted and stored in EEPROM. The target value and the initial value of the control quantity can be set between 00_H and FF_H, which corresponds to a voltage range of 0 to 5 V. The upper and lower limit values can also be set between 00_H and FF_H. The length of the sampling interval (display indication 'INTERVAL') is determined by multiplying a basic increment of 0.1 s (fixed in software) by the entered hexadecimal value.

The parameters q_0, q_1 and q_2, which are important for the control algorithm, are entered as fixed-point numbers, with one byte (two hex characters) before the decimal point and one byte after the decimal point. The bits after the decimal point are weighted according to $1/2^n$, where n represents the bit position (MSB = 1, LSB = 8). A hex value of 80 after the decimal point thus corresponds to 1/2 (0.5 decimal). Since q_1 is always negative, a short explanation of the representation of negative binary numbers is in order. A negative binary number is formed by bitwise inverting a positive number and adding one to the result. An example is:

13_D =	$0000\ 1101_B = 0D_H$
inverted:	$1111\ 0010_B = F2_H$
+ 1:	$1111\ 0011_B = F3_H$

For 8-bit signed numbers, this results in a range of values from $+127_D$ ($7F_H$) to -128_D (80_H). Negative 8-bit numbers can be recognized by the fact that the MSB is a 1.

The integer and fractional bytes of q_0 through q_2 must be entered separately (for example, Q0HI and Q0LO are entered by pressing two keys for each byte, with SET pressed between the first and second pair of value-entry keys).

If you use the simulation circuit shown in **Figure 7**, you can achieve satisfactory results with the following parameter values:

q_0 = $13.5_D = 0D80_H$
q_1 = $-18.5_D = ED80_H$
q_2 = $6.5_D = 0680_H$
sampling interval = 0.4 s (enter '04')
target value = 80_H (approx. 2.5 V)
initial control value = $A0_H$ (approx. 3.1 V)

Since the electrolytic capacitors in the simulation circuit have quite large tolerances, it is possible that the controller will not operate optimally with the above values. In this case the values of q_0 through q_2 should be empirically adjusted.

The output signal of the controller is set to 0 V during parameter entry. After parameter entry has been completed by pressing SET, the control value (CONTROL VAL) is output at a level equal to its entered initial value until the entered target value (SET-VALUE) is reached. After this, the actual control process starts up and, depending on the selected control parameter values, more or less exactly maintains the controlled variable at the set value, even in the presence of interference. During the control process, the current levels of the control value (CONTROL VAL) and the actual value (ACT.VAL) are continuously displayed.

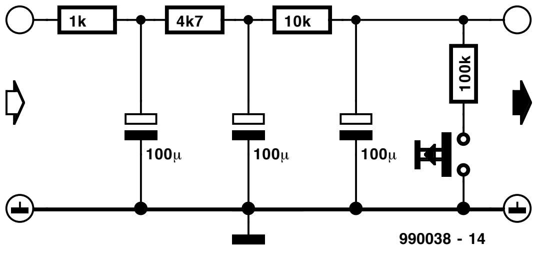
If either the upper or lower limit value (UPPER LIMIT or LOWER LIMIT) is exceeded, the alarm lamp (LED D1) blinks for the duration of out-of-limits condition. No new values are displayed during this interval. The controller output goes to zero if the upper limit is exceeded, and to FF_H (5 V) if the lower limit is exceeded.

If the controller is started anew by pressing Reset, the parameter entry process must be run through again by pressing SET after each parameter value is displayed. SETVALUE appears first on the display, with the last saved value. The values of CONTROL VAL, UPPER LIMIT, LOWER LIMIT and INTERVAL follow in turn, each appearing after SET has been pressed. The stored values are displayed for each parameter in turn, and can be modified if necessary.

(990038-1)

Figure 7. This little circuit simulates a 'process' for testing the controller.

7



Digital PID controller design

The easiest approach is to start with an analogue controller designed according to the block diagram shown in Figure 3, whose control algorithm is given by the following differential equation:

$$u(t) = K_{PR} \cdot e(t) + K_{DR} \cdot de(t)/dt + K_{IR} \cdot \int_0^t e(\tau)d\tau$$

In this equation, K_{PR}, K_{DR} and K_{IR} represent the (adjustable) gains of the three controllers (P, D and I controllers) shown in Figure 3. This equation can also be represented in the frequency domain by applying the Laplace transform, as follows:

$$u(s) = K_{PR} \cdot e(s) + K_{DR} \cdot s \cdot e(s) + (K_{ID}/s)e(s)$$

where it is assumed that the initial conditions are all zero.
The gains of the three parts of the controller (often referred to as the proportional, integral and derivative coefficients of the controller) must be set by the designer according to the desired control response.
The next step is to find a discrete equivalent for the controller described by equations 1 and 2. The backwards difference is defined as the discrete-time equivalent of the continuous-time derivative of a function. It is given by the formula

$$\Delta f(t) = \left[f(t) - f(t-T)\right]/T$$

in which T is the period of the sampling signal (the sampling interval).
The finite sum is defined as the discrete-time equivalent of the continuous-time integral of a function. It is given by the formula

$$\int_a^b f(\tau) = T\left[f(a+T) + f(a+2T) + \ldots + f(b)\right]$$

If the relationships shown in formulas 3 and 4 are applied to the first two equations, the result is the following equation, which is shown in the article as the control algorithm of a digital PID controller:

$$u_k = u_{(k-1)} + q_0 \cdot e_k + q_1 \cdot e_{(k-1)} + q_2 \cdot e_{(k-2)}$$

Once again, the index k is the running count of the individual measurements. This means that $e_{(k-2)}$ is the control error for two sample intervals prior to the present measurement.

WHEN ELECTRONICS WAS YOUNG (7)

With the passing of the years, developments and inventions in the electrical field succeeded each other in increasing tempo. In 1876, Alexander Graham Bell (1847–1922) developed the telephone. The sound to be transmitted via the telephone line was picked up by a funnel-shaped horn. The horn was terminated by a thin metal diaphragm immediately behind which was a small coil wound on a bar magnet. When sound entered the horn, the diaphragm vibrated and thereby varied the distance between the diaphragm and the coil. This caused a voltage to be generated in the coil that alternated in rhythm with the sound signal. When two such devices were linked by a long wire, the speaker at one end could bring the diaphragm at the other end into vibration. The same device was thus used for speaking and listening by alternately bringing it in front of the mouth and against the ear. This setup was improved fairly quickly and provided, among others, with a different kind of earpiece. Within a few years – in 1878 – the first public telephone network was opened in the United States.

Alexander Graham Bell (1847...1922)

Within a year after the introduction of the telephone, Thomas Alva Edison (1847-1931) had devised a contraption that consisted of a wooden cylinder around which was wound a sheet of tinfoil, an arm that enabled a needle to be brought into contact with the tinfoil, and a crank to rotate the cylinder. Edison set the needle in contacts with the tinfoil, rotated the cylinder with the aid of the crank, and into a horn-shaped mouthpiece attached to the needle he sang (or shouted): "Mary had a little lamb …". When he turned the cylinder again, the horn of the instrument produced a recognizable reproduction of his voice. Thus was born the phonograph or, as it is better known in Europe, the gramophone.

Although the reproduction was not of very good quality, the principle had been established. Ten years later, Emile Berliner (1851–1929), another American inventor, produced an improved version of the phonograph, which he called 'gramophone'. In contrast to Edison's instrument, which used a variable-resistance transmitter, that of Berliner used a flat disk instead of a rotating cylinder. The flat disk is, of course, the forerunner of the modern gramophone record. In the Berliner system, the grooves on the disk are modulated laterally, whereas those on the Edison cylinder are amplitude-modulated.

In 1879, in America, Edison, and in England, the British physicist and chemist Sir Joseph Wilson Swan (1828–1914) simultaneously introduced the first practical incandescent lamp. It was, perhaps, not so much the improvement of the filament that made the incandescent lamp a success as the evacuation of the bulb (in which Swan was aided by Charles Stearn, an expert in the production of high vacua). Edison and his team perfected the filament: after examination of thousands of alternatives, they devised the cotton-thread filament which, when joined with the bulb patents of Swan and Stearn, made the Edison lamp a commercial success.

in 1890, a young Dutch mechanical engineer, Gerard Philips (1858–1942), decided to set up his own company to produce incandescent lamps. This resulted in 1891 in the Philips Gloeilampenfabriek in Eindhoven, which soon became famous all over the world and is still an important multinational company today.

[995073]

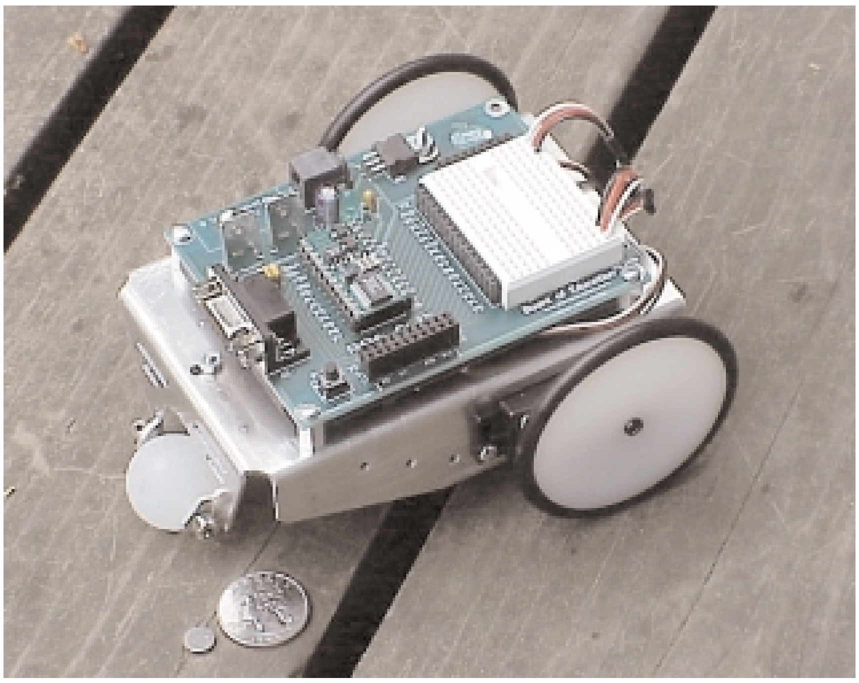
Antique phonograph

BASIC Stamp programming course (1)

Part 1: Introduction and BoE hardware

This is the first of a series of articles discussing the assembly of a simple robot, called BoE-Bot, we'll use to learn introductory programming with the BASIC Stamp. The BoE portion is the BOARD OF EDUCATION, and the Bot refers to robot chassis. The project is flexible, and could also be a rolling breadboard. Provided you've got a Stamp-based servo driven platform, following the lessons will be easy.

By Chuck Schoeffler, Ph. D. and Ken Gracey


Photograph showing author's version of BoE. Readers should note that this differs slightly from the BoE described in this article.

Robotic projects are an ideal way to learn how to interface a microcontroller to other electronic components. It's effective for students of all ages to see and hear outputs in addition to measuring them with a multimeter or oscilloscope. Teachers often say that it requires light, sound, and movement to capture a student's interest. This first column consists of assembling a BASIC Stamp II Board of Education-based robotic platform named the BoE-Bot. Once you've built this platform we'll use it to teach basic microcontroller skills. The article installments are planned as follows (subject to change):

➡ *Introduction and BoE construction*, September 1999 — The part you are reading right now. The BoE printed circuit board was designed by *Elektor Electronics* from an example supplied by the authors. Brief introduction to the BASIC Stamp II.

➡ *Building the BoE-Bot*, October 1999 – Construction process and drawings for building the BoE-Bot.

➡ *BASIC Programming*, November

1999 — Program the BoE-Bot to follow a pre-entered path using subroutines, IF...THEN, and EEPROM storage. These are the basic programming fundamentals.
- ➡ *Sensors*, December 1999 — Use a photoresistor to sense and follow light. Use a wire as a switch closure for the bumper and generate sound with a piezospeaker.
- ➡ *Infrared Control*, January 2000 – Add an infrared LED for proximity sensing and external control by handheld remote.
- ➡ *Advanced BoE-Bot Projects*, February 2000 — Schematic and source code ideas for advanced BoE-Bot projects including sonar and communication.

The BoE part of BoE-Bot is the Board of Education, a platform to hold your BS2-IC microcontroller system and prototype your sensor projects. The original board was designed by Parallax in coordination with educational customers to teach microcontroller interfacing. It is designed for use with the Parallax Stamps in Class curriculum, available for free download in Adobe's PDF format. Therefore, the use of a BoE on a robot chassis has more purpose than this single use as a robot project area.

The robot chassis is the Bot. Measurements are provided so you can build your own chassis on computer numerical control (CNC) metal machinery, or cut one from plastic or even wood. With the publication of part 2 of this course, the mechanical parts will hopefully also be available ready-made.

BASIC Stamp microcontroller

The BASIC Stamp is an inexpensive (less than $50USD) microcontroller with a built-in BASIC interpreter. The majority of microcontrollers require some special programming hardware. The BoE-Bot was designed around the most popular BASIC Stamp version — the BS2-IC. Here are some of the features of a BASIC Stamp:

- ➡ Small size, like a postage stamp.
- ➡ BASIC interpreter firmware is built into the PIC16C57 microcontroller.
- ➡ BASIC program storage in electrically erasable EEPROM. When power is applied the program executes. BASIC Stamps can be programmed at any time by temporarily connecting them to a PC running a simple host program. Type in a new program, press a key, and the program is downloaded to the BASIC Stamp.
- ➡ I/O pins that can communicate with digital devices sense switches, and even directly drive small loads like LEDs.

At this point in the course, the Stamp module will be treated as a black box. If you haven't already seen the BASIC Stamp II, you can download the manual, datasheets, and DOS/Windows editors from the Parallax web site. The UK distributor for the Parallax Stamp is Milford Instruments, telephone (01977) 683665, *www.milinst.demon.co.uk*

Building the Board of Education

The Board of Education (BoE) consists of a BASIC Stamp II module, a simple power supply, a prototyping area and some simple peripherals like a reset switch, an LED and a serial interface connector. The circuit diagram is shown in **Figure 1**.

Position IC1 is a 28-pin DIL socket in which the BASIC Stamp II module is plugged. The module communicates with your PC's serial port via a 9-way sub-D socket, K2. Note that a rudimentary form of handshaking is employed by capacitive coupling of the DTR (data terminal ready) signal on pin 4 of K2 to the ATN input line of the Stamp module.

The BASIC Stamp module can be reset by pressing push-button S1. The switch will be used whenever a program is to be downloaded into the Stamp.

All of the Stamp's I/O lines plus its V_{in} line (PWR, 9-15 V, see Stamp datasheets)) are 'bused' onto SIL sockets arranged adjacent to a prototyping area (breadboard) for easy connection to your own experimental circuits.

A simple on-board power supply

1

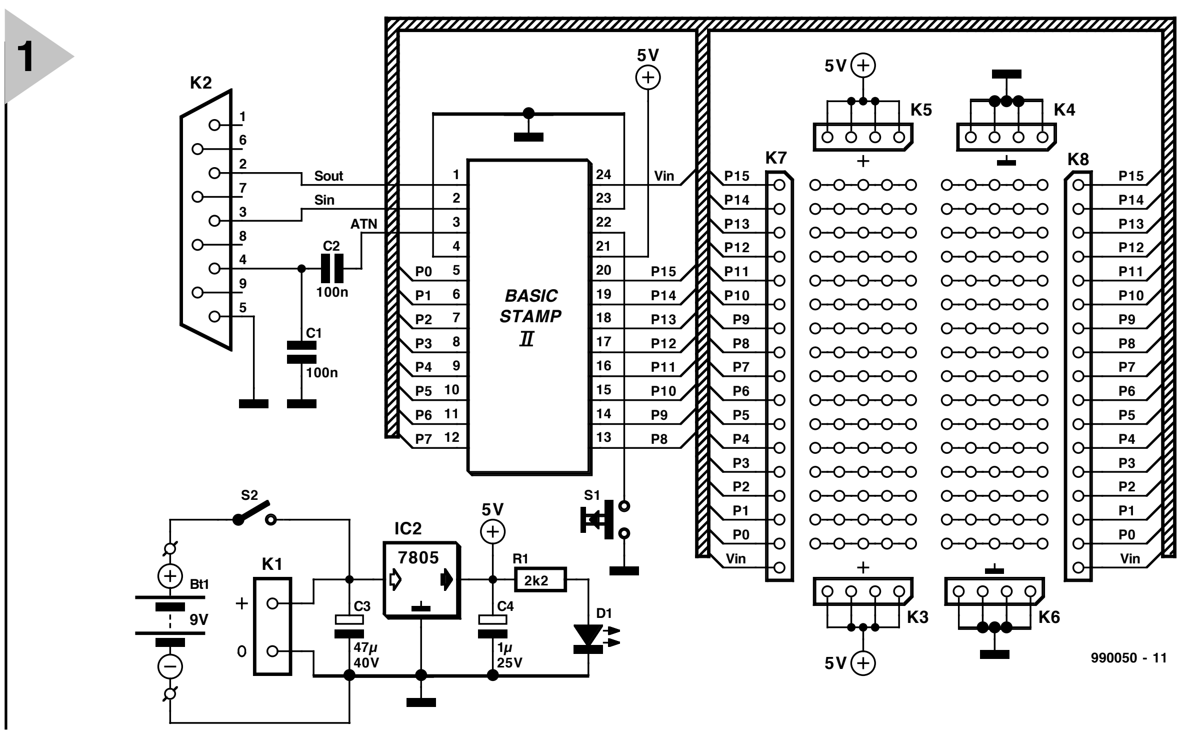
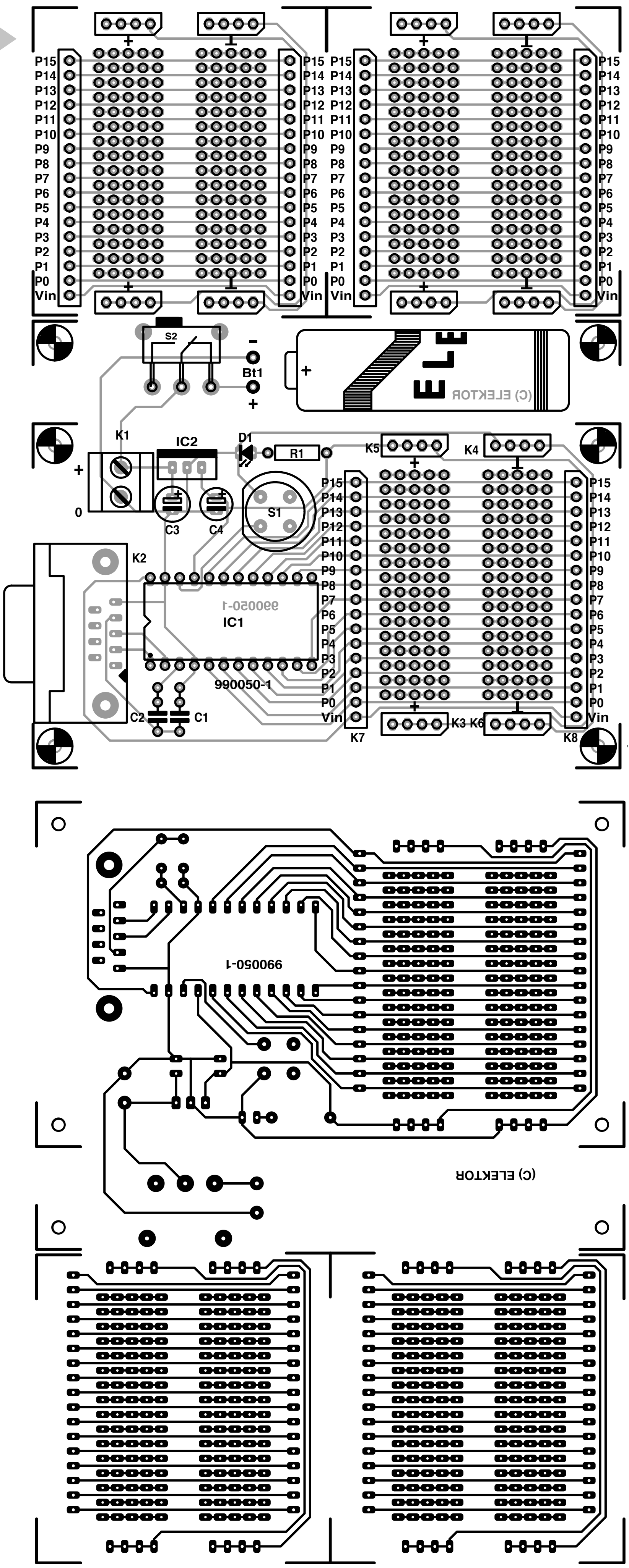


Figure 1. Circuit diagram of the Board of Education (BoE).

