ELEKTOR ELECTRONICS

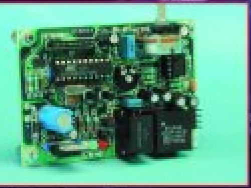
THE ELECTRONICS & COMPUTER MAGAZINE

NOVEMBER 2002
£3.20

www.elektor-electronics.co.uk

Car Diagnostics Adapter

for the OBD-2 standard



Telephone Baby Monitor

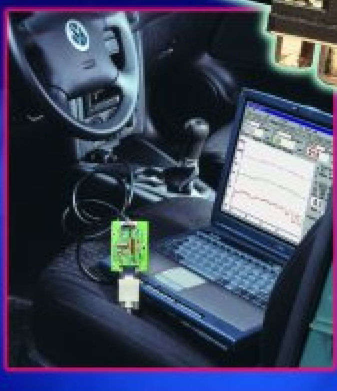
EPROM Emulator

Speed Cameras

Stand-Alone EEDTs Pro

LED Torch

Fuel Cells

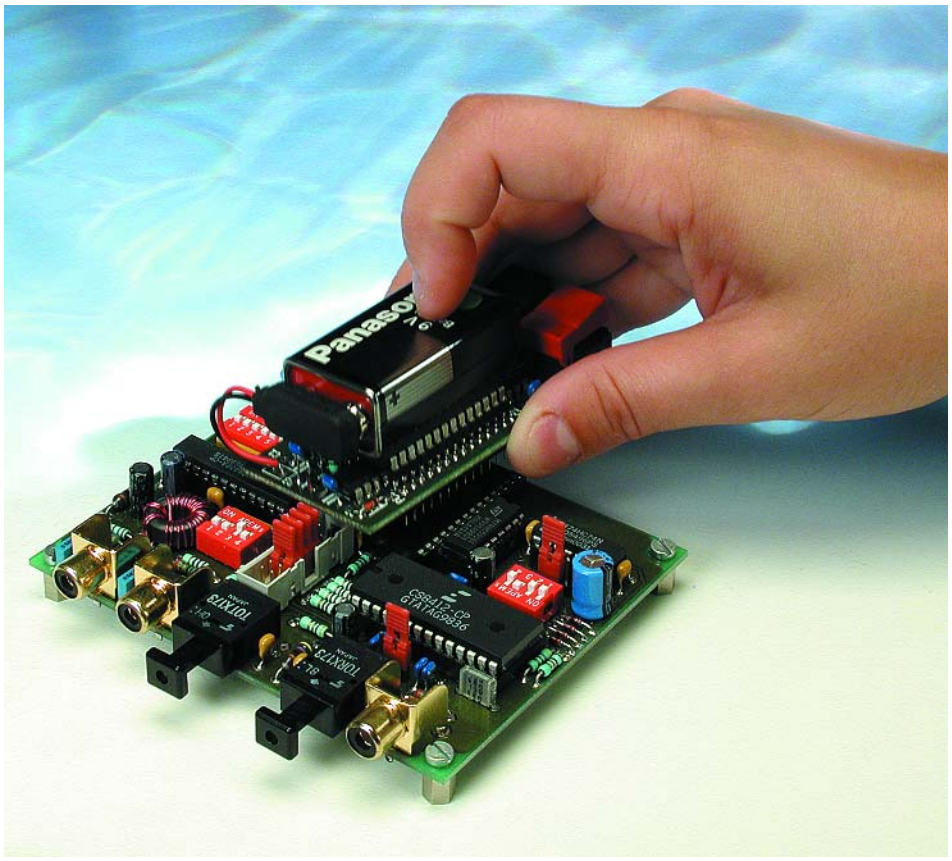


EPROM Emulator

a perfect imitation-EPROM

Design by P. Goossens

As opposed to most other EPROM emulators, the design discussed in this article provides a virtually perfect imitation of a ‘real-life’ 27C256 device, which may be programmed in any programmer as well as inserted into the target system.



An EPROM emulator is a useful if not indispensable tool if you are into developing software for a system employing an EPROM for storage of executable code or fixed data. Such a ‘programmable imitation EPROM’ obviates the need to erase an EPROM after any programming cycle, before the new (and hopefully improved) program may be loaded into the chip. Erasing an EPROM with the aid of UV light usually takes about 20 minutes, which, many of you will agree, is an annoyingly long time. If you want to test a new version of the software during those 20 (long) minutes, a second EPROM may be useful, but the entire operation is still cumbersome and tedious.

Over time, many different EPROM emulators have appeared on the market. *Elektor Electronics*, too, has published a fair number of designs. In most, if not all, cases, these emulators are connected to the target circuit (sometimes also called host circuit) by way of a length of flatcable terminated in a DIP-style plug that can be inserted in an EPROM socket. The length of the ‘umbilical cord’ is limited to prevent a too high capacitive load on the data and address bus of the target system, or timing problems caused by pulse reflection on the cable. In practice, the connection between the EPROM emulator and the target circuit is often a source of headaches.

A second, often neglected, disadvantage of the traditional EPROM emulator is that it can not be programmed using a normal EPROM programmer. Usually, programming such units requires a special program contained in the emulator software (and hopefully supplied with the kit).

The emulator presented in this article goes off the beaten track and is not hindered by any of the above disadvantages. It can be programmed in any old EPROM programmer, and the complete unit may be plugged into the EPROM socket in the target circuit without any problems.

The EPROM

An EPROM is normally programmed in a general-purpose EPROM programmer. Most of these units first run a blank check on the chip you've inserted. This is done by reading all bytes in the EPROM to see if they are at FF. If one or more bytes are found with a different value, the EPROM is not empty and a suitable error report is produced. In many cases, no further action is possible until the entire chip is blank. Some programmers are capable of detecting the EPROM brand and type inserted into the programming socket. Once this information is (automatically) established, the relevant programming voltage(s) and programming algorithm are set up. Device recognition (if supported by the chip manufacturer) is effected by applying +12 V to the A9 pin and then reading certain addresses from the EPROM.

Once the programmer has established that the EPROM is empty, the actual programming sequence is allowed to begin. First, the EPROM is switched to programming mode by applying a relatively high voltage to the V_{pp} pin on the device. The level of the programming voltage depends on the EPROM type used, and lies between 12.5 V and 21 V. At the same time, the EPROM supply voltage (at the V_{CC} pin) is raised from +5 V to +6 V. If you are curious to know how that is done in practice, look ahead at the top part of Figure 4.

With the EPROM in programming mode, a logic Low at the $\overline{\text{CS}}$ (chip select) input of the chip allows the signals applied to the datalines to be programmed at the location (address) pointed to by the address bus. A Low input level applied to the $\overline{\text{OE}}$ (output enable) input enables the contents of the current address to be read out. This is usually done immediately after programming a byte, to make sure the program-

Features

- Emulates 27C256 EPROM
- Suitable for programming with any programmer supporting the 27C256
- Electrical protection against reading of DEVICE-ID
- Easy to erase
- Compact
- Easy to use