2



COMPONENTS LIST

Resistors:
R1 = 2kΩ2

Capacitors:
C1,C2 = 100nF
C3 = 47µF 40V radial
C4 = 1µF 25V radial

Semiconductors:
D1 = LED, red
IC2 = 7805

Miscellaneous:
Bt1 = 9V PP3 (6F22) battery with clip-on leads
IC1 = BASIC Stamp-2 module (available from Parallax distributors)
K1 = 2-way terminal block, raster 7.5mm
K2 = 9-way sub-D socket (female), PCB mount, angled pins
K3-K6 = 4-way SIL socket, turned pins
K7,K8 = 17-way SIL socket, turned pins
S1 = pushbutton, 1 make contact, PCB mount, type D6-R-RD (ITC)
S2 = slide switch, PCB mount
1 off 28-pin IC socket
2 off M3 nut and bolt for securing K2
PCB, order code **990050-1** (see Readers Services page)

Figure 2. Artwork for the single-sided circuit board (available ready-made).

allows the BoE to be powered either by a battery or a mains adaptor rated at 9 VDC/300 mA. A more powerful adaptor may be required depending on what you intend to connect to the SIL socket, or build in the prototyping area. Note that IC2 may then require a heatsink.

Warning. The mains adaptor must not be connected when the on-board battery is powering the circuit (switch S2 closed).

LED D1 lights when the 5-V supply voltage is present. The 5-V regulated (and decoupled) supply voltage is also made available to the experimental circuits via SIL sockets, this time K3 and K5. Two more SIL sockets, K4 and K6, are available for ground connections.

CONSTRUCTION

The printed circuit board designed for the BoE is shown in **Figure 2**. It is single-sided and available through the *Elektor Electronics* Readers Services as well as kit suppliers like C-I Electronics and Viewcom Electronics.

Stuffing the board should not present undue difficulty easy as the PCB

3

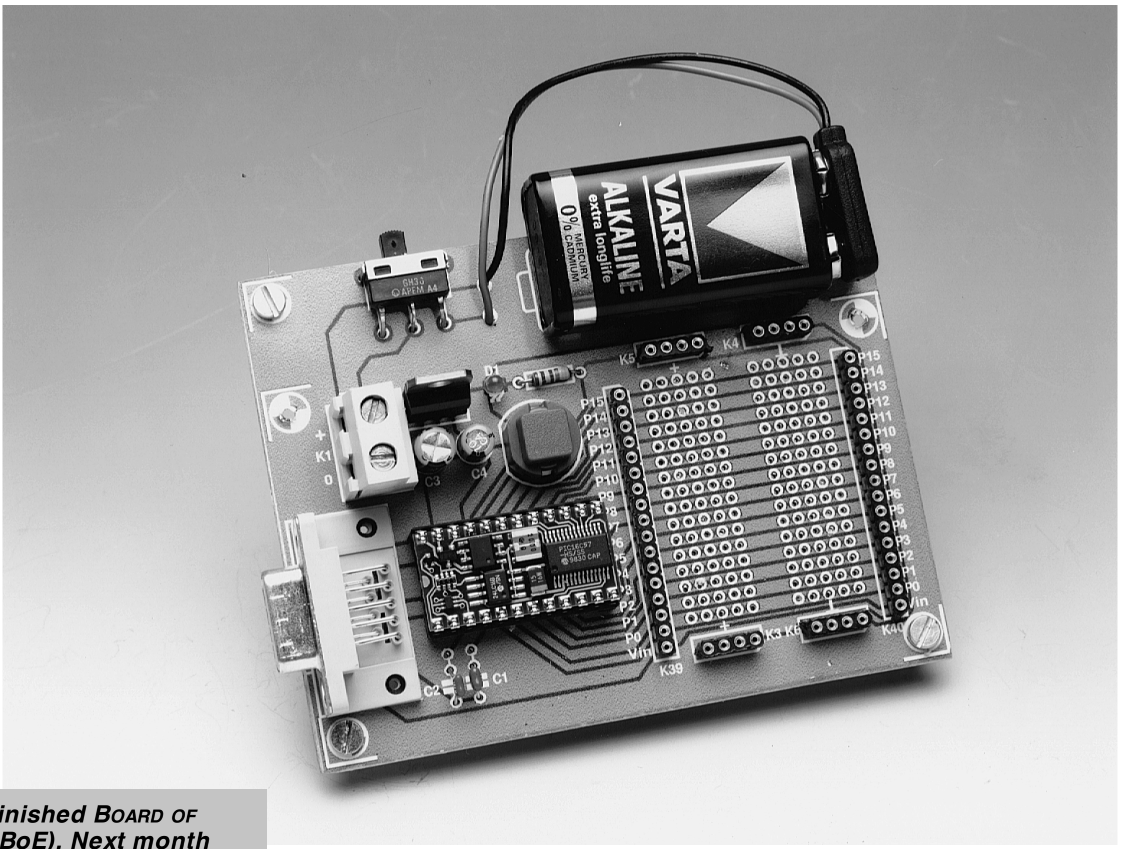


Figure 3. Finished BOARD OF EDUCATION (BoE). Next month we'll be installing the board on a home-made robot vehicle.

layout is generous and no special components are involved. Be sure to use a socket, that is, a **female connector**, in position K2. Capacitors C3 and C4, regulator IC1 and (yes) the socket in position IC1 are all **polarised components** and have to be mounted the right way around on the board. When in doubt about the orientation, study the circuit diagram and the component overlay. The BASIC Stamp module should not be installed into socket IC1 before you've finished all soldering work on the BoE.

The finished BoE (**Figure 3**) brings the BS2-IC I/O pins available adjacent to a breadboard. Only minimal wire stripping and cutting tools will be required to customize the BoE-Bot for future experimentation. The two identical prototyping areas may be cut off for experiments. If you fit suitable SIL pinheaders at the solder side of these sub-boards, they may be inserted into the sockets on the main BoE. A finished BoE will feature the following:

➡ easy plug-on connection of prototype circuits to be controlled by the BASIC Stamp module;
➡ DB9 connector for BS2-IC programming and serial communication during run-time;
➡ BS2-IC's P0-P15 I/O pins, V_{dd} (+ 5 V) and V_{ss} (GND) connections brought adjacent to 4 cm x 3 cm breadboard area;
➡ traces on top of the board show connections between BS2-IC port lines and breadboard connections;
➡ battery section may be cut off if object to be controlled has its own power supply (min. 9 VDC, connect to K1);
➡ two extra breadboards.

WANT A HEADSTART?

If you are interested in learning microcontrollers without the robot part and just the BoE, download the free curriculum available from the Parallax educational web site *http://www.stampsinclass.com*

The design concept of using BASIC Stamp on a robotic platform as an educational tool comes from Chuck Schoeffler, University of Idaho. Dr. Schoeffler's first approach was a very inexpensive configuration using the BASIC Stamp Rev. D., and this design could be considered if you need a cost-effective alternative. The BASIC Stamp Rev. D. has a built-in through-hole prototype area for projects, and may be mounted it's own unique robot chassis. This approach can be very effective for schools.

(990050-1)

Next month:
BoE-Bot mechanical assembly and calibration.

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.*

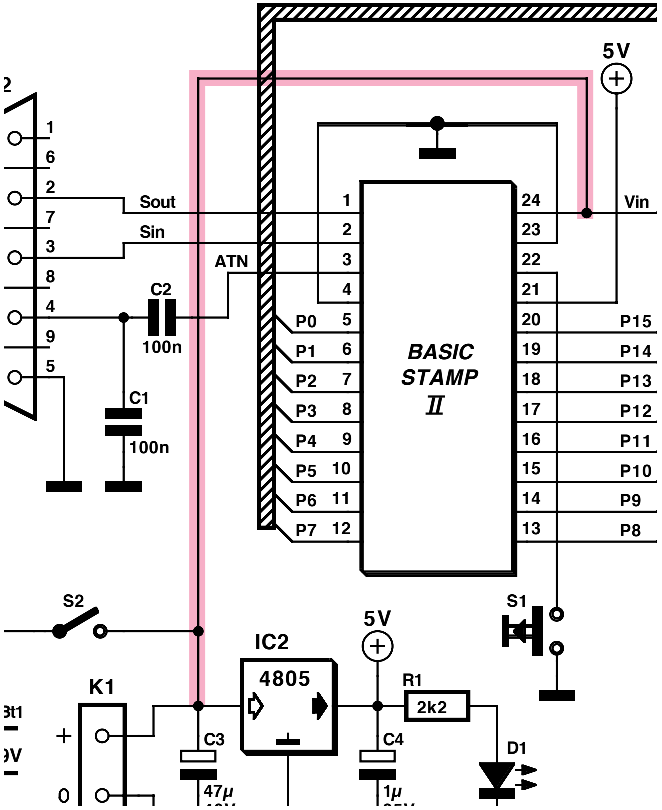
chucks@turbonet.com — creator of the BoE-Bot and author of this series. Technical assistance.

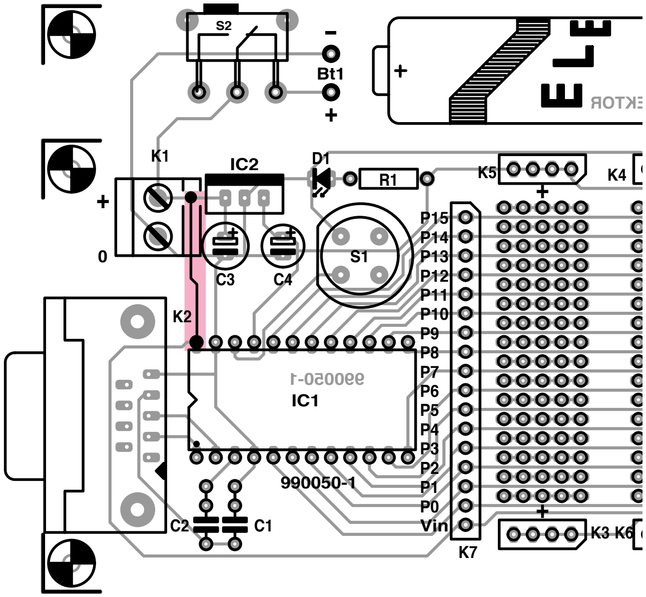
kgracey@parallaxinc.com — co-author of this article. Technical assistance and questions about the educational program.

http://www.milinst.demon.co.uk — UK distributor of Parallax BASIC Stamp.

BASIC STAMP PROGRAMMING COURSE (1)
September 1999, 990050-1
The positive battery voltage behind the on/off switch should be taken directly to the Vin pin of the Stamp module. The circuit diagram and PCB are modified as shown in the illustrations.

A batterypack consisting of four 1.5 V batteries should be used instead of the 9-volt battery originally indicated. This also requires the 7805 voltage regulator to be replaced by a low-drop type like the 4805.

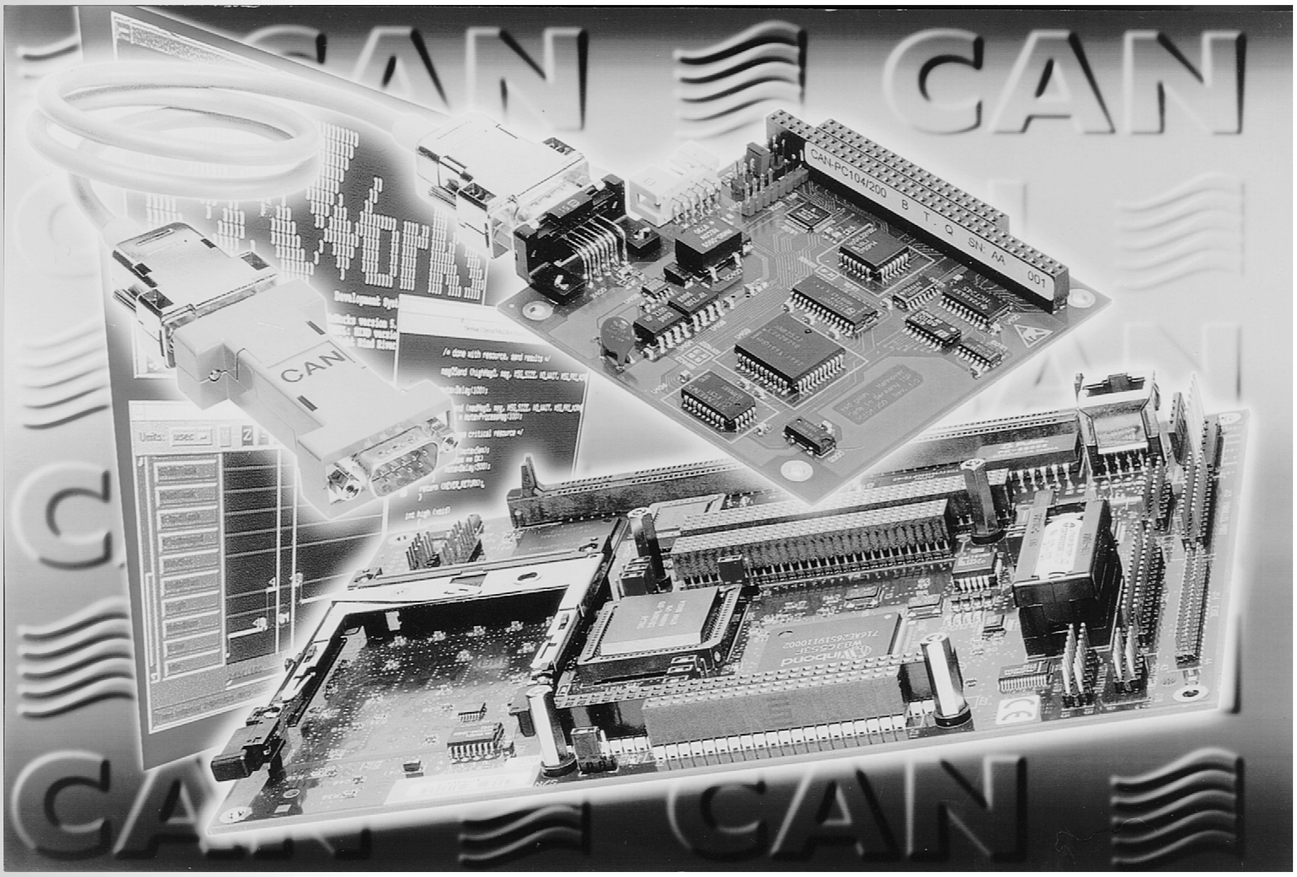




the CAN bus

intelligent, decentralized data communications Part 1

The proliferation of local communication networks in industrial systems seems inexorable: CAN bus, Profibus, LON, ASI, Interbus-S, FIP, EIB, eBus, and many more. The time has now arrived where these well-established technologies, protocol implementation in small silicon chips, dropping prices, and simple maintenance have become accessible to the smaller engineering firms and the enthusiastic amateur engineer/technician. In this short series of articles, the technologies underlying the CAN bus system will be described in a simple and practical manner. In a later part, a complete CAN bus interface for a microcontroller system will be presented.



By B vom Berg & P Groppe

1

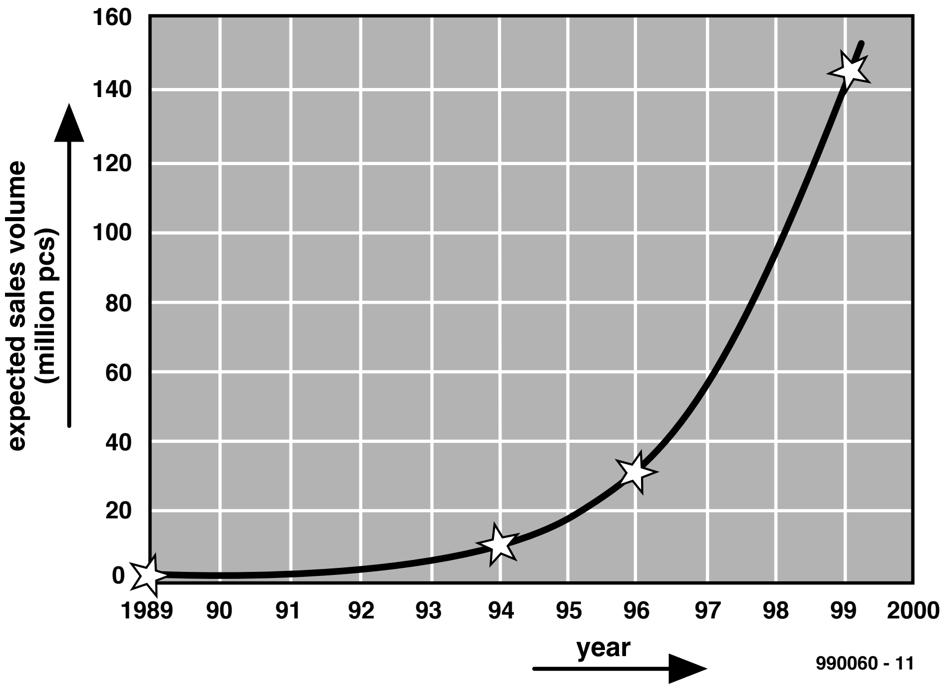


Figure 1. Estimated world-wide sales volume of CAN ICs.

INTRODUCTION

This first part in the series will take a brief look at the history and standardization of the CAN bus. It will also deal with the most important characteristics of the physical layer.

In the second part, the data link layer will be explained. This part will also contain the CAN building bricks of various manufacturers in tabular form. This is followed by the introduction of a universal CAN bus interface that may be used to make a variety of microcontroller and microprocessor system suitable for use in a CAN bus network.

The third part describes the construction, programming and usage of a small CAN bus network in conjunction with a PC and a microcontroller card.

DEVELOPMENT OF THE CAN BUS

In the early 1990s, the international car industry was faced with two problems concerned with the future development of private cars and goods vehicles. The first was concerned with the growing demand for more comfort in vehicles: electrically operated windows, seat and mirror adjustment, heated seats, electronic climate control, as well as audio-visual equipment and satellite-controlled navigation systems (GPS= Global Positioning System).

The second and more important was vehicle security, not only from an individual point of view but also to meet more stringent international safety regulations: central door locking, immobilizing systems; ABS (anti-lock braking systems), as well as economical and environment-friendly engine management.

Both problems were tackled by intense electronification of, and communication between, the many units comprising a modern vehicle. It is estimated that vehicles manufactured by the year 2005 will contain up to 100 microcontroller and all these need to communicate with each other. It is clear that all these communications paths result in a larger and more extensive cable harness. For instance, that in a modern good-quality motor car weighs close to 100 kg (220 lb) and contains up to 2000 metres (one and half mile) of cable. Moreover, a typical large car manufacturer may have up to 600 different types of cable harness in use.

Since this is clearly an untenable situation, the car industry began to look at new ways of communication and found this in the computer industry. Obviously, the bus systems used in that industry had to be adapted for use in vehicles, more particularly as regards the following:

- low-speed and high-speed data transfer in the range 5 kbit/s to 1 Mbit/s for comfort and safety systems;
- error-free data transfer with a Hamming distance greater than 4;
- optimum transfer of tiny data streams such as obtained from sensors or actuators, that is, consisting of 0–8 bytes per message;
- ease of maintenance;
- low cost (mass production);
- simplicity of bus construction (bus media, bus topology) for easy integration into the vehicle.

Unfortunately, all large car manufac-

2

ISO / OSI Layer model	CAN Bus Layer		
Layer 8 Application: "Device on Bus"	CANopen	DeviceNet	Smart Distributed System (SDS)
Layer 7 "Application Layer"	CAL: CAN Application layer for industrial Applications	DeviceNet Specifications	SDS Specifications
Layer 3 - 6	*Empty !!*		
Layer 2 "Data Link Layer"	LLC: Logical Link Control MAC: Medium Access Control acc. to ISO 11898 Result: CAN 2.0 A CAN 2.0 B } Specifications		
Layer 1 "Physical Layer"	"Low-Speed CAN" ISO 11519-2	"High-Speed CAN" ISO 11898	

990060 - 12

Figure 2. The CAN bus in the ISO/OSI layer model.

turers developed their own bus concept, which, of course, was not compatible with that of other manufacturers. All of them tried to have their system adopted as *the* international system, that is, to be accepted as *the* international standard – with clear economic and commercial advantages to them.

Not all systems could be readily standardized. Essentially, four of them have survived: CAN (Controller Area Network) in low-speed and high-speed versions; VAN; J1850SCP; and J1850DLC. Of these, VAN (standardized) and most others (not standardized) have been abandoned during the mid-1990s in favour of the CAN system. Today, the CAN system is *the* world leader in the field of vehicle buses. As such it is used nowadays not only in luxury cars from Mercedes, Lexus, Jaguar, and Chrysler, but also in less prestigious ones from Fiat and Volkswagen.

It was not only the vehicle industry that discovered the advantages of bus systems, but also the automation and production industries. These industries use the CAN concept for measuring, controlling and driving in SPS (Standard Positioning Service) systems, robots and motors. The concept is also used in civil engineering, elevator (lift) control, laboratory automation systems, sensor/actuator systems, and others.

The CAN protocol is available programmed in a silicon chip, so that the user need no longer concern him/herself with the finer details of the communication technology: CAN ICs are integrated simply as intelligent peripheral building blocks in existing microcontroller systems, or those being designed.

The virtually troublefree maintenance and usage, as well as the rapidly dropping prices, of CAN ICs make CAN buses of great interest and usefulness for non-industrial designers when it comes to small, decentralized communication networks.

Figure 1 shows the estimated world-wide sales volume of CAN ICs. The costs of a complete CAN network from various semiconductor manufacturers are given in US dollars.

STANDARDIZATION

If the structure of a communication system is to be universally accepted, a number of questions should be answered clearly and the answers incorporated in the relevant standard.

- How will the various parts of the network be physically (electrically and logically) arranged?
- What does the consequent topology of the network look like?
- How are the data coupled to, and transferred by, the relevant medium (cable, fibre optic conductor, air, or infra-red remote control)?
- What are the rules for data exchange between the various parts?
- How are data transfer errors prevented, recognized, and corrected?
- How is the data transfer protocol formed?
- How is access to the data transfer medium arranged for parts that are about to send data?
- How are conflicts resolved when several parts want to send data at the same time? This concerns the access to the medium, that is, the so-called arbitration.

As far as receiving data is concerned, there not many problems, since as a rule many receivers can be connected to the medium, all of which can receive data without any difficulty at the same time. In general, in any communication system, there should be only one sender active at any one time, although several receivers may be.

The answers to the foregoing questions, and others, must be laid down clearly and unambiguously when a communication system is to be used in a sensible manner that is acceptable world-wide.

In the early 1990s, the International Standardization Organization (ISO) started laying down an international standard for vehicle buses, during which process the CAN bus assumed an increasingly strong position.

The basis for the standardization process in the enormous area of open, non-producer-inspired data communications is a seven-layer ISO/OSI reference model. In the case of certain communication systems, including the vehicle bus system, layers 3–6 are empty, so that for the CAN bus only layers 1,2 and 7 are specified in detail (see **Figure 2**).

Layer 1: physical layer

In this layer, the specifications for the data transfer medium, connectors, the data transfer levels, and the send and receive elements are laid down. The two associated CAN standards are:

ISO11529-2: low-speed CAN. The basis of this is a development started by Bosch of Germany in the early 1980s, and continued, with the strong support of Intel, into the integration of the protocol into an IC. The low speed refers to data transfer rates from

3

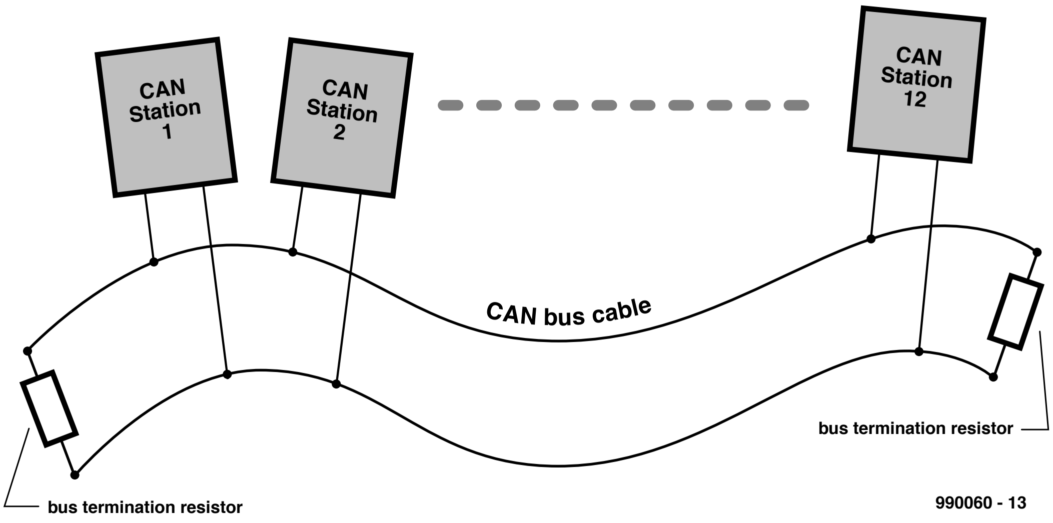


Figure 3. Topology of the CAN bus.

5 kbit/s to 125 kbit/s.

ISO11898: high-speed CAN. This standard supports data transfer rates of up to 1 Mbit/s.

Layer 2: data link layer
This layer lays down how, when a part wants to send data, the data transfer medium7m is accessed, how a message is composed (address, data, control and protection against errors), and how the data transfer protocol is structured. The standards for these may also be found in ISO11898.

In addition, the 1991 CAN specification is expanded in Layer 2, so that today there two versions: CAN2.0A and CAN2.0B. The similarities and differences between these two will be reverted to in Part 2.

Since the CAN concept has been adopted by so may different industries, the situation in Layer 7: application layer, is diverse and somewhat confused. This is because this layer defines the interface for the actual application (Layer 8) of the bus with industrial equipment.

Three extensive CAN branches for different applications were developed over the years: CANopen, DeviceNet, and Smart Distributed System (SDS). Since these specifications are fairly extensive, they will not be pursued in this article. All that will be said here is that the concepts are common, so that they are compatible with Layers 1 and 2. Detailed information on CANopen, DeviceNet, and SDS may be found on the internet: http://www.can-cia.de.

CHARACTERISTICS

The physical layer comprises the network topology of the CAN bus and the linking to the bus medium.

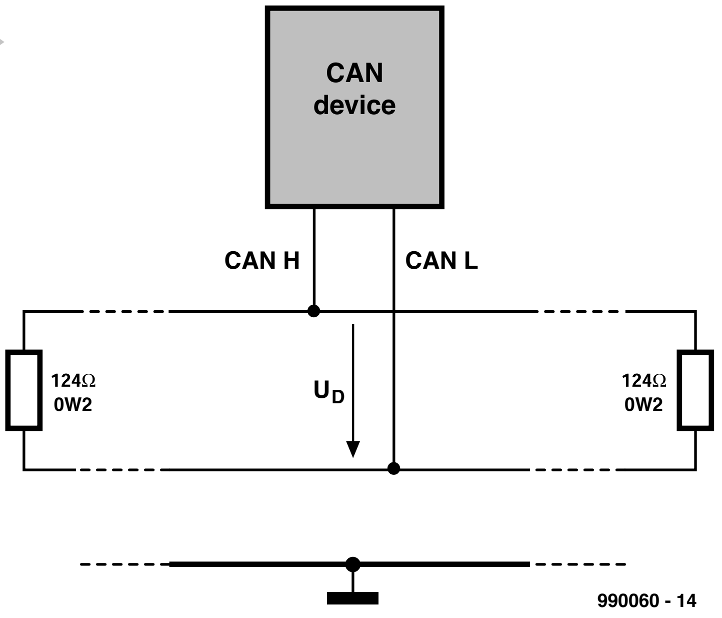
The term network topology includes the physical construction of

Voltage at ...\\ Bus state	recessive	dominant
CANH	2.5 V	3.5 V
CANL	2.5 V	1.5 V
allowable voltage difference U_D = CANH - CANL	0 - 0.5 V	0.9 - 2.0 V

Table 1. Absolute levels of the bus lines with respect to (local) earth according to ISO11898.

Figure 4. Linking an element (station) to the CAN bus.

4



5

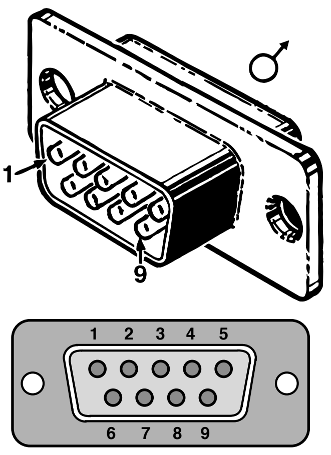


Figure 5. Pinout of a CAN bus connector.

the communication system and so gives the answer to 'how are the parts (stations) linked to the data transfer medium?'

The CAN bus uses the so-called bus topology, that is, all parts are connected to a single twisted-pair cable (screened or not), which is terminated at both ends into the relevant bus termination impedance (see **Figure 3**). This arrangement ensures that each station can communicate with any other station in the network without any limitation.

The send/receive stage of a CAN network element is linked to the bus medium via two connectors: CAN High (CANH) and CAN Low (CANL) (see **Figure 4**).

In view of the requisite protection against errors, differential voltage signals are used for the actual data transfer. This means that the voltage different between the two bus lines is quantized. ISO 11898 specifies two different differential-voltage ranges for data representation: recessive and dominant. There is a good reason that the usual logic 0 and 1 levels are not used here and this will be reverted to. For the time being, note that

- if the differential voltage between CANH and CANL ≤ 0.5 V, the status is recessive;

- if the differential voltage ≥ 0.9 V, the status is dominant.

The nominal level of the bus line, that is, the level of the individual lines with respect to (local) ground is shown in **Table 1**.

In practice, these levels are, of course, subject to tolerances, so that the voltage difference may reach the maximum permissible level shown in the last row.

The specifications in ISO 11519-2 (CANL) are slightly different, but since ISO 11898 may be used for both high-speed and low-speed, that specification is invariably used nowadays.

Users need not concern themselves over the construction of a send/receive link since most manufacturers have available ready-made ICs for this purpose. These are optimized particularly as regards electromagnetic compatibility (EMC), board space and thermal overload (in case of a short-circuit of CANH or CANL), and output standard CAN signal levels. All that is necessary to establish a CAN link is for them to be coupled to the bus line.

All that is necessary is to make sure to which standard the IC is built: ISO 11519-2 or ISO 11898: the latter should be preferred. It should be noted that there are also other differential voltage procedures that may be used in CAN signal transfer, for instance, RS485.

Final questions to be asked about the CAN bus system are
- What is the maxi8mum bus expansion for a given data transfer rate?
- How many elements (stations) may be connected to the bus?

The answers to these questions depend solely on the bus medium used. **Table 2** shows the correlation between the data transfer rate, length of the bus, bus medium, and the bus termination impedance.

The data transfer medium should preferably be a twisted-pair cable with a cross-sectional area of 0.34–0.6 mm^2, while the bus termination impedance should be around 127 Ω. The resistivity of the cable should be not greater than 60 mΩ/m, a condition that is met when the cross-sectional area is greater than 0.30 mm^2.

Care should be taken with the length of branching lines in case the station is not connected directly to the CAN bus. These lines should not be longer than 2 metres when the data transfer rate is 250 kbit/s, and not longer than 30 cm when the data transfer rate is greater. The total length of all branching lines should not exceed 30 metres.

Finally, it should be noted as regards Layer 1 that the connectors and their pinout for linking elements to the bus are standardized.

[990060]

Table 2. Correlation between data transfer rate, length of the bus, bus medium, and bus termination impedance.

Bus length	Bus cable		Bus termination resistance	Maximum data rate
	resistance	cable c.s.a.		
0 - 40 m	70 mΩ/m	0.25 - 0.34 mm^2 AWG23, AWG22	124 Ω (1%)	1 Mbit/s at 40 m
40 - 300 m	< 60 mΩ/m	0.34 - 0.6 mm^2 AWG22, AWG20	127 Ω (1%)	500 Kbit/s at 100 m
300 - 600 m	< 40 mΩ/m	0.5 - 0.6 mm^2 AWG20	150 Ω to 300 Ω	100 Kbit/s at 500 m
600 m - 1 km	< 26 mΩ/m	0.75 - 0.8 mm^2 AWG18	150 Ω to 300 Ω	50 Kbit/s at 1 km

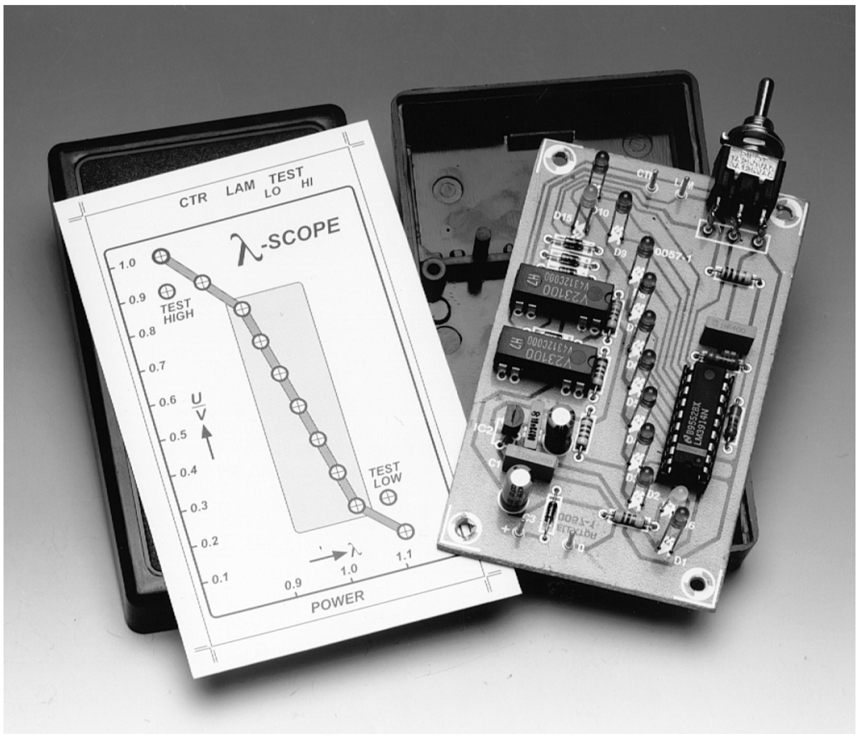
See also:
'CAN – the Controller Area Network' in the September 1992 issue (p. 56) of *Elektor Electronics*.

EGO sensor tester

ensuring a correct air-fuel ratio

One of the most important parameters in modern motor vehicle management is the air-fuel ratio, since this affects the 'cleanliness' or otherwise of the exhaust gases. In fact, the ratio is measured indirectly by the amount of oxygen in the exhaust gases. The sensor used for this is the exhaust gas oxygen (EGO) sensor. This is sometimes called a lambda sensor from the Greek letter lambda (λ) which denotes the equivalence ratio (ratio of actual air-fuel mixture to ideal air-fuel mixture). The correct operation of this sensor may be tested and monitored with the circuit described in this article.

Design by H. Bonekamp



Most modern engine control systems are digital and contain a number of microprocessors; in fact, they are essentially special-purpose computers. Their primary purpose is to regulate the air-fuel mixture, the ignition timing and and the exhaust gas recirculation (EGR).

The air-fuel ratio is important not only from an economic point of view, but also from an environmental one, since it affects the amount of harmful constituents in the exhaust gases. It is controlled by an engine control system that consists of a number of sensors (air flow; engine coolant; engine position; throttle position; exhaust gas oxygen; and others), a digital control unit, and various actuators (air-fuel ratio; ignition; exhaust gas recirculation – EGR).

One specific air-fuel ratio is crucial: the stoichiometric (chemically correct) mixture. This mixture corresponds to an air-and-fuel combination of which, if combustion were perfect, all the hydrogen and carbon would be converted by the burning process into harmless water (H_2O) and carbon dioxide (CO_2). The stoichiometric ratio for a petrol engine is 14.7:1. The ratio of the actual fuel-and-air mixture to the stoichiometric mixture is the equivalence ratio, represented by the Greek letter lambda (λ).

Unfortunately, combustion is not perfect: the exhaust gases also contain harmful carbon monoxide (CO), oxides of nitrogen (NO_x), unburned hydrocarbons (HC), oxides of sulphur, and others.

Government standards in all western countries require these harmful emissions to be reduced in stages by up to 100 per cent. The device most frequently used by vehicle manufacturers is a catalytic converter in the exhaust system. Exhaust gases passed through the converter are chemically altered to help meet these standards. The main functions of the converter are therefore to:

- oxidize hydrocarbon emissions to carbon dioxide (CO_2) and water (H_2O);

1

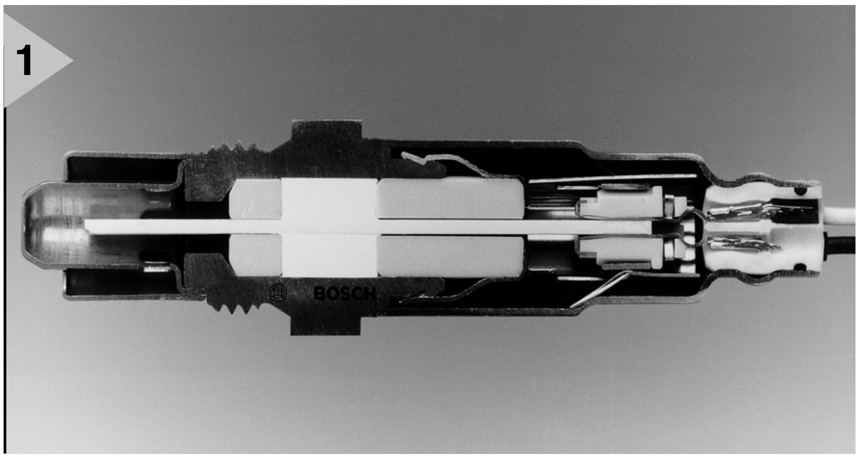


Figure 1. Typical planar, wide-band EGO sensor (Bosch).

- oxidize CO to CO_2;
- reduce NO_x to nitrogen (N_2) and oxygen (O_2).

The purpose of the oxidizing catalyst (OC) is to increase the rate of chemical reaction which initially takes place in the cylinder as the compressed air-fuel mixture burns, to an exhaust gas that has a complete oxidation of HC and CO to H_2O and CO_2.

The amount of oxygen in the exhaust gases is used as an indirect measurement of the air-fuel ratio. Consequently, one of the most important sensors in use in modern vehicles is the exhaust gas oxygen (EGO) sensor. This is sometimes called the lambda sensor since this Greek letter (λ) denotes the equivalence ratio mentioned earlier. The most commonly used EGO sensor is based on zirconium dioxide (ZrO_2); another one is based on titanium dioxide (TiO_2).

HEATED EGO SENSOR

The stringent exhaust emission requirements have made it necessary for the time from engine start to the point at which the EGO reaches operating temperature to be shortened. This has led to the development of the heated exhaust gas oxygen (HEGO) sensor. This is electrically heated from engine start until the sensor is active (at about 600 °C).

The HEGO sensor shown in **Figure 1** analyses the various constituents of the exhaust gases. The output of the sensor is fed back to the electronic control unit, which is then able to adjust the air-and-fuel mixture as required.