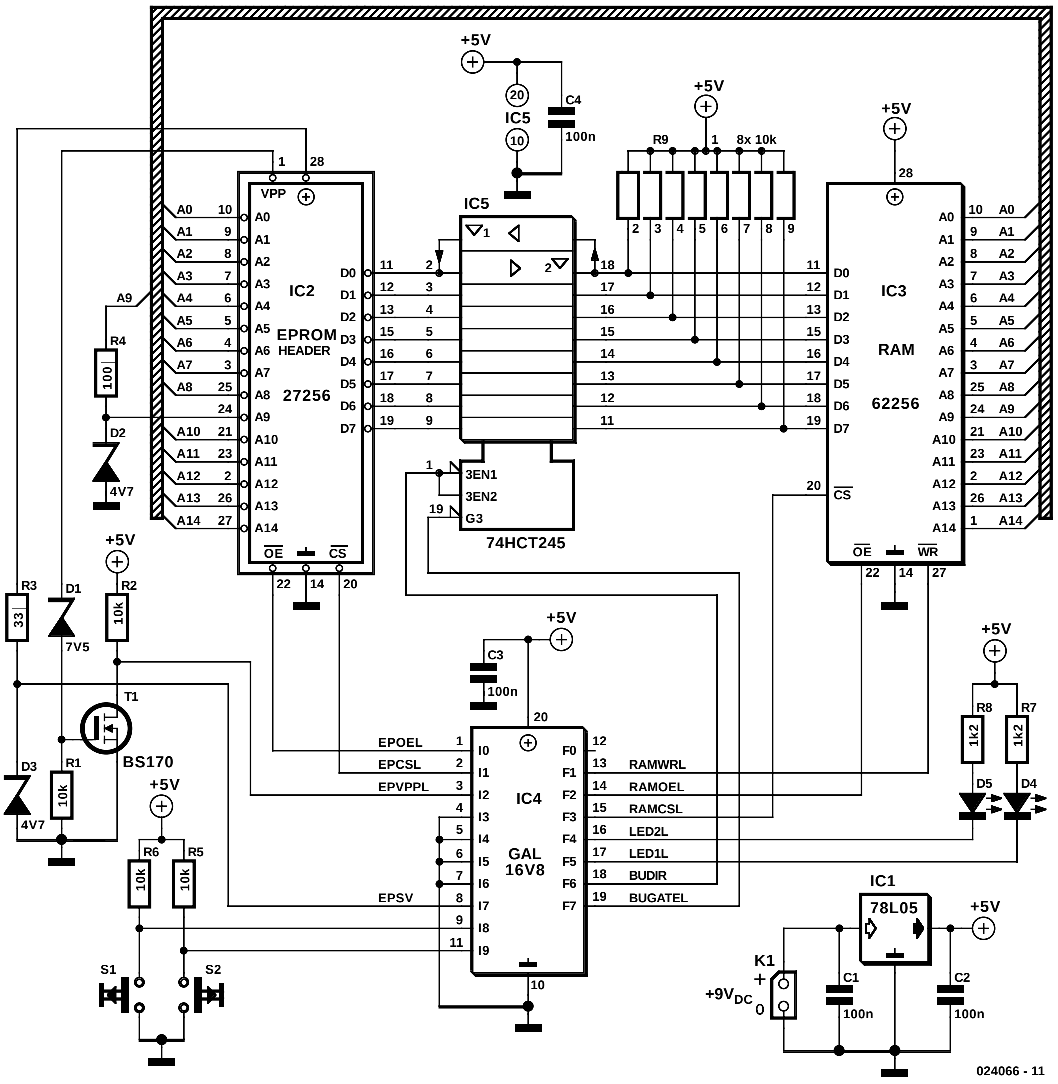


Figure 1. The circuit diagram of the EPROM emulator — exquisite simplicity.

ming operation was successful. If the verify returns ‘ok’ the next byte is programmed.

The length of the programming pulse depends on the programming algorithm applied. The same goes for repetition of the programming cycle if the relevant byte was not successfully programmed.

During normal use of the EPROM (V_{pp} and V_{cc} at 5 V), the device will respond ‘normally’ to signals applied to the $\overline{CS}$ and $\overline{OE}$ pins. Only with both inputs activated (i.e., at logic Low) will the EPROM supply the data belonging with the address placed on the address bus.

Practical Circuit

The circuit diagram shown in **Figure 1** comprises relatively few components — three ICs, a voltage regulator, one FET and two indicator LEDs do the job. IC2 is not an integrated circuit but an assembly of two 14-way pinheaders that form the connections of the EPROM being emulated by the circuit.

IC3, a RAM type 62C256, forms the heart of the circuit. Although a 70-ns version is specified in the parts list, a faster chip is, of course, also allowed. Slower versions, however, are not recommended for use in this circuit.

IC4 is a GAL type 16V8 which has been programmed to supply a fair amount of (invisible) logic circuitry that ‘translates’ external signals to the ones required by IC3. This translation process requires the GAL to be connected to the EPROM socket via a few logic links, as well as to be provided with information as to the presence of an external voltage at the V_{cc} pin, and a voltage greater than 12 V at the V_{pp} pin.

The combination R3-D3 affords protection of the GAL against higher voltages at the EPROM V_{cc} pin. That is necessary because this voltage is usually raised to +6 V during programming. The circuit around transistor T1 detects the programming voltage at the V_{pp} connection of the EPROM socket. Zener diode D1 starts to conduct if the programming voltage is present. The voltage at the gate of T1 is then high enough for the FET to be switched on. In combination with resistor R1, zener diode D1 lowers the voltage to about 7.5 V, thus ensuring that the FET remains securely switched off at a voltage of 5 V at the V_{pp} terminal.
IC5, then, acts as a buffer device between the

```
*IDENTIFICATION
    024066;
*TYPE
    GAL16V8;

*PINS

% Pin-assignment for input signals %

    EPVP    = 8,        % Detection of 5V supply                  %
    /EPVPP  = 3,        % Low active detection of VPP > 12V       %
    /EPOE   = 1,        % Input for /OE from EPROM-socket         %
    /EPCS   = 2,        % Input for /CS form EPROM-socket         %
    /SW1    = 9,        % Input for switch S1                     %
    /SW2    = 11,       % Input for switch S2                     %

% Pin-assignment for output signals %

    /RAMCS.t   = 15,    % output for /CS of RAM IC3               %
    /RAMOE.t   = 14,    % output for /OE of RAM IC3               %
    /RAMWR.t   = 13,    % output for /WR of RAM IC3               %
    /BLANK.t   = 16,    % BLANK-state output for LED D5           %
    /PROGRAM.t = 17,    % PROGRAM-detect output for LED D4        %
    /BUGATE.t  = 19,    % output for /G of buffer IC 5            %
    BUDIR.t    = 18;    % output for DIR of buffer IC 5           %

*BOOLEAN-EQUATIONS
   RAMCS.e   = VCC;     % These lines make sure that all used     %
   RAMOE.e   = VCC;     % output-lines of the GAL are             %
   RAMWR.e   = VCC;     % constantly driven                       %
   BLANK.e   = VCC;
   PROGRAM.e = VCC;
   BUGATE.e  = VCC;
   BUDIR.e   = VCC;

% Switch to BLANK-state is S1 is pressed and hold this state      %
% Until the EPROM programmer wishes to program the EPROM          %
% OR the user wishes to switch to normal operation by pressing    %
% S2                                                              %
% This signal also drives LED D5 to indicate the BLANK-mode       %
   BLANK    = BLANK*/PROGRAM*/SW2+SW1 ;

% Detection of a programming pulse to drive LED D4                %
   PROGRAM = EPVPP*EPCS*/EPOE;

   % CS for the RAM : Continuous during programming, else equal %
   % to CS on the ROM                                           %
   % Only active during presence of external power to save the  %
   % battery                                                    %
   RAMCS= EPVPP*EPVP+/EPVPP*EPCS*EPVP;

   % OE for the RAM : During programming if OE on the ROM is    %
   % active AND Cs of the ROM is inactive                       %
   RAMOE= EPVPP*EPOE*/EPCS + /EPVPP*EPOE*/BLANK;

   % WR for the RAM : During programming if ROM-OE is inactive  %
   % AND ROM-CS is inactive, else if in BLANK-mode AND ROM-OE   %
   % is active. This means that reading while in blank-mode     %
   % causes the RAM to be written. Pull-ups make sure that the  %
   % code 0xFF will be written                                  %
   RAMWR= EPVPP*EPCS*/EPOE + /EPVPP*BLANK*EPOE;

   % The DIR-signal for the 74HCT245. When this pin is active,  %
   % Data is transferred from the socket to the RAM-IC          %
   % Otherwise, the datapath is FROM the RAM to the socket      %
   BUDIR = EPVPP*/EPOE;

   % The GATE-signal for the 74HCT245. If it is active (low)    %
   % the drivers are active, otherwise the A and B pins are     %
   % in High-Z state                                            %
   % It has to be active even in the BLANK-mode, so it wil force%
   % 0xFF on the external databus during blank-check!           %

   % BUGATE is only active when external power is supplied to   %
   % save power from the battery whenever the device is not     %
   % required to be active                                      %

   BUGATE=EPVP*EPVPP*EPOE + EPVP*EPVPP*EPCS  + EPVP*/EPVPP*EPOE;
*END
```