The sensor consist of a section of zirconium dioxide with thin platinum electrodes on the inside and outside of the ZrO_2. The inside electrode is exposed to air, and the outside electrode to exhaust gases through a porous protective coating.

The platinum plate at the air side is exposed to a much greater concentration of oxygen ions than the plate at the exhaust gas side. This results in the air-side plate becoming electrically more negative than the exhaust-side plate. Consequently there is an electric potential between the plates: positive at the exhaust gas side and negative on the air side. The level of this potential depends on the concentration of oxygen in the exhaust gases and on the sensor temperature.

NOT TOO LEAN; NOT TOO RICH

When the air-and-fuel mixture is stoichiometric (14.7:1), the equivalence ration, λ, is 1. When this ratio is < 1, the mixture is rich, and when it is > 1, the mixture is lean. At the stoichiometric mixture, the output of the EGO sensor alternates around 0.5 V; the frequency of alternation depends on the time constant of the control loop. A steady output of 0.5 V indicates that either the control loop is defect or the EGO sensor is inactive or cold.

When the sensor output is higher than 0.5 V, the mixture is too rich (not enough oxygen). This is only permissible when the engine is cold or under full-load conditions (in some makes of car the fuel-injection unit then enriches the mixture). In other circumstances, the fuel-injection unit and/or the engine temperature sensor should be tested.

2

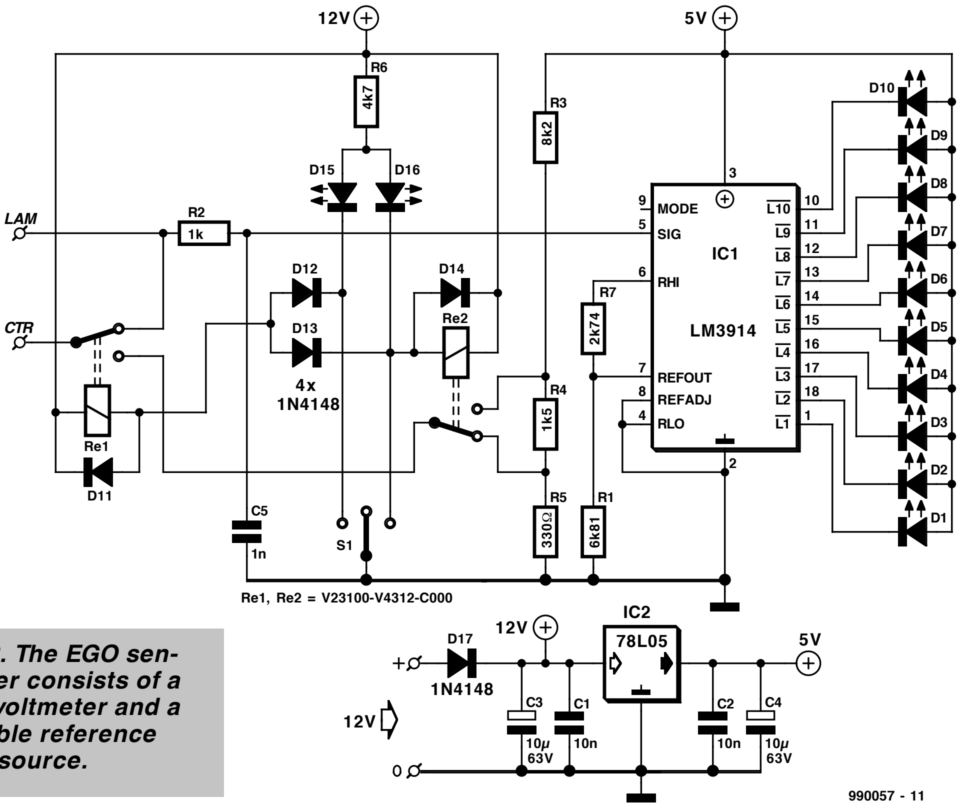


Figure 2. The EGO sensor tester consists of a kind of voltmeter and a switchable reference voltage source.

3

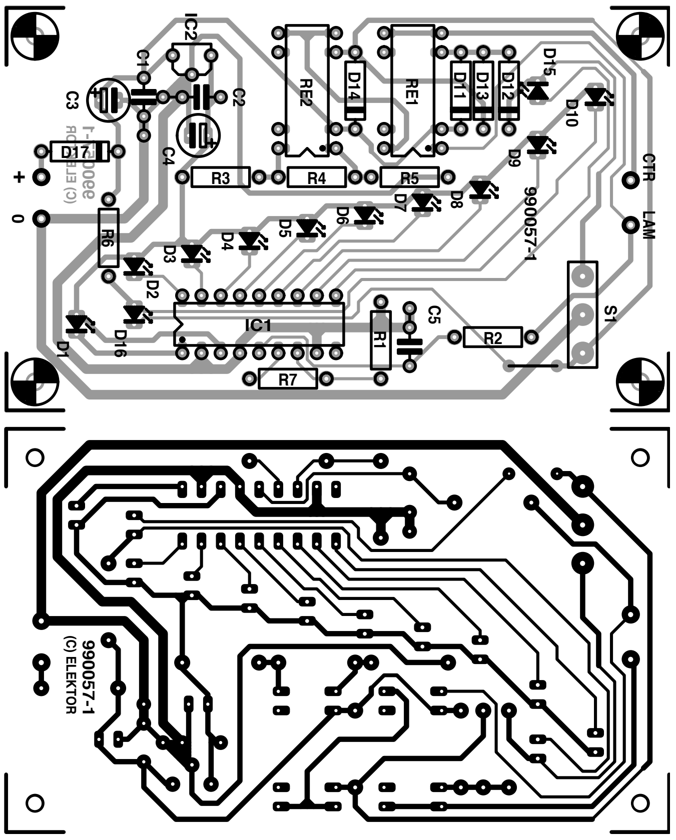


Figure 3. Printed-circuit board for the EGO sensor tester (not available ready made).

Parts list

Resistors:
R_1 = 6.81 kΩ
R_2 = 1 kΩ
R_3 = 8.2 kΩ
R_4 = 1.5 kΩ
R_5 = 330 Ω
R_6 = 4.7 kΩ
R_7 = 2.74 kΩ

Capacitors:
C_1, C_2 = 0.01 µF
C_3, C_4 = 10 µF, 63 V, radial
C_5 = 0.001 µF

Semiconductors:
D_1–D_{10} = LED, 3 mm, red, high efficiency
D_{11}–D_{14}, D_{17} = 1N4148
D_{15} = LED, 3 mm, yellow, high efficiency
D_{16} = LED, 3 mm, green, high efficiency

Integrated circuits:
IC_1 = LM3914
IC_2 = 78L05

Miscellaneous:
S_1 = 3-pole change-over switch with centre rest position for board mounting
Re_1, Re_2 = reed relay, 12 V coil with single change-over contact, Siemens V23100-V4312-C000

When the output of the sensor is < 0.5 V, the mixture is too lean. This may be the case when the throttle is released at high engine revolutions. It is more likely, however, that there is insufficient fuel injected.

CIRCUIT OF PROPOSED TESTER

All that is needed for checking the correct operation of the EGO sensor is a kind of voltmeter as shown in **Figure 2**. Apart from terminals for inputting the supply voltage, the unit has terminals for connecting the output of the (H)EGO sensor (LAM) and the electronic engine controller (CTR).

The sensor output is applied to pin 5 (SIG) of LED-display driver IC_1. Network R_2-C_5 forms a low-pass filter that eliminates noise peaks on the signal line.

The reference voltage of IC_1 is set to such a value by R_1 and R_7 that the diodes cover a voltage window from 0 V (D_1) to 1 V (D_{10}) in 0.1 V steps. Diode D_5 represents the stoichiometric value (0.5 V).

The MODE pin (9) is open so that the display operates in the dot mode, which means that only one LED lights at any one time.

The cable in the vehicle between the electronic control unit and the sensor must, of course, be disconnected before the tester can be used. In some vehicles, it may be possible to cut this cable as appropriate and use one of the resulting lengths to connect the control unit to the tester, and the other to connect the tester to the sensor. This may, however, affect your rights under the terms of the vehicle warranty.

MONITORING & TESTING

Switch S_1 enables one of three modes of operation to be selected. When it is in the centre position as shown in the diagram, terminals LAM and CTR are interlinked, so that the control unit behaves as if the tester were not there. The tester then merely monitors the output voltage of the sensor, which is indicated by the LED display. It may remain connected when the vehicle is in use, but it should be borne in mind that this may invalidate guarantees and warranties.

In both other positions of the switch, the unit is in the test mode. The control unit is then no longer linked to the EGO sensor and the fuel injection is driven by a virtual high sensor output (test-hi) or a low one (test-lo). These virtual outputs are derived from potential divider R_3–R_5, the output of which is selected by the contact of relay Re_2. Diodes D_{15} and D_{16} indicate the selected test voltage (D_{15} in test-lo and D_{16} in test-hi).

The fuel injection actuator will optimize the air-fuel mixture: enrich it when the test voltage is low, and make it leaner when the test voltage is high.

The supply voltage regulation is traditional. Diode D_{17} prevents any damage to the electronic circuits when the supply lines are linked with incorrect polarity. The output is a clean and stable voltage at a level of 5 V. Note that the two relays are operated directly from the 12 V supply input.

CONTROL CHARACTERISTIC DISPLAYED

The display LEDs are placed on the printed-circuit board (not available ready made) for the EGO tester in the shape of a control curve. When only the diodes in the marked window at the centre of the display light it may be assumed that all is well. When diode D_1 lights, the air-fuel mixture is too lean; when D_9 or D_{10} lights, the mixture is too rich.

The diodes are placed along the control curve in such a way that the fuel injection control must correct the mixture to a value adjacent to the lit diode.

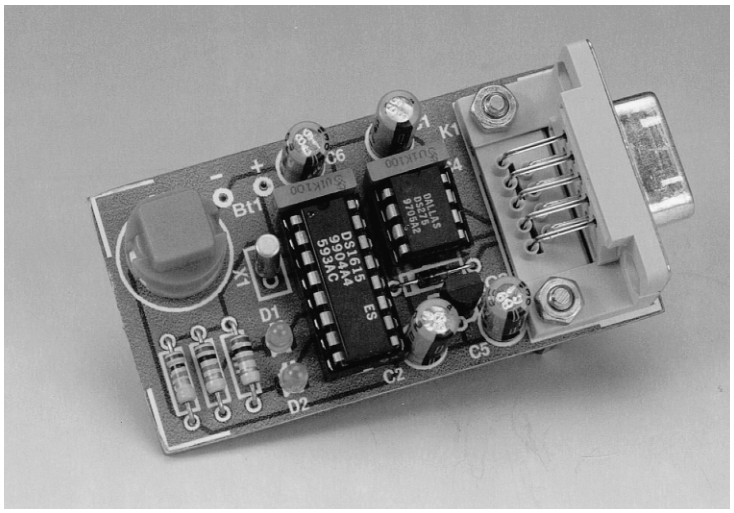
Completing the board and assembling the tester in a suitable case are straightforward. Circuit IC_1 and the two reed relays should be soldered directly to the board. The LEDs should not be soldered in place until it has been ascertained that they protrude slightly from the requisite holes in the enclosure.

The cables linking the tester to the sensor and to the control unit should be single screened types. The sensor output should be taken from a connector as close as possible to the sensor (which is mounted securely in the exhaust manifold). [990057]

temperature recorder DS1615

for computerised long-term temperature logging

Dallas Semiconductor supplies a series of sensors that add capture, storage and even data sorting functions to their expected task of measuring physical quantities. The DS1615 temperature recorder chip is a particularly interesting member in this family. In this article we propose a small PCB for it, as well as software so you can experiment to your heart's content.



Design by L. Lemmens

The DS1615 is an integrated temperature recorder which is not only capable of working as a data logger (for temperature measurements) but also of histogram tabulation (number of discrete measurements) taken within the temperature range –40°C to + 85°C.

Its on-chip real-time clock (RTC) counts seconds minutes, hours, days, months, days of the week and years, all in BCD (binary coded decimal) format. The RTC is aware of leap years and claimed to be fully Millennium proof.

The temperature recorder may be programmed to any alarm time you want within one week. When the alarm time is reached, the DS1615 generates an interrupt.

The IC allows you to program measurement intervals ranging from 1 minute to 255 minutes. This applies to both measurement modes. The non-volatile measurement data memory has a capacity of 2048 samples including start a time marker ('stamp') when used as a data logger, or 63 two-byte data bins in 2°C increments when the histogram mode is employed. Each of these 63 data bins is capable of recording 25,535 measurements.

High and low temperature trip points may be programmed at which an alarm condition is generated (i.e., an interrupt is generated, or a status pin toggles).

The DS1615 is connected to a host PC via a serial interface capable of synchronous and asynchronous operation. The command to start measuring does not necessarily have to arrive via the serial interface — it may also be issued by pressing a pushbutton.

Memory map

Because of the relative complexity of the IC, it is impossible to describe all its features on a couple of magazine pages. If you want to become conversant with the DS1615 hardware and software environment, you should download the 20-page datasheet from this web URL

www.dalsemi.com/DocControl/PDFs/1615.pdf
A selection of essential facts about the DS1615 may also be found on the *Datasheets* in this month's issue.

The block diagram in **Figure 1** shows the general structure of the temperature recorder chip. It is linked to the outside world by control logic, the serial interface and, of course, the sensor. The clock signal is supplied by a 32.768 kHz watch crystal.

The rest of the architecture covers the memory, which is divided in pages of 32 bytes each. Pages 0 and 1 contain the real-time clock and the control registers, while the non-volatile user RAM is accommodated on page 2. Page 16 contains an optional 64-bit serial number which may be useful for product identification and tracking. The alarm time markers and alarm durations are stored on pages 17, 18 and 19. Pages 47-67 are reserved for the temperature histogram, while the temperature logging memory resides in pages 128-191. Pages not mentioned here are reserved for future functions.

The most interesting page is without doubt the first one (RTC and Control Page). Its organisation as shown in **Figure 2** enables us to explain the different commands and modes of operation.

Temperature recording

The temperature sensor has a range of –40°C to + 85°C at a maximum error of ± 2°C. The temperature value is expressed in a byte starting with value 00000000_b for –40°C, up to 11111010_b for + 85°C. The temperature data byte (TByte) may be converted into degrees Celsius by the following equation:

$$°C = 0.5\,(\text{TByte}) - 40$$

The current temperature is stored at location 11 of the RTC and Control page. Unless data capturing has been started, the value is available for further processing or instantaneous displaying. Temperature values are valid when the Temperature Ready (TR) bit is set. This bit is at 0 when a measurement value is being computed and written into the memory.

Data logging

In *data logging* mode temperature samples are successively written to memory locations (registers) starting at 1000_h, until a total of 2,048 registers have been filled.

As already mentioned, a data logging mission may be started either by the host PC via the serial interface, or by the user pressing a pushbutton connected to the DS1615. When PC control is used, the SE bit in the control register has to be made 0, causing the $\overline{\text{ST}}$ input pin to be disabled. Any non-

1

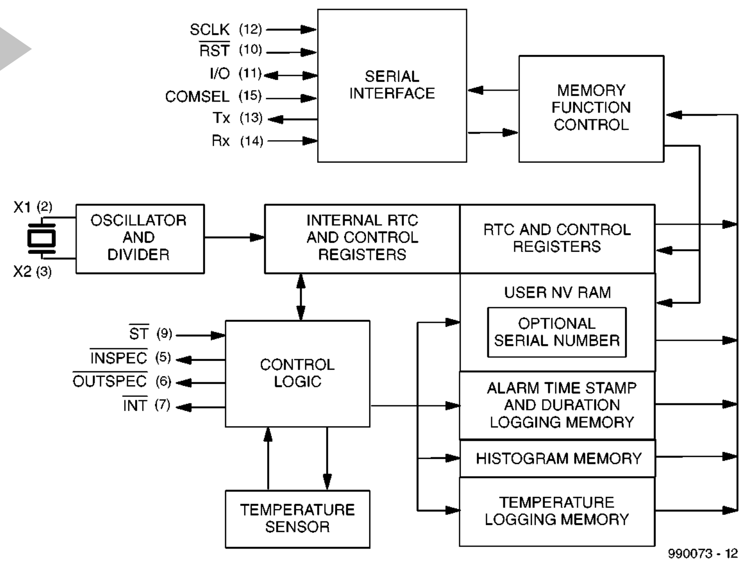


Figure 1. Block diagram of the DS1615 temperature recorder chip.

2

Figure 2. RTC and Control page.

DS1615 RTC AND CONTROL PAGE Figure 2b

ADDR.	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION
00	0	10 Seconds			Single Seconds				Real Time Clock Registers
01	0	10 Minutes			Single Minutes				
02	0	12/24	10 h A/P	10 h	Single Hours				
03	0	0	0	0	0	Day Of Week			
04	0	0	10 Date		Single Date				
05	0	0	0	10 m.	Single Months				
06	10 Years				Single Years				
07	MS	10 Seconds Alarm			Single Seconds Alarm				Real Time Clock Alarm
08	MM	10 Minutes Alarm			Single Minutes Alarm				
09	MH	12/24	10 ha. A/P	10 h. alm.	Single Hours Alarm				
0A	MD	0	0	0	0	Day Of Week Alarm			
0B	Low Temperature Threshold								Temperature Alarm
0C	High Temperature Threshold								
0D	Number Of Minutes Between Temperature Conversions								Sample Rate
0E	$\overline{\text{EOSC}}$	CLR	0	SE	RO	TLIE	THIE	AIE	Control
0F	(reads 00h)								Reserved
10	(reads 00h)								Reserved
11	Current Temperature								Temperature
12	Start Delay Register (LSB)								Start Delay
13	Start Delay Register (MSB)								Start Delay
14	TR	MEM CLR	MIP	SIP	LOBAT	TLF	THF	ALMF	Status
15	Minutes								Start Time Stamp
16	Hours								
17	Date								
18	Month								
19	Year								
1A	Low Byte								Current Samples Counter
1B	Medium Byte								
1C	High Byte								
1D	Low Byte								Total Samples Counter
1E	Medium Byte								
1F	High Byte								
20–3F	(Read 00H)								Reserved

990073 - 13

3

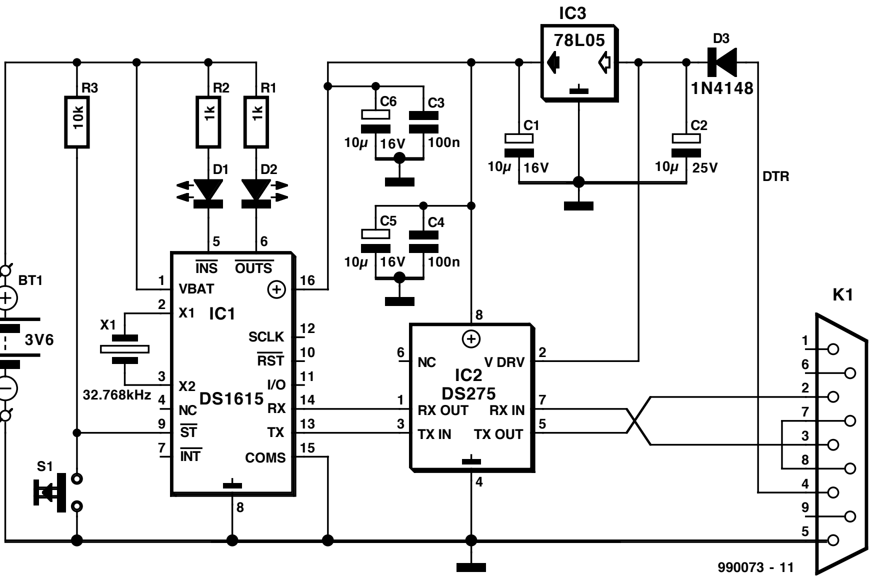


Figure 3. Circuit diagram of the temperature recorder.

zero value written to the sample rate register initiates the datalog mission. The pushbutton method of starting a datalog mission is enabled by setting the SE bit to 1. The mission will start when a non-zero value has been written to the sample rate register and the $\overline{\text{ST}}$ pin has been held low for at least 0.5 second.

Once the datalog mission has been initiated (by either method), outputs $\overline{\text{INSPEC}}$ and $\overline{\text{OUTSPEC}}$ will generate four low pulses simultaneously, and the *mission-in-progress* bit (MIP) is set to 1 in the status register. The time between the start of the mission and the first temperature recording is indicated in minutes in the *start delay register* at addresses 12 and 13. This register is clocked once a minute by the seconds register of the real-time clock.

Two methods are available to deal with data overrun of the data samples counter. The first alternative is to employ the roll-over feature and set bit 3 of the control register to 1. New data will then overwrite old data from address 1000_h onwards. The other method is to halt data capturing once 2048 measurements have been made. This requires programming a 0 for the *roll-over bit* (RO).

Although the DS1615 uses the RTC to make a time stamp for the first sample recording, time stamps for subsequent measurements have to be computed from the value of the *current sample counter* and the sample address. This is not a problem when roll-over has been disabled. By contrast, in roll-over mode the user has to evaluate the contents of the *session sample register*. By dividing this value by $07FF_h$ you get the number of overruns that have occurred.

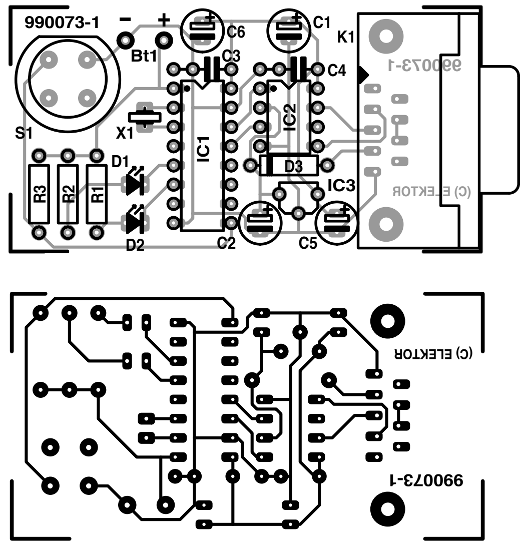
For security reasons, the end user can not write to the temperature datalog memory.

HISTOGRAMS

During a datalogging mission the DS1615 also computes a histogram of the recorded temperature samples. The results are stored in the four temperature histogram memory pages addresses 0800_h – $087F_h$. These pages contain 63 two-byte data bins. Each bin consisting of a 16-bit counter, its count range is 0-65,535. This counter is automatically incremented each time a measured temperature falls within its range. For example, bin # 0 captures temperatures in the lowest range, –40°C to –38.5°C (values 0_b, 1_b and 11_b), bin # 1 is for temperatures between –38°C and –36.5°C (values 11_b, 100_b and 101_b), and so on up to bin # 62 for samples between + 84°C and + 85°C (values 11111000_b, 11111001b and 11111010_b). All data bins will remain at the maximum count of 65,535 when this number is exceeded.

Figure 4. PCB track layout and component overlay (PCB available ready-made).

4



COMPONENTS LIST

Resistors:
R1,R2 = 1kΩ
R3 = 10kΩ

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

Semiconductors:
D1,D2 = LED, low-current
D3 = 1N4148
IC1 = DS1615 (Dallas Semiconductor)
IC2 = DS275 Dallas Semiconductor)
IC3 = 7805

Miscellaneous:
X1 = 32.768 kHz quartz crystal
BT1= Lithium cell, 3.6V
S1 = pushbutton, 1 make contact
PCB, order code **990073-1** (see Readers Services page)
Disk set, set, order code **996030-1** (see Readers Services page)

Out of spec

In many applications it is useful to know the number of occurrences, and the duration, of temperatures outside a certain 'allowed' range. The relevant limits (upper and lower) are stored in the *temperature alarm register* which may be found at addresses $000B_h$ and $000C_h$. No alarm condition is recorded by the DS1615 as long a measurement value falls within this user-defined range. If an out-of range value is measured, the IC sets the temperature high flag (THF) or the *temperature low flag* (TLF), depending on the actual value. These fags are found in the status register at address 0014_h. The temperature recorder also produces a time stamp marking the start of the alarm, and measures how long the temperature is out of range. When the upper limit is exceeded, the $\overline{\textbf{INT}}$ pin produces an interrupt provided the temperature-high interrupt enable bit (THIE) is set. Similarly, an INT signal is produced when the *temperature low interrupt enable* (TLIE) bit is set, and the measured temperature drops below the lower limit.

Time stamps and duration data may be found in the address ranges 0220_h – $024F_h$ for low temperatures, and 0250_h – $027F_h$ for high temperatures. In this way, up to twelve individual alarm conditions ('over' and 'under' temperature) may be recorded and time-stamped. The date and time information for the alarm periods are computed from the data read from *start time stamp* and the sampling interval.

The status of the DS1615 may be read from the logic states of its $\overline{\textbf{INSPEC}}$ and $\overline{\textbf{OUTSPEC}}$ terminals. When the user launches a datalog mission, four active-low pulses appear simultaneously at both outputs. If the DS1615 is polled for data on recorded temperature samples, and these values are within the allowable range, then the $\overline{\textbf{INSPEC}}$ pin supplies four low pulses. If at least one sample was over or under the set limits, the $\overline{\textbf{OUTSPEC}}$ pin is pulsed. If the chip is polled after a datalog mission has been started, but still before the first measurement, then four low pulses appear alternating on both pins.

Clock, calendar and alarm

Access to the timekeeping and calendar data (stored in BCD format) is via read or write operations to the corresponding register bytes. The DS1615 operates either in 12-hour or 24-hour-mode (bit 6 at address 09 of the RTC and Control Page). The previously stored bit employs the AM/PM information to decode between the first and second set of 12 hours.

The alarm resisters are stored in address area 0007_h to $000A_h$. The four bits 7 (MS, MM, MH and MD) produce a bitmask which determine the degree of matching that should exist between the alarm and RTC registers (0000_h to 0003_h) for an alarm condition to be set up. For the security's sake, a change in the alarm setup halts the data capture operation and resets the MIP bit.

Special purpose registers

The DS1615 has additional special functions in store besides the standard functions mentioned so far. These are the bits not mentioned so far (all on RTC/CONTROL page):

$\overline{\textbf{EOSC}}$ (address 0Eh, bit 7)
Enable oscillator controls the state of the oscillator in battery back-up mode. When set to 0, the oscillator is active. When set to 1, the oscillator is halted and the DS1615 goes into a low-power standby state with a current drain of less than 100 nA.

CLR (address 0Eh, bit 6)
Clear Enable enables *the clear memory* command. Following the issuing of this command, the CLR bit is also cleared to zero.

AIE (address 0Eh, bit 6)
Alarm Interrupt Enable allows the *alarm flag* (ALMF) to generate an $\overline{\textbf{INT}}$ signal.

MEM CLR (address 14h, bit 0)
Memory Clear indicates that the following memories are cleared: datalog, histogram, temperature alarm, current samples, start time stamp, start delay and sample rate register. When a datalog mission is started, MEM CLR goes to 0.

SIP (address 14, bit 4)
Sample in Progress is active when a measurement value is being processed. This takes up to 750 ns.

LOBAT (address 14h, bit 3)
The *Low Battery Flag* is raised when the Lithium cell reaches a low energy level.

ALMF (address 14h, bit 0)
The *Alarm Flag* when active indicates that the current time matches the time set up in the alarm registers. When then AIE bit is active, $\overline{\textbf{INT}}$ is made low. When you make AIE low, ALMF is also reset.

Serial interface

The DS1615 provides two different types of serial communication: synchronous and asynchronous. In both modes, the LSB is transmitted first, and the MSB, last. The *Communications Select Input* (COMSEL) selects between the two available modes. In synchronous mode, the communication (with $\overline{\textbf{RST}}$ = 1) is via the serial clock line SCLK and the data I/O line. Addresses are supplied first, and data last, when a read or write command is issued. The highest data communication rate is 2 Mbps.

Synchronous communication is selected by pulling COMSEL to ground, or leaving it open. The DS1615 then employs a half-duplex format at a data speed of 9600 bits/s. As customary with UARTs, the protocol includes the transmission of start and stop bits. The data input of the DS1615 is labelled 'RX', the data output, 'TX'.

The asynchronous communication mode employs a CRC (cyclic redundancy check). If the communication fails (for whatever reason), the DS1615 recognises the absence of the stop bit, or the presence of more than 10 bit periods when reception is interrupted. It responds by resetting the communication process.

In synchronous mode, the communication may simply be interrupted by activating the $\overline{\textbf{RST}}$ line.

Commands

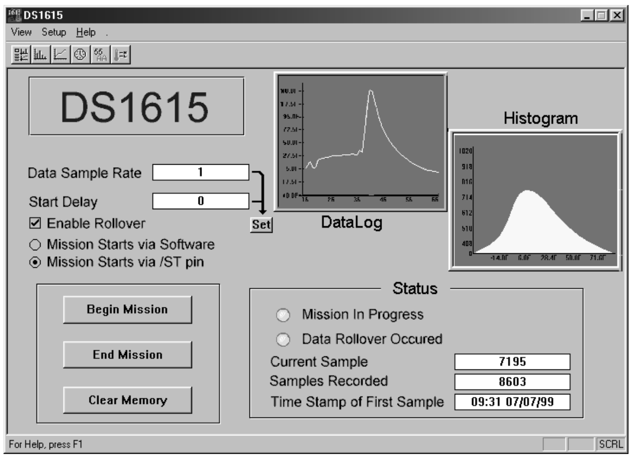
All communication with the DS1615 is accomplished by writing a command to the device followed by *parameter bytes*. There are five commands in all.

WRITE BYTE (22h)
The host writes a byte into the RTC, the control register and the non-volatile user RAM. The write command is followed by the address. Next comes the date byte. Response from DS1615: none.

READ PAGE (33h)
The host sends a read command followed by 16 address bits (MSB first). The DS1615 then supplies the byte contents of the selected page (plus two CRC bytes for data integrity checking). In asynchronous mode, TX becomes inactive after the last register. In synchronous mode, however, outputting continues as long as the serial interface keeps extracting a clock signal.

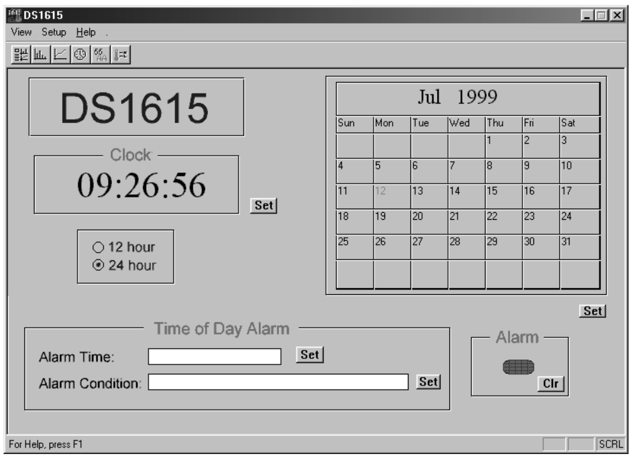
SPECIFICATION TEST (44h)
The host transmits the command, and the DS1615 replies with four low pulses on the $\overline{\textbf{INSPEC}}$ our $\overline{\textbf{OUTSPEC}}$ pin, depending on the measured values being inside the defined range or not.

READ TEMPERATURE (55h)
Prompted by this command from the host, the DS1615 reads the current temperature and stores it in the Current Temperature Register (when MIP is at 0). However, the temperature value is not stored in the datalog memory or the histogram memory. After the command, the TR bit has to be evaluated to determine the end of the conversion.

a


Data Sample Rate: 1
Start Delay: 0
☑ Enable Rollover
○ Mission Starts via Software
◉ Mission Starts via /ST pin

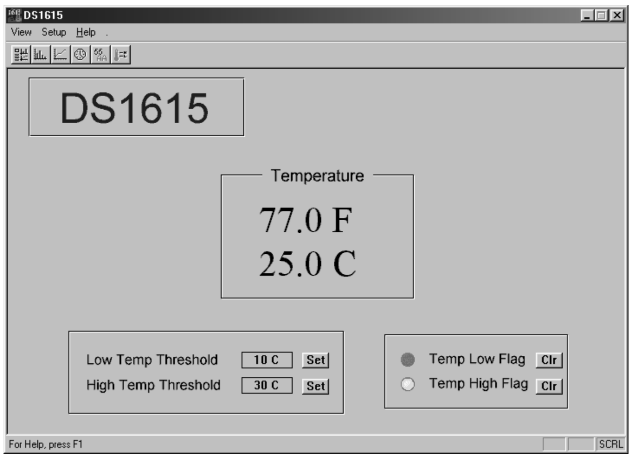
Status
○ Mission In Progress
○ Data Rollover Occured
Current Sample: 7195
Samples Recorded: 8603
Time Stamp of First Sample: 09:31 07/07/99

b


Clock: 09:26:56
○ 12 hour
◉ 24 hour

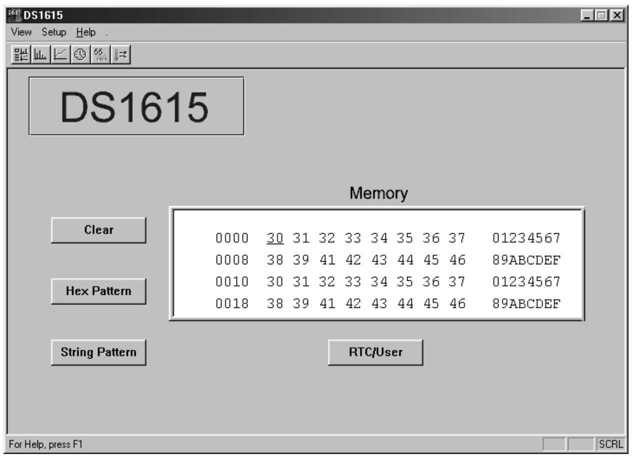
Jul 1999						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Time of Day Alarm
Alarm Time:
Alarm Condition:

c


Temperature
77.0 F
25.0 C

Low Temp Threshold: 10 C
High Temp Threshold: 30 C
● Temp Low Flag
○ Temp High Flag

d


Memory

```
0000  30 31 32 33 34 35 36 37   01234567
0008  38 39 41 42 43 44 45 46   89ABCDEF
0010  30 31 32 33 34 35 36 37   01234567
0018  38 39 41 42 43 44 45 46   89ABCDEF
```

Figure 5. The screendumps prove that the software was written for intuitive control.

CLEAR MEMORY (A5h)
Once this command has been enabled by CLR and its transmission, the following memories are cleared: datalog, histogram, temperature alarm, current samples, start time stamp, start delay and sample rate register.

DS1615 Evaluation Kit

Dallas Semiconductor offer a Evaluation Kit to demonstrate the power and versatility of the DS1615 temperature recorder. The kit allows the registers, memories, date, alarm conditions and temperature limits to be programmed. The kit comprises a small circuit board with a DS1615 on it and connected-up to a sub-D socket, a couple of other components, an RS232 cable and control software for the Windows 95/98 platforms. This software is available on two disks through the *Elektor Electronics* Readers Services. The disks also contain all documentation files for the DS1615 and some source code files.

The circuit diagram in **Figure 3** shows another Dallas IC, the DS275. This chip is included to arrange all serial communication with the PC. Its main function is to convert the symmetrical voltages on the RS232 lines into single-swing TTL/CMOS levels and the other way around. In fact it does practically the same as the ubiquitous MAX232, only much more efficiently because the IC 'steals' energy from the negative excursion of the RX_IN signal, making it available to the TX_OUT pin.

Menus and options

Once the program has been installed it is available for running either through a desktop icon or an entry in the Programs menu.

The main menu appears (**Figure 5a**). This not only provides a quick visual impression of the datalogging mission and the associated histogram, but also the start delay and the start condition (PC or pushbutton). The two diagrams in the window may be enlarged to full size by clicking on the respective buttons in the menu bar. The measurement is launched and closed via the main menu. The number of measurement values, the time stamp of the first measurement and the status of the MIP and RO bits are all shown in the Status window. This window also allows all memories to be cleared.

The **Time and Calendar** menu (**Figure 5b**) allows you to set and read the time format (12/24 h), the calendar and the alarm. The year is shown as two digits (Y2K compatible). The alarm indicator shows you the occurrence of a time/date alarm.

The **Current Temperature/Threshold Set** menu (**Figure 5c**) is mostly self-explanatory. In the status area, the two temperature limits may be defined between –40°C and +85°C in 0.5°C increments. The status window also shows whether or not all collected measurement values are within the user-defined range.

Finally, the **User Memory** menu (**Figure 5d**) allows the user to program and/or edit the non-volatile memory. This memory may be cleared by a single mouse click on the CLEAR button. Programming is done in hexadecimal or using character strings. The first line in the User NV Memory window represents the address range. The next line contains the eight databytes. The corresponding ASCII characters are shown at the right.

(990073-1)

APPLICATION NOTE

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

digitally-controlled potentiometers

with integrated voltage comparators

Although microprocessors are integrated into more and more electronic circuits, there remain requirements for analogue components and devices. One such is the potentiometer, which, until now, has been difficult to emulate by a digital equivalent. Xicor have overcome this difficulty with their X9440 and X9448. These devices combine the flexibility of an analogue product with the power of digital communications.

A Xicor Application

1

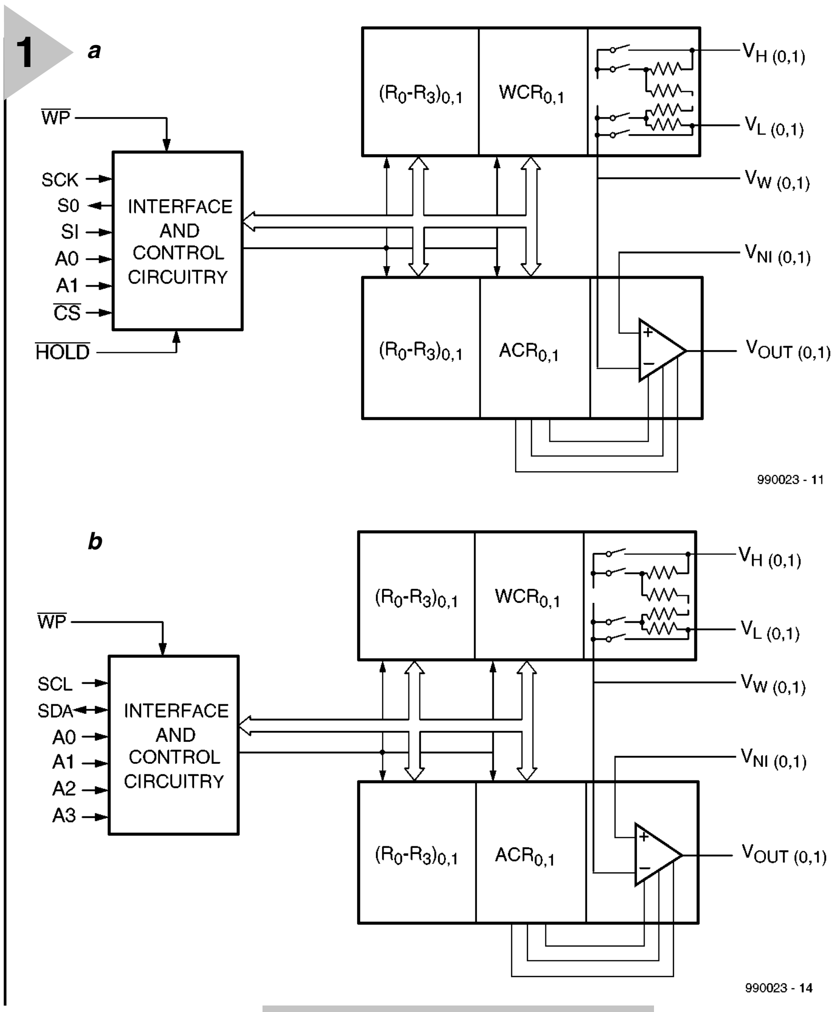


Figure 1. Functional diagram of the X9440 (a) and the X9448 (b). Differences between the two concern mainly the serial interface.

DESCRIPTION

Xicor's X9440 and X9448 integrate two non-volatile (EEPOT) digitally-control-led potentiometers and two voltage comparators on a CMOS monolithic microcircuit.