Figure 2. JEDEC file listing of the GAL contents.

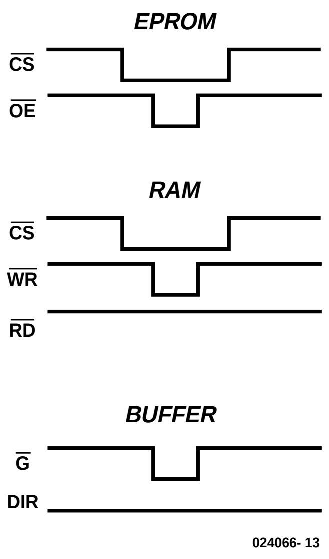


Figure 3. Signals in Blank mode.

EPROM and the databus. This IC is necessary while the emulator is in 'Blank' mode when it is exptected to supply the value FF at all times during a read command. The GAL arranges for the driver to be activated in the direction B→A and the EPROM is not read (actually, a write command is given to the RAM IC, more about this further on). Because of the pull-up resistors inside R9, IC5 sees only logic Ones at the B inputs, and faithfully copies these to the external databus.

Some EPROM programmers are equipped with pull-down resistors. That is why IC5 is included in the circuit, since without it the resistance value of R9, depending on the programmer, would have to be selected low enough to prevent the emulator acting as a significant (but unwanted) voltage source in the target circuit.

Switch S1 allows you to switch the EPROM emulator into Blank mode, while S2 needs to be actuated to manually leave this mode.

Finally, there are the components around voltage regulator IC1. This is a standard 5-volt regulator that steps down the 9-V input voltage to 5 volts required by the logic circuitry.

The GAL

Apart from IC3, the most important part of the circuit is the GAL chip. As already mentioned, it contains all logic circuitry we need to translate and format external signals so that they can be understood by the RAM chip.

The JEDEC source file listing for the GAL is shown in **Figure 2**. Although an uncomplicated and modest bit of programming, the equivalent logic function of the GAL is the same as that offered by a rather more than a handful of ICs… which if used would make the project unwieldy.

The GAL will operate in one of three modes: Blank, Normal and Program. Each of these modes requires the GAL to work in a different manner. The Blank and Normal modes may be selected manually by the user. By pressing S1 once, the GAL is configured for **Blank mode**. The contents of the emulator may then be erased by means of a blank check run by the programmer. This state is held until the user presses S2, or the programmer starts to program the emulator.

Normal mode is selected by pressing S2 once. The emulator unit will hen behave like an ordinary 27C256 EPROM.

Program mode is automatically selected as soon as the programmer starts to program the emulator (V_{pp} raised to >12.5 volts). Once the programming is finished (V_{pp} dropped to +5 V), the emulator automatically returns to Normal mode.

Blank mode

This mode actually has two functions: mimic an empty (blank) EPROM and clearing the RAM IC. The signals used during Blank mode are shown in **Figure 3**. Most EPROM programmers will check if the device is empty before programming it. The conclusion that the entire EPROM is empty is also employed to speed up the programming process. Because the programmer 'knows' that every address reads FF, it may safely skip all addresses that have to contain FF. This cuts down on programming time, although it does require the EPROM emulator to be 'empty' before it is programmed.

The normal procedure to check if an EPROM is empty is to read all addresses and compare the resultant data against the value FF. To do so, your programmer will generate all possible addresses and generate a read pulse at each of them. These pulses may be used to clear the EPROM emulator memory. Each address is programmed by converting the read pulse from the programmer into a write pulse for the RAM IC. All datalines being pulled to +5 V via resistor array R9, IC3 is then filled with the value FF at all memory locations.

At this point, the programmer has issued a read command and expects device data (preferably FF) to be placed on to the databus. This is arranged by buffer IC5. The DIR inputs remain Low, so the A signals are configured as outputs and the B signals, as inputs. The B signals, too, are connected to +5 V via resistor array R9, so the buffer IC will faithfully place the desired data (FF) on to the databus.

Program mode

The GAL monitors the signal $\overline{EPV_{pp}}$ to detect the presence of a programming voltage (≥ 12.5 mming mode. An EPROM being programmed responds differently to the signals at $\overline{CS}$ and $\overline{OE}$. Actually, in this state, these signals may be renamed to $\overline{WR}$ and $\overline{RD}$. Also, there is no longer an additional signal available to enable the EPROM to be selected. Fortunately, that is not necessary during programming, because there is no need for the EPROM to share the data bus and address bus with other memory devices. In this state, therefore, it appears as if the EPROM is permanently selected — see also **Figure 4**.

During a programming pulse, the buffer IC has to transfer the data from the programmer to the RAM (IC3). This is achieved by pulling the DIR signal logic High. During a read pulse ($\overline{OE}$ activated), the HC245 device has to buffer the data in the opposite direction. This

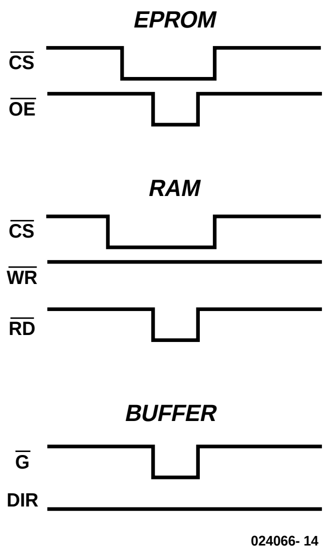


Figure 4. Signals in Program mode.

is arranged by the GAL pulling the DIR input logic Low.

Normal mode

The major signals with the circuit in Normal mode are shown in **Figure 5**. From the diagram it is clear that in this mode the GAL does not have to perform particularly difficult functions.

The GAL is also responsible for automatic power reduction when the emulator is not in use. By means of R3 and D3, IC4 is able to detect the presence of an external supply voltage. If no external supply voltage is detected, IC5 and IC3 may be switched off. IC4 prevents the signals $\overline{G}$ and $\overline{CS}$ from being activated, effectively reducing the current consumption of the RAM and the buffer when the circuit is not used. A very useful feature indeed because our EPROM emulator is battery-powered!

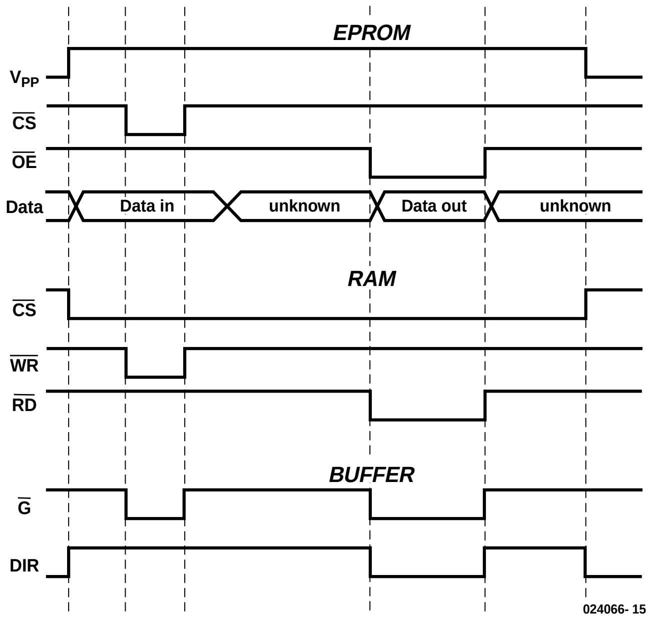


Figure 5. Signals in Normal mode.

COMPONENTS LIST

Resistors:
R1,R2,R5,R6 = 10kΩ
R3 = 33Ω
R4 = 100Ω
R7,R8 = 1kΩ2
R9 = 8-way SIL array 10kΩ

Capacitors:
C1-C4 = 100nF

Semiconductors:
D1 = zener diode 7.5V
D2,D3 = zener diode 4.7V
D4 = LED, high-efficiency, yellow
D5 = LED, high-efficiency, red
T1 = BS170 or BS107
IC1 = 78L05
IC3 = 62256-70CP
IC4 = GAL 16V89, programmed, order code **024066-31** (see Readers Services page)
IC5 = 74HCT245

Miscellaneous:
IC2 = 2 ? 14-way pinheader
S1,S2 = pushbutton
K1 = 9V battery clip with leads
PCB, order code **024066-1** (see Readers Services page)
Disk, GAL JEDEC listing, order code **024066-11**

PCB design and GAL listing also available as files from Free Downloads at www.elektor-electronics.co.uk

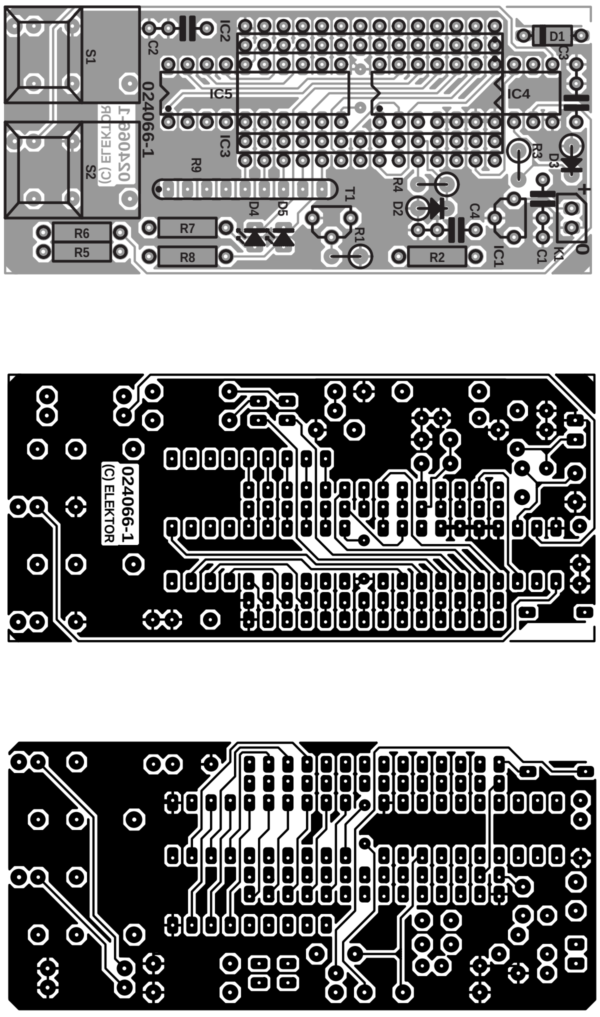


Figure 6 The unit is compact thanks to IC3 being mounted above IC4 and IC5.

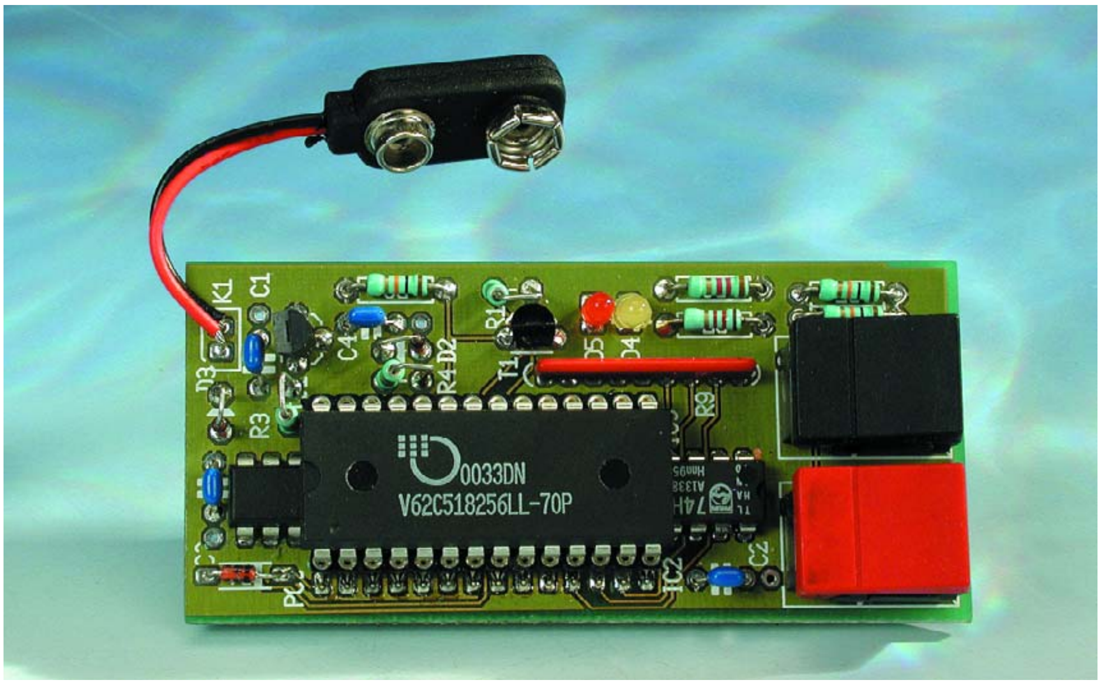
Power supply

The complete circuit is powered by a 9-volt battery. Purposely no provision is made for the circuit to be powered from the +5-V supply pin of the EPROM. After all, the voltage at this pin may become +6 V during programming, which is not a healthy level for the ICs in the emulator. If you do want to employ this voltage, measures should be taken for it to be stabilised at +5 V. A DC-DC converter could be used for this purpose, but the disadvantages will always outweigh the benefits, mainly because the circuit becomes bulky and difficult to work with in practice. The battery can be eliminated anyway because the circuit is not permanently connected to an external power supply which would guarantee that the RAM contents are not lost or corrupted the instant the emulator is removed from the programmer and taken to the EPROM socket of the target system.