Both types contain two resistor arrays, each composed of 63 resistive elements. Between successive elements and at either end are tap points accessible to the wiper elements. The position of the wiper element on the array is controlled by the user through the serial bus interface.

Each potentiometer has an associated voltage comparator, which compares the external input voltage, V_{NI},

REGISTER BIT DESCRIPTIONS

Wiper Counter Register (WCR)

0	0	WP5	WP4	WP3	WP2	WP1	WP0
		(volatile)					(LSB)

WP0-WP5 identify wiper position.

Analog Control Register (ACR)

0	0	User-bit5	User-bit4	User-bit3	Latch	Enable	Shut-down
			(volatile)				(LSB)

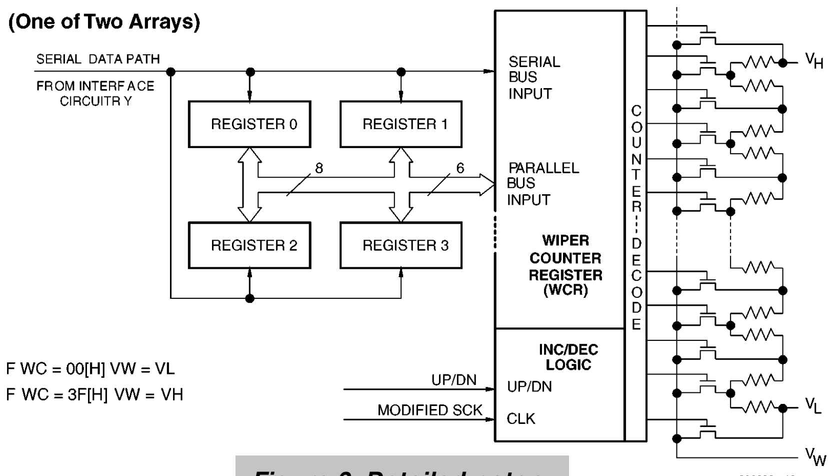


Figure 2. Detailed potentiometer block diagram.

with the wiper voltage, V_W, and sets the output voltage level to a logic high or low.

Each resistor array and comparator has associated with it a wiper counter register (WCR), analogue control register (ACR), and eight 6-bit data registers that can be directly written and read by the user.

The contents of the wiper counter register control the position of the wiper on the resistor array.

The contents of the analogue control register control the comparator and its output.

The potentiometer is programmed via a serial interface (X9440: SPI; X9448: two-wire).

ARRAY DESCRIPTION

The integrated circuits comprise two resistor arrays and two voltage comparators. Each array contains 63 resistive segments that are connected in series. The physical ends of each array are equivalent to the fixed terminals of a mechanical potentiometer (V_H and V_L inputs).

At both ends of each array and between successive resistive segments is a CMOS switch connected to the wiper (V_W) output. Within each individual array only one switch may be turned on at a time. These switches are controlled by a volatile wiper counter register (WCR). The six bits of the WCR are decoded to select, and enable, one of sixty-four switches.

The WCR may be written directly, or it can be changed by transferring the contents of one of four associated data registers into the WCR. These data registers and WCR can be read and written by the host system.

SERIAL INTERFACE

The X9440 supports the SPI interface hardware conventions. The device is accessed via the SI input with data clocked in on the rising SCK. $\overline{\text{CS}}$ must be LOW and the $\overline{\text{HOLD}}$ and $\overline{\text{WP}}$ pins must be HIGH during the entire operation.

The SO and SI pins can be connected together since they have three-state outputs. This may help to reduce system pin count.

The X9448 supports a bidirectional bus oriented protocol. The protocol defines any device that sends data on to the bus as a transmitter and the receiving device as the receiver. The device controlling the transfer is a master and the device being controlled is the slave. The master will always initiate data transfers and provide the clock for both transmit and receive operations. Therefore, the X9448 will be considered a slave device in all applications.

Clock and data conventions (X9448)
Data states on the SDA line can change only during SCL LOW periods (t_{LOW}). SDA state changes during SCL HIGH are reserved for indicating start and stop conditions.

Start condition (X9448)
All commands to the X9448 are preceded by the start condition, which is a HIGH to LOW transition of SDA while SCL is HIGH (t_{HIGH}). The X9448 continuously monitors the SDA and SCL lines for the start condition and will not respond to any command until this condition is met.

Stop condition (X9448)
All communications must be terminated by a stop condition, which is a LOW to HIGH transition of SDA while SCL is HIGH.

Acknowledge (X9448)
Acknowledge is a software convention used to provide a positive handshake between the master and slave devices on the bus to indicate the successful receipt of data. The transmitting device, either the master or the slave, will release the SDA bus after transmitting eight bits. The master generates a ninth clock cycle and during this period the receiver pulls the SDA line LOW to acknowledge that it successfully received the eight bits of data.

The X9448 will respond with an acknowledge after recognition of a start condition and its slave address and once again after successful receipt of the command byte. If the command is followed by a data byte, the X9448 will respond with a final acknowledge.

VOLTAGE COMPARATOR

The comparator compares the wiper voltage, V_W, with the external input voltage, V_{NI}. The comparator and its logic level output are controlled by the Shutdown, Latch, and Enable bits of the Analogue Control Register (ACR).

Enable connects the comparator output to the V_{OUT} pin, Latch memorizes the output logic state, and Shutdown removes the analogue section supply voltage to save power. The ACR is programmed via the SPI (X9448: two-wire) serial interface.

The ACR may be written directly, or it can be changed by transferring the contents of one of four associated data registers into the ACR. These data registers and the ACR may be read and written by the host system.

WCR AND ACR

The X9440 contains two Wiper Counter Registers, WCR, one for each EEPOT potentiometer, and two Analogue Control Registers, ACR, one for each of the voltage comparators. The WCR is equivalent to a serial-in, parallel-out counter with its outputs decoded to select one of sixty-four switches along its resistor array.

The contents of the WCR and ACR can be altered in four ways: they may be written by the host via the Write WCR instruction (serial load); they may be written indirectly by transferring the contents of one of four associated data registers (DRs) via the XFR Data Register instruction (parallel load); they can be modified one step at a time by the Increment/Decrement instruction (WCR only). Finally, they may be loaded with the contents of their data register zero (R0) upon power-up.

The WCR and ACR are volatile registers; that is, their contents are lost when the X9440 is powered-down.

Although the registers are automatically loaded with the value in R0 upon power-up, it should be noted this may be different from the value present at power-down.

Programming the ACR is similar to the WCR. However, the six bits in the WCR position the wiper in the resistor array, whereas three bits in the ACR control the comparator and its output.

DATA REGISTERS

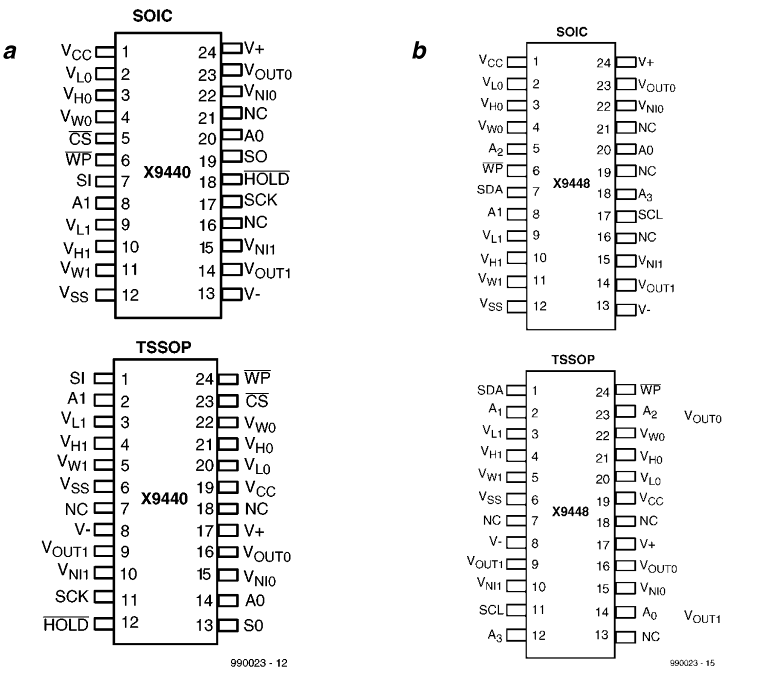
Each potentiometer and each voltage comparator has four non-volatile Data Registers (DR). These may be read or written directly by the host and data can be transferred between any of the four DRs and the WCR or ACR. It should be noted that all operations changing data in one of these registers are non-volatile and will take not more than 10 ms.

If the application does not require storage of multiple settings for the potentiometer or comparator, the registers may be used as regular memory locations for storing system parameters or user preference data.

Detailed datasheets for the X9440 and X9448 may be found on the Internet: **http://www.xicor.com**

[990023]

3



a
SOIC
X9440
1 VCC, 2 VL0, 3 VH0, 4 VW0, 5 CS, 6 WP, 7 SI, 8 A1, 9 VL1, 10 VH1, 11 VW1, 12 VSS
24 V+, 23 VOUT0, 22 VNI0, 21 NC, 20 A0, 19 SO, 18 HOLD, 17 SCK, 16 NC, 15 VNI1, 14 VOUT1, 13 V-
TSSOP
X9440
1 SI, 2 A1, 3 VL1, 4 VH1, 5 VW1, 6 VSS, 7 NC, 8 V-, 9 VOUT1, 10 VNI1, 11 SCK, 12 HOLD
24 WP, 23 CS, 22 VW0, 21 VH0, 20 VL0, 19 VCC, 18 NC, 17 V+, 16 VOUT0, 15 VNI0, 14 A0, 13 S0
990023 - 12
b
SOIC
X9448
1 VCC, 2 VL0, 3 VH0, 4 VW0, 5 A2, 6 WP, 7 SDA, 8 A1, 9 VL1, 10 VH1, 11 VW1, 12 VSS
24 V+, 23 VOUT0, 22 VNI0, 21 NC, 20 A0, 19 NC, 18 A3, 17 SCL, 16 NC, 15 VNI1, 14 VOUT1, 13 V-
TSSOP
X9448
1 SDA, 2 A1, 3 VL1, 4 VH1, 5 VW1, 6 VSS, 7 NC, 8 V-, 9 VOUT1, 10 VNI1, 11 SCL, 12 A3
24 WP, 23 A2, 22 VW0, 21 VH0, 20 VL0, 19 VCC, 18 NC, 17 V+, 16 VOUT0, 15 VNI0, 14 A0, 13 NC
VOUT0
VOUT1
990023 - 15

Figure 3. Pin configuration of the X9440 (a) and the X9448 (b). Both devices are available in 24-lead TSSOP and 24-lead SOIC packages.

PIN DESCRIPTIONS

Host interface pins (X9440)

Serial output (SO)
SO is a push/pull serial data output pin. During a read cycle, data is shifted out via this pin. Data is clocked out by the trailing edge of the serial clock.

Serial input (SI)
SI is the serial data input pin. All opcodes, byte addresses and data to be written to the pots and pot registers are input via this pin. Data is latched by the leading edge of the serial clock.

Serial clock (SCK)
The SCK input is used to clock data into and out of the X9440/X9448.

Chip select ($\overline{CS}$)
When $\overline{CS}$ is HIGH, the X9440 is deselected and the SO pin is at high impedance, and (unless an internal write cycle is underway) the device will be in the standby state. $\overline{CS}$ LOW enables the X9440, placing it in the active power mode.

Hold ($\overline{HOLD}$)
$\overline{HOLD}$ is used in conjunction with the $\overline{CS}$ pin to select the X9440. Once it is selected and a serial sequence is underway, $\overline{HOLD}$ may be used to pause the serial communication with the controller without resetting the serial sequence. To pause, $\overline{HOLD}$ must be made LOW while SCK is LOW. To resume communication, $\overline{HOLD}$ is made HIGH, again while SCK is LOW. If the pause feature is not used, $\overline{HOLD}$ should be held HIGH at all times.

Device address (A_0–A_1)
The address inputs are used to set the least significant two bits of the 8-bit slave address. A match in the slave address serial data stream must be made with the Address input in order to initiate communication with the X9440. A maximum of four devices may share the same SPI serial bus.

Host interface pins (X9448)

Serial clock (SCL)
The SCL input is used to clock data into and out of the X9448.

Serial data (SDA)
SDA is a bidirectional pin used to transfer data into and out of the X9448.

Device address (A_0–A_1)
The address inputs are used to set the least significant four bits of the 8-bit slave address. A match in the slave address serial data stream must be made with the Address input in order to initiate communication with the X9448. A maximum of 16 devices may share the same 2-wire serial bus.

Potentiometer pins

V_H (V_{H0}–V_{H3}), V_L (V_{L0}–V_{L3}) (X9440)
V_H (V_{H0}–V_{H1}), V_L (V_{L0}–V_{L1}) (X9448)
The V_H and V_L inputs are equivalent to the terminal connections on either end of a mechanical potentiometer.

V_W (V_{W0}–V_{W1})
The wiper output, V_W, is equivalent to the wiper output of a mechanical potentiometer and is connected to the inverting input of the voltage comparator.

Comparator and device pins

Voltage input V_{NI0}, V_{NI1}
V_{NI0} and V_{NI1} are the input voltages to the plus (non-inverting) inputs of the two comparators.

Buffered voltage outputs, V_{OUT0}, V_{OUT1}
V_{OUT0} and V_{OUT1} are the buffered voltage comparator outputs controlled by bits in the volatile ACR.

Hardware write protect input, $\overline{WP}$
The $\overline{WP}$ pin when low prevents non-volatile writes to the WCR and ACR.

Analogue supplies, V+ , V–
The analogue supplies, V+ and V– are the supply voltages for the EEPOT analogue section and the voltage comparators.

System supply V_{CC} and ground V_{SS}
The system supply, V_{CC}, and its reference, V_{SS}, are used to bias the interface and control circuits.

PC-controlled model railway: EEDTS Pro

booster amplifier revisited

This short article deals with several matters relating to the booster amplifier described in the July/August issue of this magazine. One of these is an interface for use with an existing Märklin booster amplifier within the EEDTS Pro system. Also, there are a few tips for those who are worried about coupling the power output of the amplifier to the track.

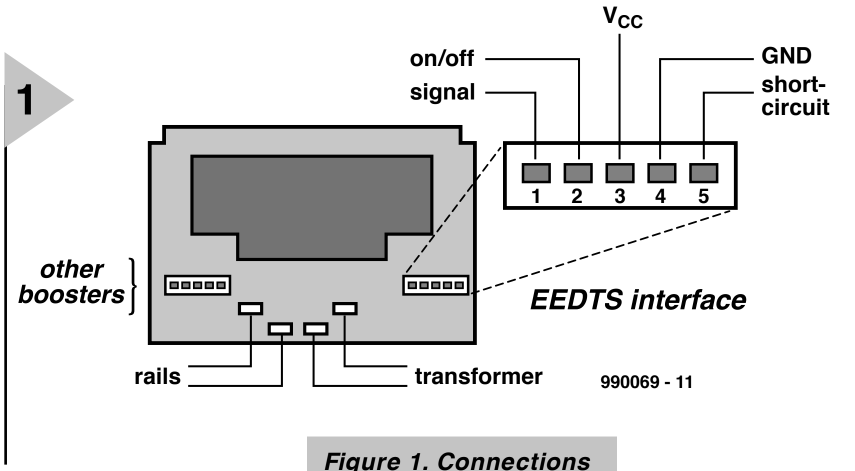


Figure 1. Connections of the Märklin booster.

As elsewhere, compatibility is an important matter in model railway systems. Because of this, it was already stated earlier in this series of articles that the Märklin booster amplifier would be supported. For this purpose, a special interface has been designed that may be linked to one of the 5-way connectors at the back of the Märklin unit.

The back of the Märklin unit is shown in **Figure 1**. The two 5-way connectors are identical so that it does not matter which of the two is used. The controller of the EEDTS Pro system has a signal output at TTL level which is intended for the booster amplifier. This signal, available at pin 3 of K_8 (see Part 1 in our June 1999 issue), is linked directly to pin 1 of the Märklin unit. In case of a short-circuit, this signal need not be decoupled as is the case with the EEDTS unit.

The GND connection of the controller (pin 2 of K_8) is continued to the GND terminal (pin 4) of the booster amplifier. This results in the interface being powered when the controller is switched on.

In normal operation, pin 5 is at low TTL level, which is passed on to the interface. When a short-circuit occurs, this situation no longer prevails.

CIRCUIT DESCRIPTION

The diagram in **Figure 2** shows that the interface is based on three Schmitt trigger inverters. Connector K_1 is linked to K_8 on the controller and connector K_2 to the Märklin booster amplifier via a four-core cable.

When the power supply is switched on, capacitor C_1 is discharged. This forces R-S bistable IC_{1b}-IC_{1c} to assume a defined status: if the output of IC_{1b} is low, the output of IC_{1c} is high. After a while, C_1 will have been charged via R_3, but diode D_2 ensures that this has no effect on the circuit. Resistor R_4 ensures that when the supply is switched off, C_1 will be discharged reasonably quickly so that the capacitor can fulfil its function every time the supply is switched on.

The initial status of the R-S bistable results in pin 2 of the booster amplifier being linked to a low level, so that the booster amplifier assumes the idle mode so ensuring that there is no output signal.

When GO key S_2 is pressed, the bistable changes state, resulting in diode D_1 lighting and pin 2 of the booster connector going high. This actuation of the booster amplifier is indicated by the lighting of an LED in the booster itself.

When a short-circuit occurs, pin 5 of the booster connector, and of K_2, of course, changes from low to high. This is converted by IC_{1a} into a negative pulse, which has the same effect as if STOP key S_1 were operated: the booster amplifier is switched off. Since the short-circuit monitoring is effected by a wired-OR gate, all booster amplifiers used will be switched off when a short-circuit occurs.

When the cause of the short-circuit has been remedied or removed, the system may be re-actuated with the GO key.

PROTECTION TIPS

Some model railway enthusiasts do not like applying the full power of the booster amplifier to the railway track. They get visions of seized wheels, fire hazard, and other risks and inconveniences. Although the short-circuit protection of the EEDTS booster amplifier keeps the likelihood of such disasters to an absolute minimum, there is always a very small risk that a short-circuit current of some 10 A may flow.

2

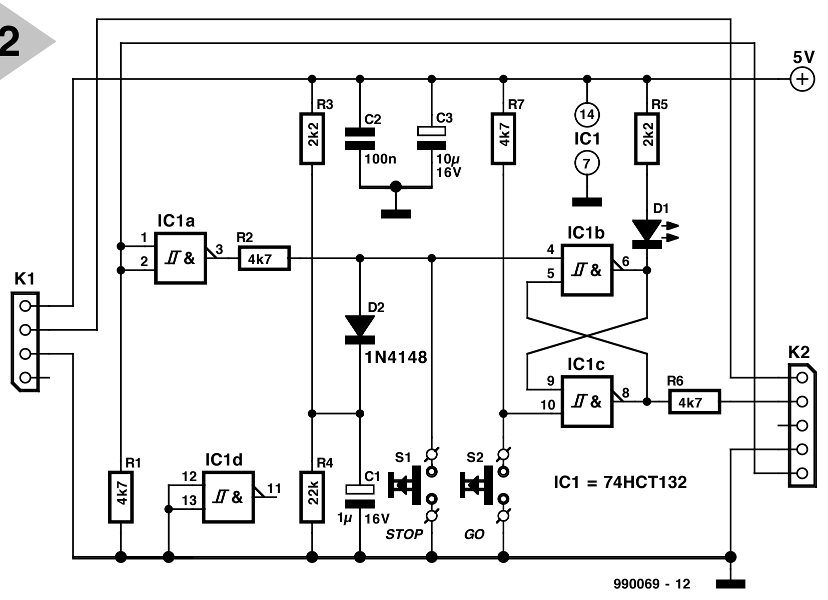


Figure 2. Circuit diagram of the requisite interface.