Construction

The artwork of the printed circuit board designed for the EPROM emulator is shown in **Figure 6**. The board allows the unit to be compact and easy to handle. Construction is easy by almost any standard. As customary, it is best to start with the lower profile components (in this case, the horizontally mounted resistors). Then follow the other discrete parts, the integrated circuits and the pushbuttons. Two points are worth noting. The first concerns the emulator connections marked 'IC2'. These consist of two rows of 14-way pinheaders whose pins have to protrude downwards, requiring the headers themselves to be soldered at the **underside** of the PCB, too. This is best done with the ICs already mounted. A further pint to note is that IC3 is mounted **above** IC4 and IC5. The latter two have to be soldered on to the board first, while IC3 is fitted with a raised socket. This is conveniently made from two 14-way pinheaders. As a matter of course, IC4 and IC5 are mounted without sockets, else IC3 would require a second socket! By the way, note the position of IC3: viewed from above, it should be at the right-hand side of connectors for 'IC2'.

Here are some more practical details. The holder for the 9-V battery may be secured to IC3 using two-sided sticky tape to make the unit sturdy and compact. It may also be a good idea to file a away a small piece of PCB material to the left of diode D1 to make room for the lever on the programmer's ZIF socket. In practice, however, we found that the lever had to be lifted ever so slightly to enable the circuit to be inserted into the socket. The above tip, may, therefore, not be applicable in all cases.



Finally...

With the board fully stuffed and the battery connected, the emulator is, in principle, ready for use. First, however, we recommend running a few function tests.

The operation of the GAL may be checked by pressing S1. LED D5 should light and go out again when S2 is pressed once more.

Now insert the EPOM emulator into the IF socket of your programmer and then press S1. Run a Blank Check on the programmer. If everything works as it should, the programmer should report an empty (blank) EPROM. Now press S2 to force the emulator into Normal mode. Next, run another Blank Check on your programmer. This test should also be completed without problems, proving that each and every memory location in the emulator RAM (IC3) contains the value FF.

As a final test, we can check the programming. Choose an arbitrary programming file on the programmer and 'burn' it into (emulated) EPROM. LED D4 should light. Next, run a Verify operation to see if the programming operation was successful.

If a problem is encountered, carefully inspect the construction of your emulator. Check the orientation of the ICs, diodes, etc., as well as the value of all components. Also inspect the solder joints for short-circuits and/or dry joints.

One more tip: temporarily install a ZIF socket in the EPROM socket of the target circuit. This allows the emulator unit to be inserted and removed as many times as necessary without the risk of bad contacts between target circuit and emulator. The cheaper brands of IC sockets which proliferate in the hobby circuit these days are prone to developing contact problems when used more than two or three times.

(024066-1)

Everything you always wanted to know about Speed Cameras but were afraid to ask…

By: B. Bouchez

Feared by many motorists (let those who have never driven too fast cast the first stone), the speed camera, usually containing a radar, is an electronic device whose workings are still a mystery to most electronics hobbyists. In this article we take the covers of these curious devices and describe their operation in detail, also considering a few ideas to guard against their actions.

(Source: Applied Concepts, Inc. / Stalker Radar)

A little bit of history

Many, many years ago, the investigations carried out on electromagnetic radiation in the thirties revealed that RF waves appeared to be reflected back from some objects. This property became more discernible as the frequency was increased (above 100 MHz). It wasn't until the end of the thirties that SHF (f > 1 GHz) waves could be generated easily, thanks to the introduction of the *magnetron* (which is still used in modern microwave ovens).

The first practical application of this system was for *Radio Detection And Ranging,* better known as its acronym RADAR. The operation of radar is based on the transmission of a series of SHF waves at regular intervals. When these waves hit an object of sufficient mass then part of them will be reflected back to the transmitter, where an aerial can detect them.

In the original systems the start of the horizontal sweep in the cathode-ray tube was synchronised with the transmission of the signal burst and the return signal received by the aerial was fed to the vertical deflection amplifier. When the aerial received a return signal (the waves reflected back by an object) a spike could be seen further to the right of the tube, the position depending on the distance of the object (see **Figure 1**).

This system could not only be used to detect the presence of every object (aeroplane) that was in its field of 'view', but it could also determine the distance accurately between the aerial and the object, since the propagation speed of the waves and the rate of horizontal deflection (time base) were known. This is how R.A.D.A.R. came about, an acronym that soon turned into an everyday word.

Subsequently the radar was improved to cover all of the surrounding area rather than just a small section directly in front of the aerial. This was done by mounting the aerial assembly on top of a turntable (the cathode-ray tubes were now given deflections in two directions).

Introducing the Doppler effect

The device just described is not capable of determining the speed of the detected object; this was limited to measuring the movement of the echo on the screen, which gave a rather inaccurate result.

As an example, consider a car that makes a sound with a fixed frequency (a car that is driven with fixed revs for example). When you are in the car you won't notice any variation in the frequency of the engine sound.

If however you stand at the side of the road and listen to the car when it drives past under identical conditions you will notice that the frequency of the engine sound increases as the car comes nearer and then decreases as the car travels past you. (This effect is also noticeable in F1 Grand Prix races when the cars roar past the camera.)

This phenomenon works the other way round as well: when you drive your car past somebody who is shouting on the pavement, you should notice that the frequency of the shouting increases as you go towards the person, and then decreases as you move away.

When the distance between the sound source and the receiver remains constant then the frequency of the received signal won't vary either.

The Doppler effect (named after the physicist who discovered it) is nothing more than that described by the formula in **Figure 2**:

$f_M = 2\, v\, f_E \cos(\alpha / c)$

where

f_M is the frequency of the received signal.
v is the speed of the vehicle.
f_E is the frequency of the transmitted signal.
α is the angle between the transmitter and the path along which the vehicle travels.
c is the propagation speed of the signal in air (300,000 km/s for radio waves, 340 m/s for sound waves).

From this we can deduce that sending a fixed frequency signal towards the car and then measuring the frequency of the returning signal will give you the speed of a car.

This is the principle used for radars in speed cameras, although they have little in common with the systems described in the first part of this article.

It should be mentioned that the sensitivity of the radar increases as the angle between the beam and the path of the vehicle decreases. For this reason the aerials of speed cameras are positioned parallel to the roads rather than across them! This is also the reason why few types of radar can work along bends, since the angle between the beam and the vehicle continually changes, creating errors in the measurements.

From theory to practice!

Now that we've seen how the Doppler effect can be used to measure the speed of vehicles, we'll take a look at the commercial applications that are found at the side of the road.

The basis of every speed camera (let's call it a radar) is a SHF generator, which can transmit a beam in a specific direction. From the previous section we have learnt that the sensitivity of the device is directly proportional to the frequency of the beam. The exact frequency used depends on the manufacturer, but is generally between 2 GHz and 15 GHz. The device can either have a SHF oscillator based on a Gunn diode and a resonant cavity, or a transistor oscillator followed by a power amplifier. The power of these oscillators is not very high (usually less than 10 mW), but the effective power output is increased through the use of a directional aerial.

The receiver for the reflected signal is often based on a Schottky diode, situated at the focal point of the aerial (usually the same aerial is used for transmission and reception), which functions as a mixer of the transmitted and reflected signals.

The output signal of the receiver is amplified, conditioned by an analogue circuit and then passed on to the measurement section, which is nothing more than a frequency counter.

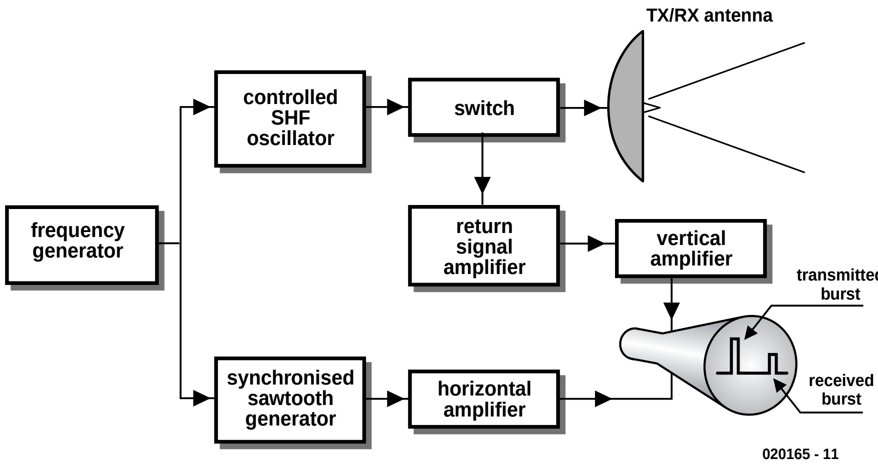


Figure 1: Principle of operation of first generation radars (1940).

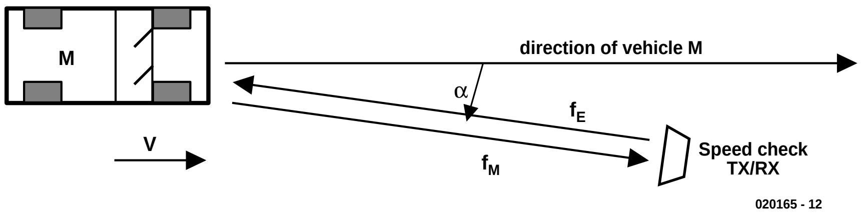


Figure 2: The Doppler effect.

The signal from the frequency counter goes to a microprocessor that calculates the speed and sends it to a display. It also checks if the measured speed exceeds a preset value and warns the police officers who are nearby that an offender has just passed, or it activates a camera and flashgun.

In short, the basic principles behind a high frequency speed detector (**Figure 3**) are not very complex. (Note that we're talking about the principles, a practical implementation is something else altogether, especially the high frequency section.)

How well does it work?

Now that we know how it all works we may wonder how reliable the measurements made by these devices are. Let's keep one thing clear: we have no intention of encouraging any of our readers to break the speed limit or behave irresponsibly. We just want to look at the problem from a technical viewpoint to discover what the limits are of SHF speed cameras. In this way we can distinguish between proven facts and 'rumours' that are doing the rounds, made by people with little knowledge of electronics. Instead of straying into difficult technical considerations we'll just answer the most common questions.

Operation during rain or mist: in contrast to widespread opinion, radar works perfectly well during rain or mist (after all, radar is used extensively to help with the landing of aeroplanes in bad weather!). In general, when it rains it comes down vertically (at least it does here!), which is at right angles to the radar beam, bringing about a Doppler effect of zero (cos $90^\circ = 0$, so $f_M = 0$). Heavy rain that comes down at an angle due to strong gusts of wind can affect the signal-to-noise ratio of the receiver and prevent its correct operation. In this case the processor will simply reject the measurements.

Since mist doesn't move with respect to the radar beam (or only very slowly) it will be practically invisible to the receiver and the measurements are completely unaffected.

Measurement range: the distance from which a radar can measure the speed of a vehicle depends on two factors: the power of the SHF oscillator and the sensitivity of the detector. We already knew that the oscillator power is generally low and that the use of a directional aerial increased the transmitted power. The biggest problem for the detector is the signal-to-noise ratio, which doesn't get any better with Schottky diodes. In this section the sensitivity can also be improved through the use of aerials. Whilst the first radars could only take measurements up to twenty meters, the newer models with their ultra-sensitive detectors are capable of taking measurements over several hundred meters, so well before they can be seen from the car!

Reaction time: just as with any other equipment that use frequency counters, these speed cameras also require a certain time to take a measurement. Furthermore, most devices now take several measurements rapidly, making it possible to reject any possibly erroneous measurements. Older models required about half a second to take a reliable measurement. Current models react within a tenth of a second, so any motorist who ignores the speed limit will have very little chance of avoiding a fine after noticing a speed camera. Sometimes the radar equipment also contains a DSP (Digital Signal Processor), which uses a special algorithm with a very short pull-in time, making extremely fast readings possible.

Continuous transmission: in contrast to what you may have thought after reading the theoretical part of this article, a radar does not need to have its oscillator functioning continuously. It only needs to be active long enough to stabilise and take a measurement. Actual radar equipment works on a random basis or is activated only when a vehicle comes nearby.

Discrimination: when several vehicles travelling at different speeds encounter a radar beam, the resulting Doppler signal contains a mixture of signals at different frequencies. The majority of current devices can't separate these components and reject the measurement as faulty. There are however newer systems that contain a DSP (the author has worked on one of these systems), which can measure the speed of several cars simultaneously. So now only those cars that happen to be in the 'shadow' of others can escape the speed cameras.

The long and short of it is that speed cameras have become so accurate and reliable (which can be confirmed by drivers who don't keep to the speed limit), that it has become extremely difficult to evade them!

On the wrong side of the law

Mankind, and especially homo automobilis, behaves in such a way that when he comes across an obstacle he will try everything to get round

it. Speed cameras are no exception to this, and numerous boffins (whether competent or not) have contributed to the development of counter measures.

Before we continue we would like to make one thing perfectly clear: in the majority of European countries the possession, and even more so, the *use* of equipment intended to disrupt the operation of speed cameras is illegal. Indifference in this area is not recommended, since the consequences can be dire (handing in of the driver's licence, confiscation of the vehicle, prosecutions, etc.).

In spite of this, some drivers think it is worth the risk and don't shrink back from using these devices.

Loosely speaking, there are two types of 'anti-radars`: jamming devices and detectors.

The jamming devices are simply small SHF oscillators, which are used to send a 'fake' signal back to the speed camera, causing the measurement to fail and preventing the logical analysis of the frequency. Besides the fact that these devices are relatively ineffective (in most radars the circuits are less sensitive to interference signals: the frequency of the jamming signal therefore has to be as close as possible to that of the speed camera, and every device has its own frequency), the electronic circuits in the radar can detect such jamming signals and notify the police. A jamming device is therefore a sure-fire way to get caught!

A detector on the other hand consists of a simple SHF receiver, and by definition these can't be detected. In the USA (where their use is permitted) they are sold in large quantities. On the Internet they are readily available and in some European countries they are also freely available (their sale is permitted but their use isn't!). These are usually relatively simple circuits containing a microwave detector and a comparator that drives an alarm. In short, simplicity itself (although their price seems to suggest the opposite is true!).

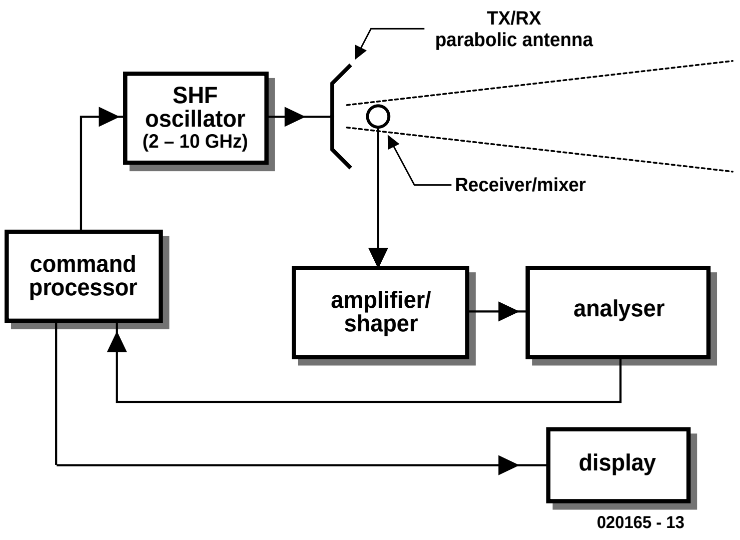


Figure 3: Basic principle of a high-frequency speed camera.