Although unlikely, this risk may be obviated by splitting the output of the booster amplifier into five independent branches, each of which is protected by a 2 A fuse—see **Figure 4**. There are neat thermal fuses with integral reset knob available for this purpose. The railway system is then powered by these five branches, which are all isolated from one another. This may be effected by isolating the central rail or, in case of a two-rail system, one of the rails.

However, the best solution for efficient current control is shown in **Figure 5**. In this, the supply is provided by two separate booster amplifiers, each of which supplies half the requisite current. Halving the output current of an EEDTS booster amplifier is effected simply by omitting T_2, T_4, R_{13}, and R_{15}. The current limiting then starts at about 5 A.

As shown in Figure 5, each booster needs its own booster interface, although in the second one switches S_1 and S_2 may be omitted.

If more than two booster amplifiers are to be used, the requisite additional booster interfaces may be interlinked via terminals A, B, and C. The number of booster amplifiers that can be used is virtually unlimited.

As Figure 5 shows, a number of booster amplifiers may be powered by a single, heavy-duty supply, or they may have their own independent power supply. A 5 A supply requires a 150 VA transformer and two 10,000 μF electrolytic capacitors. The return lines of the supplies must, of course, be bonded together.

Parts list

Resistors:
R_1, R_2, R_6, R_7 = 4.7 kΩ
R_3, R_5 = 2.2 kΩ
R_4 = 22 kΩ

Capacitors:
C_1 = 1 μF, 16 V, radial
C_2 = 0.1 μF
C_3 = 10 μF, 16 V, radial

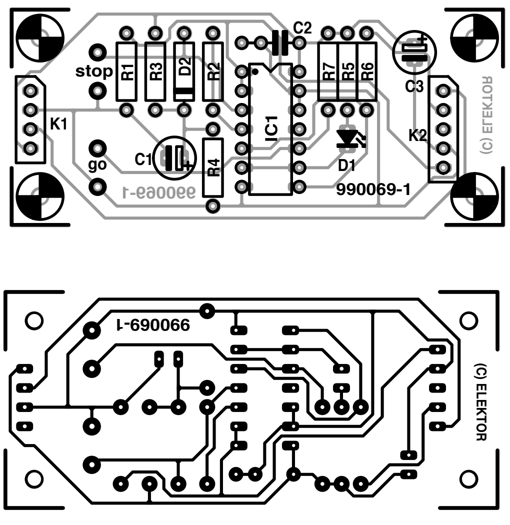
Semiconductors:
D_1 = low current LED
D_2 = 1N4148

Integrated circuits:
IC_1 = 74HCT132

Miscellaneous:
K_1= 4-way SIL header
K_2 = 5-way SIL header
S_1, S_2 = single-pole push button switch

Figure 3. Printed-circuit board for the Märklin booster interface.

3



4

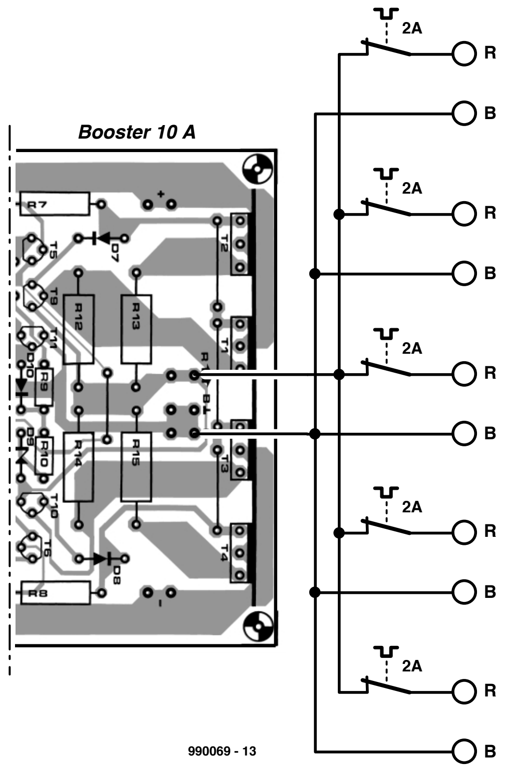


Figure 4. Dividing the booster amplifier output into five independent branches.

CONSTRUCTION

The interface is best built on the printed-circuit board shown in **Figure 3**. A small piece of prototyping board may also be used, of course. In either case, construction should not take more than an hour. As always, it is, of course, essential to check and inspect the finished unit thoroughly before it is taken into use.

Note that it is imperative that the controller is switched on before the Märklin booster amplifier(s).

[990069]

5

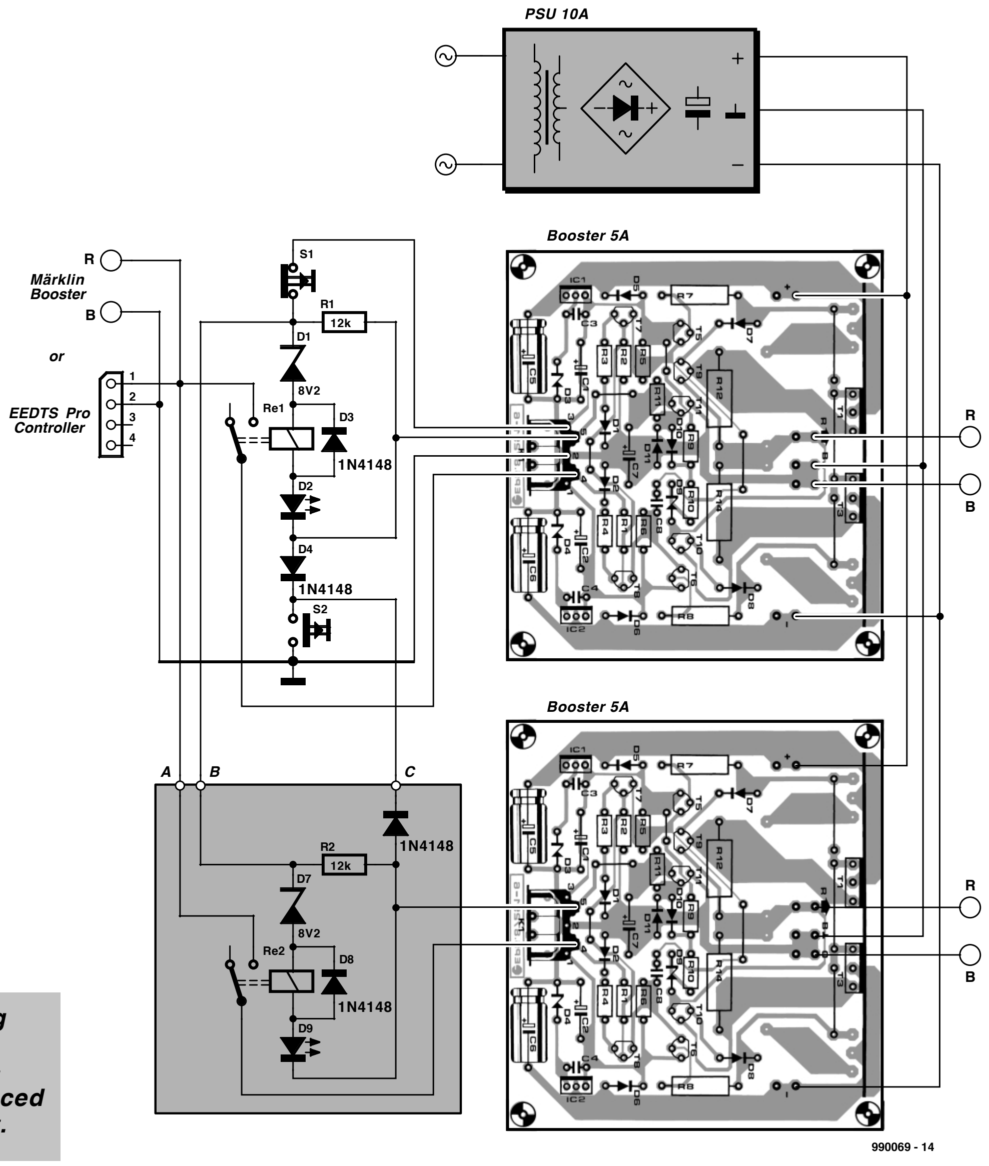


Figure 5. Using two EEDTS booster amplifiers with reduced output current.

stepper motor control

part 1: stepper motor construction, function and control

Stepper motors have been with us since the early sixties and their significance has grown enormously over the past few years. They are used as driving mechanisms in clocks and other pointer-based instruments, in printers and plotters, various tools and, of course, in robots.

By K.-H. Domnick

1

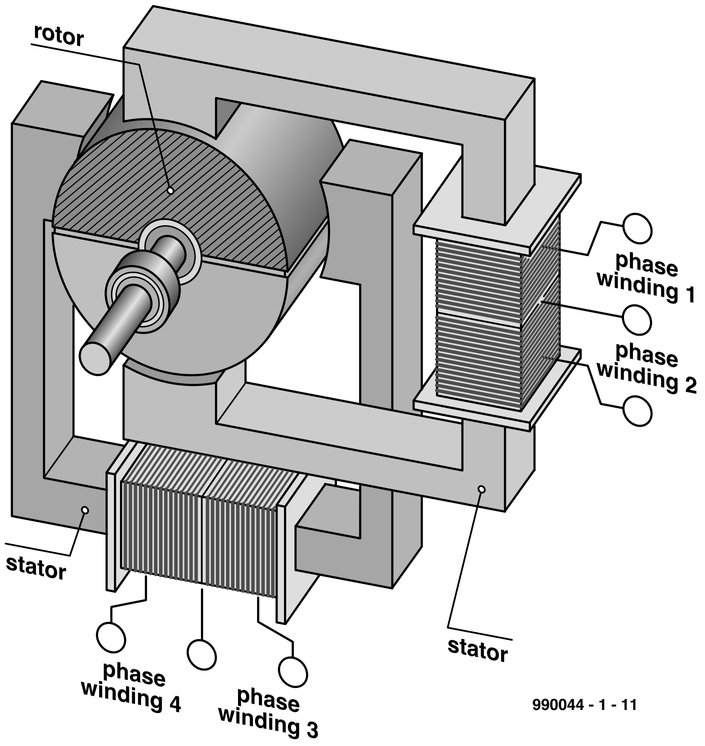


Figure 1. Model of an ideal stepper motor with a 2-pole rotor and two phase windings arranged at an angle of 90 degrees.

Just like almost any other electrically powered motor, a stepper motor consists of a fixed stator and a moving rotator. However, while the rotor (and sometimes the stator) is fitted with electromagnets in dc motors, the stepper motor has fixed electromagnets only. The rotor, consisting of non-magnetic soft iron or a permanent magnet, rotates as a result of the field reversals brought about by the stator magnets. So, before we know just how a stepper motor works, we've discovered an important advantage: there is no electricity supply to the rotor; consequently, the motor is brushless and therefore practically free of wear and tear.

The first (inexpensive) stepper motors were **reluctance** types with rotors consisting of cogged wheels made from soft iron. Disregarding the poor torque developed by these motors, reluctance motors lacked 'rest' positions of the shaft because the soft iron did not in itself act as a magnetic pole. This shortcoming was overcome with the introduction of stepper motors employing permanent magnets. Although the permanently magnetic rotor did allow rest positions, these occurred only at relatively large step angles. This shortcoming was caused by the limited number of **magnetic poles** that could be arranged on the radially-magnetised cylindrical rotor. None the less, this type of stepper motor provides a good starting point for our purpose.

Figure 1 shows a stepper motor in its most rudimentary form. The core is a single magnet (i.e., having just two poles), and there are two **phase windings** arranged at an angle of 90 degrees. When a current flows in one of the phase windings, a magnetic field is established. The rotor then turns its magnetic poles into a direction (or position) in which (1) the smallest air gap is established between the pole and the actuated phase winding, and (2) the largest magnetic flux density occurs (rule: opposite poles attract).

By reversing the current flow, the rotor can be made to turn to one of four different positions, with the rotational direction corresponding to the order in which the polarity changes occur. This so-called **wave drive mode** is shown schematically in **Figure 2**.

Another method of making the motor spindle turn is to energise the two phase windings in **normal mode**. This mode is marked by the presence of two pairs of two identical phase windings arranged next to each other. The resulting rotor action is easily explained using **Figure 2b**.

A **sequence** is one full 'electrical' revolution of 360° (electrical step angle), which is required to perform a complete mechanical step angle. With the two previously explained **full-step** modes, a sequence consists of four clock pulses. In the example, a mechanical step angle equals a complete spindle rotation of 360°.

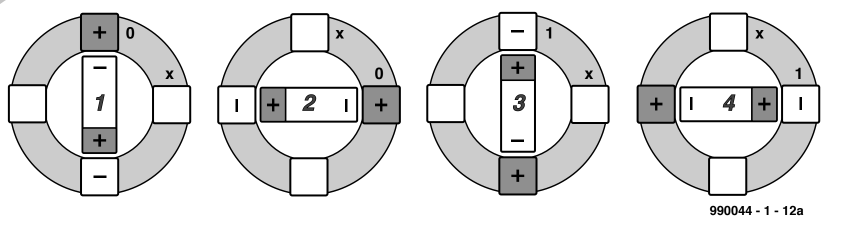
However, there can be no objection against combining the wave drive and normal modes **into half-step** control. This mode allows 'quasi' intermediate steps to be inserted, which is a free means of doubling the motor resolution (steps per revolution). In this mode, the supply is alternately connected to one or two phase windings, so that a sequence consists of eight clock pulses.

Depending on the actual construction of the phase winding coils, two more control techniques are used. **Unipolar** operation is achieved by adding just one switch (**Figure 3a**). This however requires the coils to have centre taps. Also, because of the reduced coil currents, both the torque of the motor and its spindle speed are relatively low. **Bipolar** operation as illustrated in **Figure 3b** was not possible until the arrival of integrated and inexpensive stepper motor drivers. In this mode of operation, both windings are reverse polarised end-to-end, which calls for two switches instead of one.

Stepper motors with a small number of phase windings will typically 'jerk' at low frequencies, even in half-step mode. Further improvements are, however, possible by gradually increasing and decreasing the coil current instead of simply switching it on and off. This so-called **microstep operation** guarantees smooth spindle movement. On the down side, both **torque** and **positioning accuracy** are reduced. These unwelcome side effects are particularly noticed with stepper motors having relatively few rotor positions.

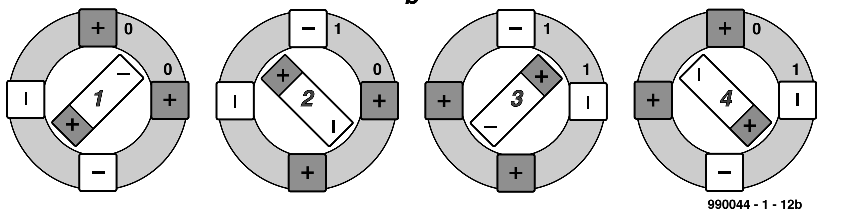
2

a



turn cw					turn ccw				
Clock	Phase 1		Phase 2		Clock	Phase 1			
	Polarity	Current	Polarity	Current		Polarity	Current	Polarity	Current
1	0	1	x	0	1	0	1	x	0
2	x	0	0	1	2	x	0	1	1
3	1	1	x	0	3	1	1	x	0
4	x	0	1	1	4	x	0	0	1

b



turn cw					turn ccw				
Clock	Phase 1		Phase 2		Clock	Phase 1		Phase 2	
	Polarity	Current	Polarity	Current		Polarity	Current	Polarity	Current
1	0	1	0	1	1	0	1	0	1
2	1	1	0	1	2	0	1	1	1
3	1	1	1	1	3	1	1	1	1
4	0	1	1	1	4	1	1	0	1

turn cw					turn ccw				
Clock	Phase 1		Phase 2		Clock	Phase 1		Phase 2	
	Polarity	Current	Polarity	Current		Polarity	Current	Polarity	Current
1	0	1	X	0	1	0	1	X	0
2	0	1	0	1	2	0	1	1	1
3	X	0	0	1	3	X	0	1	1
4	1	1	0	1	4	1	1	1	1
5	1	1	X	0	5	1	1	X	0
6	1	1	1	1	6	1	1	0	1
7	X	0	1	1	7	X	0	0	1
8	0	1	1	1	8	0	1	0	1

Figure 2. Coil current distribution in (a) wave drive, and (b) normal operation. Half-step operation is obtained by combining these two full-step modes.

In practice

So far we've based our discussion on a stepper motor model that does not exist in practice. Modern **hybrid stepper motors** employ axially and permanently magnetised discs as a kind of core. These discs are fitted with cogwheels mutually offset by half a cog width, so that North and South poles alternate. The photograph in **Figure 4** shows the innards of a hybrid stepper motor. The toothed structure of the rotor is clearly recognisable.

The **step angle** or **resolution** of the motor not only depends on the number of pole pairs, that is, the North and South pole cogs on the rotor, but also on number of individually controllable phase windings. In practice, the number of poles is between two and five to keep wiring and circuit complexity within reason. If a specific application calls for high torque then two-pole

3

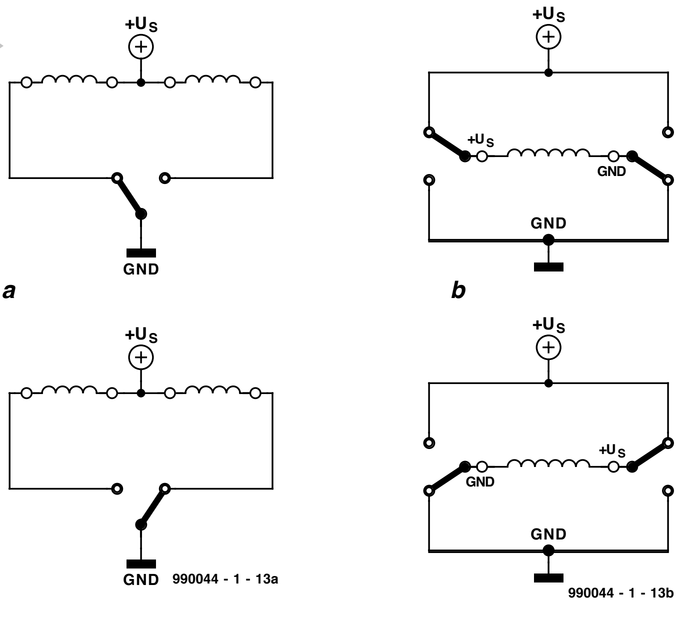


Figure 3. In unipolar mode (a) each motor coil should have a centre tap. Bipolar motors (b) require coil current drive using two switches per winding.

4

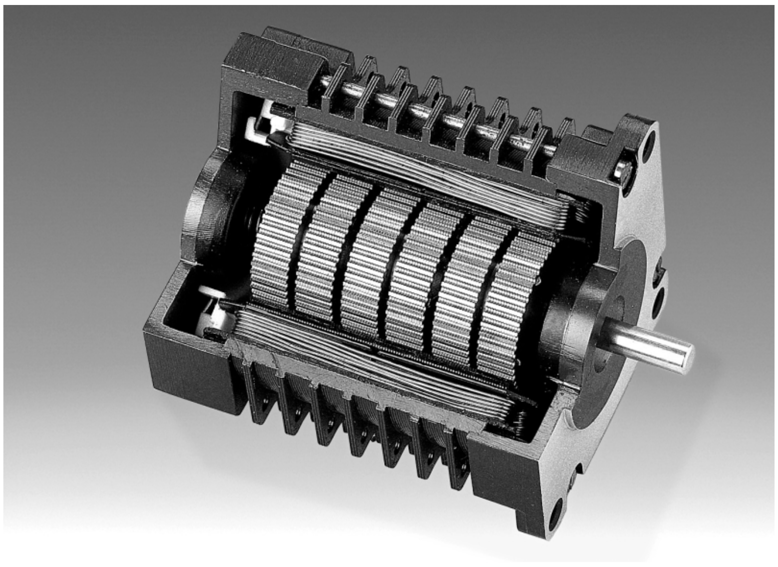
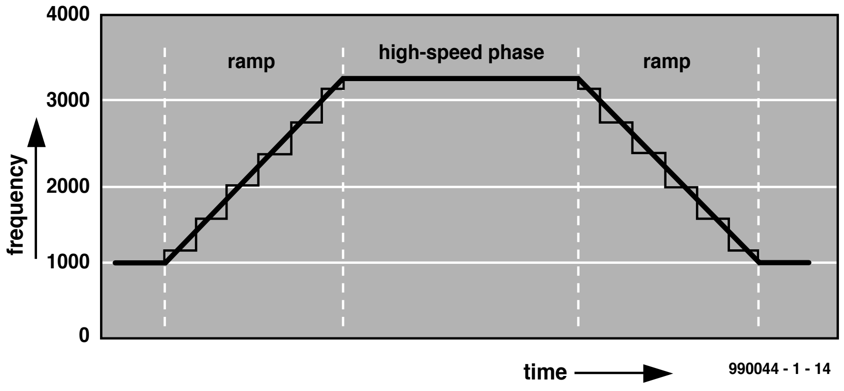


Figure 4. Photograph showing the cogs on the rotor.