It isn't difficult to design a broadband detector that reacts to frequencies between 2 and 10 GHz, which is the range where most modern devices operate. However, if the oscillator of a speed camera is set to a frequency that is outside the range covered by the detector, or it uses an optical laser, then you're bound to get caught.

The second problem is that in order to detect something, there first has to be something to detect (obvious, isn't it?). Older radar equipment transmitted continuously, which made the task simpler, but newer models only transmit intermittently, either randomly or in short bursts, reducing the chance of detecting these devices. Some models (such as the Mesta 208 sold in France where the author lives) are even more cunning and only come into action when a car comes within their range. These 'green bullets', as they are known because of their shape and colour, have an optical detector on top that literally sees the vehicles coming.

As soon as there is some movement in front of the device it springs into action.

This brings us to the third problem: a radar detector will sense the beam at that instant. But at the same time the speed camera is already doing its work. From this it follows that in the time taken by the driver (a typical reaction time for people is about half a second) to take appropriate action (to brake or disrupt the measurement), the radar will already have taken four or five measurements.

The detection is made more difficult by the fact that very narrow beams are used, making for a small 'detection area'. Some users of radar detectors have noticed that the beam can also be detected when it is reflected off other cars ahead and have gladly made use of this property.

And now for the final problem: most radar equipment can take measurements of approaching (from the front) or receding (from the rear) vehicles. But the sensitivity of most detectors is limited to just one direction. To be prepared for any eventuality the vehicle should therefore have a detector at both the front and rear!

And finally...

As the saying goes, if you play with fire you may get burned. As we have seen in this article, speed cameras have become reliable instruments that are difficult to locate or interfere with.

Knowing how to detect a SHF beam is one thing; to disrupt a speed measurement is another story. In short, although most in-car radar detectors sold at the moment do work and show the presence of speed cameras, they are of little use because the speed camera is likely to be triggered *before* the driver can slow down.

Even though speed cameras are sometimes situated in places where they aren't justified (which has given rise to the name 'euro-pump` on the continent), in our opinion it shows more sense to respect the speed limits (also out of respect for other road users), rather than attempt to get round the law at all costs.

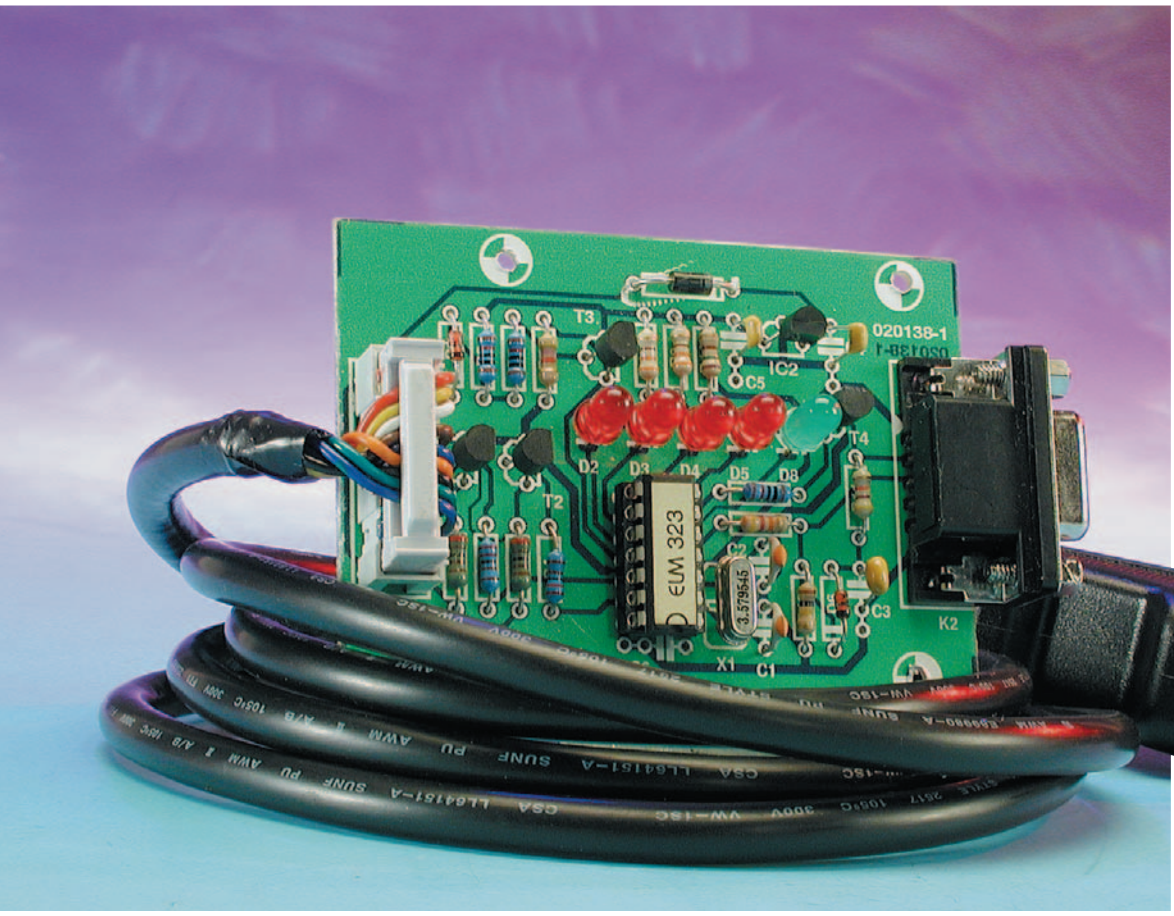
(020165-1)

Vehicle Diagnostics Adapter

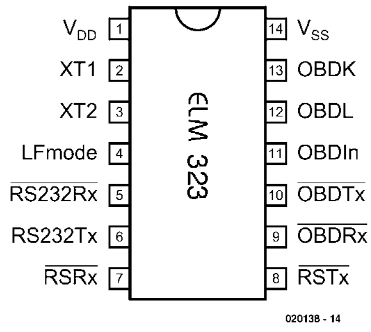
Interface between an OBD-2 vehicle diagnostic connector and a serial PC port

Design by G. Müller

In last month's article we took a look at the background and specification of the OBD-2 diagnostic connector fitted to new cars. Now, as promised we present an interface adapter that allows your car to confess its innermost secrets to your computer.



We can thank an EU directive dating from 1998 for the introduction and standardisation of the OBD-2 vehicle diagnostic connector. The connector is standard on all spark-ignition engined vehicles from 2000 and should be fitted to diesel engined variants by 2003. The communication protocols used can come in three different varieties. In Europe the most common protocol is the ISO standard. All of these protocols rely on serial data transfer but the



V_{DD} 1
XT1 2
XT2 3
LFmode 4
$\overline{\text{RS232Rx}}$ 5
RS232Tx 6
$\overline{\text{RSRx}}$ 7
14 V_{SS}
13 OBDK
12 OBDL
11 OBDIn
10 OBDTx
9 $\overline{\text{OBDRx}}$
8 $\overline{\text{RSTx}}$
ELM 323
020138 - 14

Figure 1. The ELM323 pin-outs.