5

Figure 5. Acceleration and deceleration (braking) is achieved by ramp-shaped control currents.



stepper motors will be preferred. If, on the other hand, smooth spindle action is the foremost requirement, a five-pole motor in microstep mode will be the best choice. For cases 'in between', a three-pole motor represents a good compromise.

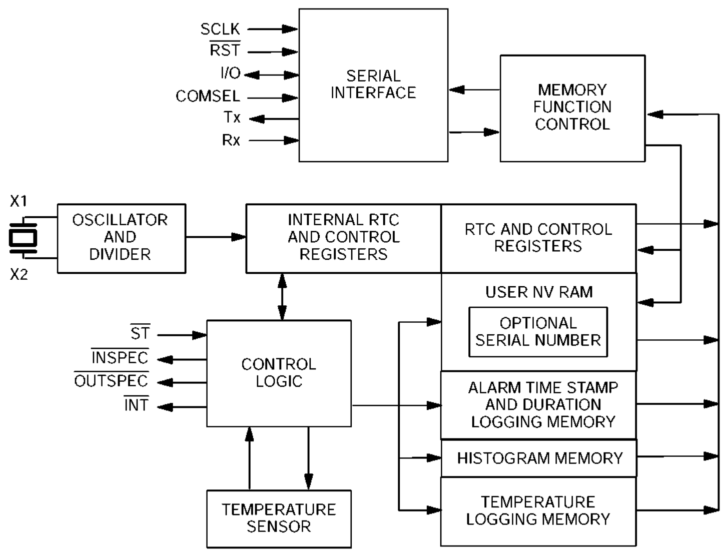
Most of today's stepper motors have a resolution of at least 24 steps (15°) or 48 steps (7.5°) per spindle revolution. To guarantee accurate positioning of the read/write head, most older hard-disk drives contain stepper motors with a resolution of 200 steps (1.8°) or even 400 steps (0.9°).

When a clock frequency of several kilohertz is applied the motor will typically not operate at all because the rotor's inertia prevents it from keeping pace with the rapidly rotating stator field. The use of a **start/stop frequency** which, depending on the motor type, lies between 50 Hz and 2000 Hz, guarantees reliable starting of the motor. Once the motor runs, the clock frequency may be stepped up. Although there is no lower limit to motor acceleration, there will be a definite upper limit to observe. Exceeding the highest possible acceleration rate may cause the motor to stall, just as with any attempt to exceed the highest possible clock frequency.

The increase from the start/stop frequency, via the acceleration phase to the nominal speed is best described by a ramp-shaped curve like the one shown in **Figure 5**. Hence the term **acceleration ramp**. Likewise, the deceleration and the eventual switching off of the motor should also follow a ramp, this time a falling one. Abruptly switching off the coil currents would cause the rotor to keep turning because of inertia, and it would not be possible for the control system to tell the spindle position. Motor control without ramp-shaped coil currents and using clock frequencies low enough for reliable turning in both directions is only possible (and allowed) during zero calibration and slow operation.

Computer control is soon called for when it is required to make the motor turn in half-step or microstep mode while providing for the proper acceleration and deceleration slopes of the coil currents. These functions are admirably handled by the 80C166 microcontroller system described in next month's instalment.

(990044-1)



DS1615 Block diagram.

interrupt-enable bit is set.

$\overline{\text{INSPEC}}$ (Open Drain In-Specification Output)
This pin, in conjunction with the $\overline{\text{OUTSPEC}}$ pin, is used to signal the status of the operation and data of the DS1615.

$\overline{\text{OUTSPEC}}$ (Open Drain Out-of-Specification Output)
This pin, in conjunction with the $\overline{\text{INSPEC}}$ pin, is used to signal the status of the operation and data of the DS1615.

$\overline{\text{ST}}$ (Start/Status Button Input)
The $\overline{\text{ST}}$ pin provides two functions. First, when enabled as the datalog start source (SE bit in Control register is a logic 1), the $\overline{\text{ST}}$ pin is used to instruct the DS1615 to begin recording temperature data based on the programmed start delay and data sample rate. The $\overline{\text{ST}}$ pin must be held low for at least 0.5 seconds for a datalog mission to begin. An external pull-up resistor should be connected to this pin.
Secondly, the $\overline{\text{ST}}$ pin can be used to poll the status of the recorded data. After datalogging has begun, the $\overline{\text{ST}}$ pin instructs the DS1615 to report the status of the recorded data via the $\overline{\text{INSPEC}}$ and $\overline{\text{OUTSPEC}}$ pins.

X1, X2
Connections for a standard 32.768 kHz quartz crystal, Daiwa part number DT-26S or equivalent. For greatest accuracy, the DS1615 must be used with a crystal that has a specified load capacitance of 6 pF. There is no need for external capacitors or resistors. Note: X1 and X2 are very high impedance nodes. It is recommended that they and the crystal be guard-ringed with ground and that high frequency signals be kept away from the crystal area. For more information on crystal selection and crystal layout considera-

DS1615
Temperature Recorder

Manufacturer
Dallas Semiconductor
Website: *www.dalsemi.com*

Features
- Digital thermometer measures temperature –40°C to + 85°C in 0.5°C increments (–40°F to + 183.2°F in 0.9°F increments)
- Digital thermometer provides ±2°C accuracy
- Real Time Clock/Calendar in BCD format counts seconds, minutes, hours, date, month, day of the week, and year with leap year compensation (Y2K compatible)
- Automatically wakes up and measures temperature at user-programmable intervals from 1 to 255 minutes
- Logs up to 2048 consecutive temperature measurements in read-only nonvolatile memory
- Records long-term temperature histogram in 63 bins with 2.0°C resolution
- Programmable temperature-high and temperature-low alarm trip points
- Two serial interface options: synchronous and asynchronous
- 3-wire synchronous serial interface
- Asynchronous serial interface compatible with standard UARTs
- Memory partitioned into 32 byte pages for packetizing data
- On-chip 16-bit CRC generator to safeguard data read operations in asynchronous communications mode
- Optional unique, factory lasered and tested 64-bit serial number

Application example
Temperature Recorder,
Elektor Electronics September 1999

Description
The DS1615 is an integrated temperature recorder that combines a real time clock with temperature data logging and histogram capabilities. It has been designed for applications that require temperature profiling over a given period of time. A programmable sampling rate feature makes the device ideal for applications requiring temperature monitoring over short or long time frames. The integrated Real Time Clock (RTC) provides seconds, minutes, hours, day, date, month, and year information with leap year compensation and also provides an alarm interrupt.
Temperature measurement is provided via integrated thermal technology which can measure temperatures from –40°C to + 85°C in 0.5°C increments.
The DS1615 is a powerful data recording device, providing both a datalog of sampled temperature values over time and a histogram of temperature. The datalog function simply samples the temperature at a user defined sample rate and writes the data to the Temperature Datalog memory. Up to 2048 datalog samples may be recorded. Histogram functionality is implemented by sampling the temperature and then incrementing the count value in a data bin associated with that temperature.
The DS1615 provides 63 two-byte data bins in 2°C increments. The user can program data sampling for both data logging and for histogram tabulation at intervals ranging from once per minute to once every 255 minutes.
The DS1615 also supports programmable high and low temperature alarm trip points that allow the device to monitor whether the temperature stays within desired limits. The device can drive an interrupt or status pin if the temperature falls outside of the programmable limits.
The DS1615 can be programmed to begin sampling data via a pushbutton input or via a command sent over the serial interface with a host machine.
The DS1615 also provides an optional 64-bit serial number which is useful for product identification and tracking.

Overview
The block diagram shows the relationship between

the major control and memory sections of the DS1615. The device has five major data components:

① Real Time Clock and control block,

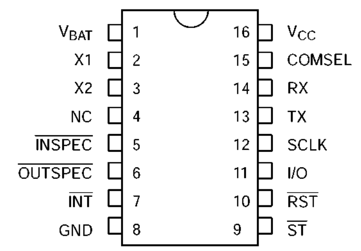
② 32-byte User NVRAM with optional 64-bit lasered serial number,

③ 96 bytes of Alarm event/duration memory,

④ 128 bytes of histogram RAM, and

⑤ 2048 bytes of datalog memory. All memory is arranged in a single linear address space.

PIN ASSIGNMENT



DS1615 16±PIN DIP
DS1615S 16±PIN SOIC (300 MIL)
993007 - 11

Signal descriptions

V_{CC}
V_{CC} is a + 5V input supply. Communication with the DS1615 can take place only when V_{CC} is connected to a + 5V supply.

V_{bat}
Battery input for standard lithium cell or other energy source. All functions of the DS1615 with the exception of the serial interface circuitry are powered by V_{bat} when $V_{CC} < V_{bat}$. All functions are powered by V_{CC} when $V_{CC} > V_{bat}$. If a battery or other energy source is not used, the V_{bat} pin should be connected directly to GND.

GND
Ground

COMSEL (Communication Select Input)
This pin determines whether serial communication is asynchronous or synchronous. When pulled high to V_{CC}, communication is synchronous and will take place via the SCLK, I/O, and $\overline{RST}$ pins. When COMSEL is tied to ground, asynchronous communication utilizing the TX and RX pins is selected. If this pin is floated, the DS1615 will operate in the asynchronous communications mode since the COMSEL pin has a weak internal pull-down resistor.

Tx (Transmit Output)
Transmit output of the asynchronous serial interface. Tx is tri-stated whenever $V_{CC} < V_{bat}$.

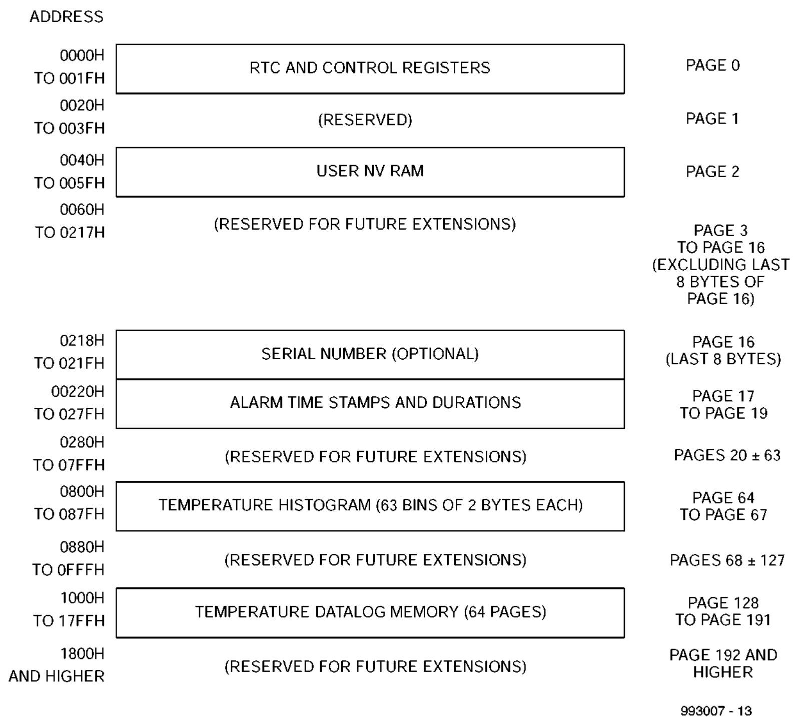
Rx (Receive Input)
Receive input of the asynchronous serial interface.

SCLK (3-Wire Serial Clock Input)
The SCLK pin is the serial clock input for the 3-wire synchronous communications channel.

I/O (3-Wire Input/Output)
The I/O pin is the data Input/Output signal for the 3-wire synchronous communications channel.

$\overline{RST}$ (3-Wire Reset Input)
The $\overline{RST}$ pin is the communications reset pin for the 3-wire synchronous communications channel.

$\overline{INT}$ (Interrupt Output) – The $\overline{INT}$ pin is an open drain active low output that can be connected to an interrupt input of a microprocessor. The $\overline{INT}$ output remains low as long as the status bit causing the interrupt is present and the corresponding



DS1615 Memory map.

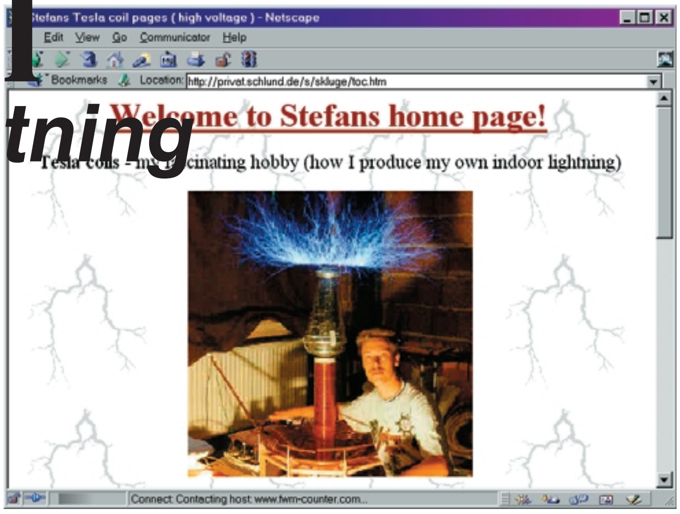
tions, please consult Application Note 58, Crystal Considerations with Dallas Real Time Clocks.

NC (No Connect)
This pin should be left unconnected.

Memory
The memory map shows the general organization of the DS1615. As can be seen in the figure, the device is segmented into 32 byte pages. Pages 0 and 1 contain the Real Time Clock and Control registers. The User NV RAM resides in page 2. Pages 17 to 19 are assigned to storing the alarm time stamps and durations. The temperature histogram bins begin at page 64 and use up four pages.
The temperature logging memory covers pages 128 to 191. Memory pages 1, 3 to 16, 20 to 63, 68 to 127, and 192 and up are reserved for future extensions.

the Tesla coil

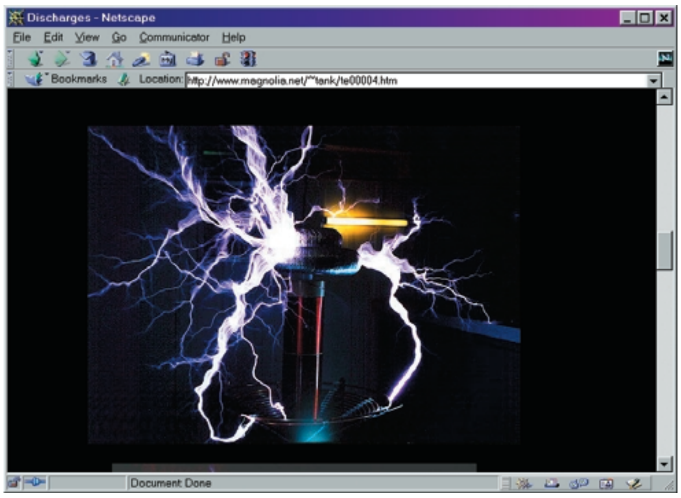
make your own lightning

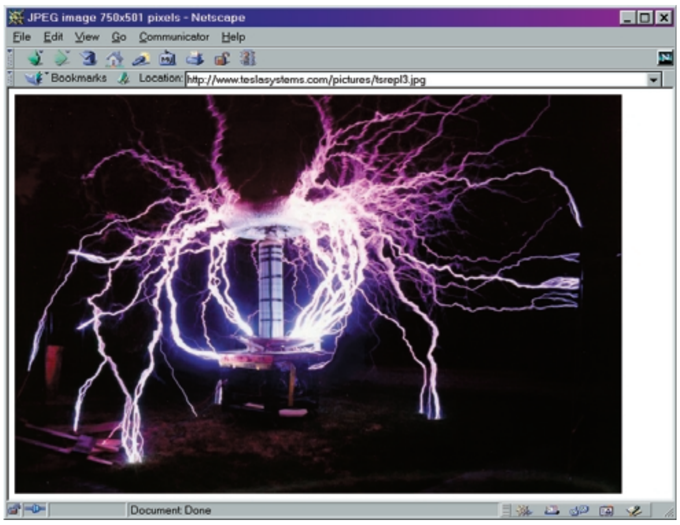

Stefans Tesla coil pages (high voltage) - Netscape
File Edit View Go Communicator Help
Bookmarks Location: http://privat.schlund.de/s/skluge/toc.htm
Connect: Contacting host: www.fwm-counter.com...

Welcome to Stefans home page!

Tesla coils - my fascinating hobby (how I produce my own indoor lightning)

It is unlikely that Nikola Tesla, born in 1856, could have foreseen the widespread interest in his inventions, more than one hundred years later. The Tesla Coil in particular is the subject of many an Internet site.


Discharges - Netscape
File Edit View Go Communicator Help
Bookmarks Location: http://www.magnolia.net/~tank/te00004.htm
Document: Done


JPEG image 750x501 pixels - Netscape
File Edit View Go Communicator Help
Bookmarks Location: http://www.teslasystems.com/pictures/tsrepl3.jpg
Document: Done

The idea is simple, really. Make a transformer with a huge step-up ratio, (secondary winding has far more turn than primary winding), and tune both windings to resonance at the same frequency. If, under these conditions, a pulse is applied to the primary winding (for example, by means of a spark), the voltage across the secondary will rise to level at which substantial discharges can be observed, usually in the form of light flashes, arcs and coronas.

Meanwhile, it appears that many hobbyists find pleasure in building such a Tesla transformer, and use it to create wonderful spark showers. Indeed, it has become something like a sport to 'draw' the largest artificial flashes of lightning. Tesla's own record, however, has not yet been broken or even equalled — in 1899, his experimental setup in Colorado Springs produced an arc of 41 metres (approx. 125 ft), wrecking the electricity supply in the whole area.

As an indication of the popularity of the Tesla transformer, there are no fewer than 120 websites connected to the **Tesla Coil Webring** (*http://www.webring.org/cgi-bin/webring?ring=TeslaRing&list*). Alternatively, go to *http://www.webring.org*, and enter the keyword 'tesla'. The site at *http://www.pupman.co*, also has a long list of Tesla-related URLs.

From the long lists we picked some examples for those of you whose curiosity has been aroused. On **Stefan's Tesla pages** at *http://privat.schlund.de/s/skluge/toc.htm* we found an extensive description of the structure of a Tesla coil, complete with lots of tables, diagrams and equations. If you want something a little more practical, we'd say the explanations on **Guy's Tesla Page** (*http://www.pages.vossnet.de/wilson/tesla1.htm*) are a better starting point.

There are also tools available on the Web for the calculation of a Tesla coil. A good collection is found on the **JavaScript Tesla Coil Calculator Page** at *http://www.geocities.com/CapeCanaveral/Hangar/3108/calculat/html*. Provided you enter the right design data everything is calculated for you.

Many sites may be found that show pictures of home-made Tesla coils and the results obtained from them. Great fun to look at! Some nice examples are **Inonized Ether** at *http://www.magnolia.net/~tank/tesla.htm*, and **Chuck Corran's Tesla Coil page** at *http://www.execpc.com/~ccurran/*

Meanwhile a couple of companies have also entered the field. The activities are mostly construction and demonstration. A typical example is **Tesla Systems Research** at *http://www.teslasystems.com*. Very spectacular!

If you want to know more about the great inventor himself, lots of relevant information may be found at *http://members.xoom.com/_XOOM/tastraum/tesla.html* and *http://www.bena.com/lucidcafe/library/96jul/tesla.html*.

(995070-1)