ELM323 Technical Data

Absolute maximum ratings:

Storage temperature	–65 °C to +150 °C
Ambient temperature with power applied	–40 °C to +85 °C
Voltage on V_{DD} with respect to V_{SS}	0 to +7.0 V
Voltage on any other pin with respect to V_{SS}	–0.6 V to (V_{DD} + 0.6 V)

Electrical characteristics

All values assume operation at 25 °C and 5 V supply unless otherwise stated. For further details refer to note 1 below.

Characteristic	Minimum	Typical	Maximum	Units	Conditions
Operating voltage VDD	4.5	5.0	5.5	V	
VDD rate of rise	0.05			V/ms	See note 2
Average supply current IDD		1.0	2.4	mA	See note 3
Input low voltage	VSS		0.15 VDD	V	
Input high voltage	0.85 VDD		VDD	V	
Output low voltage			0.6	V	Current (Sink) = 8.7 mA
Output high voltage	VDD – 0.7			V	Current (source) = 5.4 mA
$\overline{\text{RS232Rx}}$ Pin input current	0.5		0.5	mA	See note 4
RS232 Baud rate		9600		Baud	See note 5

Notes:

1) This chip has a PIC16C505 from Microchip Technology as a core embedded microcontroller.
2) This spec must be met to ensure that a correct power-on-reset occurs. If the supply rises too slowly problems with the internal reset occur.
3) Device only. Without any load current.
4) This value represents the current flowing through the input protection diodes when applying large voltages to the $\overline{\text{RS232Rx}}$ input (pin 5) through a current limiting resistor. Values given are the maximum that should be allowed to flow continuously.
5) Nominal data transfer rate when the recommended 3.579 MHz crystal is used as the frequency reference. Data is transferred to and from the ELM323 with 8 data bits, no parity and 1 stop bit (8 N 1).

signal levels and message format are not compatible with the serial communications port of a Personal Computer.

The interface adapter described here contains a pre-programmed microcontroller produced by the company Elm Electronics of Canada. This controller together with a few external components allows the OBD vehicle connector to communicate with the serial port of a PC, laptop or PDA running a terminal emulation program. Alternatively a more sophisticated program can be developed for the PC to provide a better user-interface and allow interpretation and resetting of failure codes, together with real-time display of actual sensor information. In a follow-up article we will look at the development of just such a program and describe in detail the steps necessary to produce the finished program. The source code for this program is written in C and can be ported to any of the common operating systems such as Linux, BeOS, or QNX using the freely available gcc compiler program. The source code together with a version of the program compiled to run under Windows will be available to download.

The interpreter chip

The ELM323 was specifically designed as a low-cost solution for interfacing a PC or PDA to a vehicle diagnostics connector. To keep things simple it communicates at a fixed baud rate of 9600 baud and does not offer a handshaking option for the RS232 interface. In addition it is only able to communicate using the 10.4 kHz ISO 9141 Protocol. This standard is the most common used by the majority of European and Asian manufacturers. Vehicles built in the US use VPW and PWM protocols and suitable interpreter chips are also available from Elm Electronics.

The most important technical specifications of the ELM323chip are listed under the heading 'Technical Data' The pinouts are shown in **Figure 1** and the internal block diagram is in **Figure 2**. The pin descriptions now follow:

V_{DD} (Pin 1)
This pin is the positive supply pin and should be the most positive point in the circuit (see the technical specifications). An internal power-on reset is derived from this pin to initialise the microcontroller.

XT1 (Pin 2) and XT2 (Pin 3)
Connect a 3.579545 MHz crystal (NTSC TV colour burst) between these two pins. A capacitor (typically 27 pF) is fitted to each of these pins down to Vss.

LFmode (Pin 4)
This input selects the default linefeed mode after a reset or at power-up. A high-level on this pin will mean that each line sent by the ELM323 will be terminated by a carriage return (CR) and line feed (LF) character. A Low level on this input will mean that each line sent will be terminated by a carriage return only. The mode can also be changed in software by issuing the ATL0 or ATL1 command from the AT command set.

$\overline{\text{RS232Rx}}$ (Pin 5)
The RS232 transmit signal can be connected directly to this pin providing that a current limiting resistor (typically about 47 kΩ) is connected in series. On-chip diodes ensure that

inputs to the ELM323 are protected against overvoltage while Schmitt triggers reduce the effects of noise on the inputs.

RS232Tx (Pin 6)
The ELM323 transmits data from this output. The signal level is compatible with most interface driver IC's and there is even enough current to use a single PNP transistor as a line driver.

LED Drive outputs (Pin 7, 8, 9 and 10)
These four pins are low when the ELM323 is transmitting or receiving RS232 or OBD data. Providing that a suitable series current limiting resistor is fitted, the outputs can source or sink sufficient current to directly drive an LED.

OBDIn (Pin 11)
The serial OBD-Data is input on this pin. A logic high represents the active state of the OBD K line. There is no Schmitt trigger fitted to this input so an external input buffer should be employed to reduce the input signal transition times.

OBDL (Pin 12) and OBDK (Pin 13)
These active-high output signals are used to drive the OBD bus using external NPN driver transistors. Data transfer normally occurs over the K line but the standards specify that the driver for the L line be implemented also to ensure the bus is properly initialised. More on this later.

V_{SS} (Pin 14)
The common ground pin. (The most negative point in the circuit).

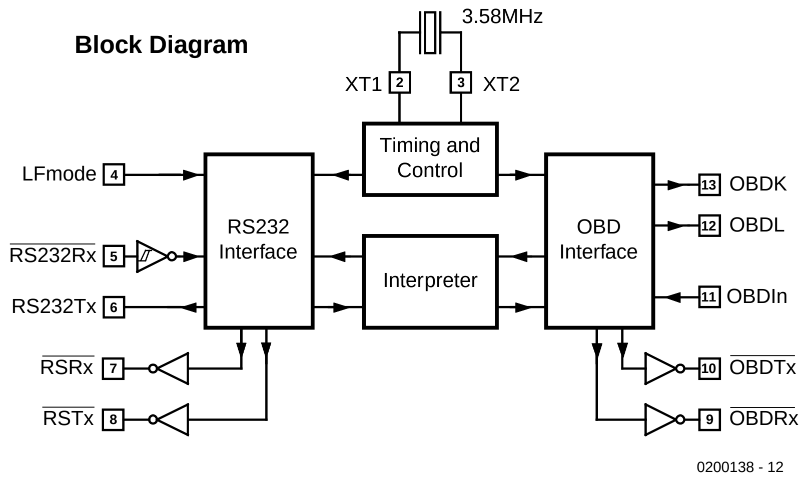


Figure 2. Block diagram for the OBD-2/RS232 converter.

The Interface circuit

The SAE Standard J1962 stipulates that all OBD compliant vehicles must provide a standard connector close to the driver's seat. The shape and pinouts of the 16-pin connector has already been described in the previous article (Card Diagnosis Systems, *Elektor Electronics* October 2002). The circuit described here plugs directly into this connector without any changes necessary to the vehicle.

The male J1962 connector (**Figure 3**) needed to plug into the vehicles connector may be difficult to obtain (see parts list) and you may be tempted to improvise by connecting to the back of the vehicle's OBD connector. If you attempt this, it should be stressed that you should do nothing to compromise the integrity of the vehicle's OBD network. The use of any type of connector that could easily short out pins (e.g., the RJ11 type phone connector) is not recommended.

The circuit of the OBD/RS232 interface with the ELM323 is shown in **Figure 4**. Power is derived from the vehicle battery (nominally 14.4 V) via pin 16 of the OBD connector (K1) while the vehicle earth is at pin 5. Voltage regulator IC2 provides 5 V for the circuit and its built-in current limit offers some protection for the circuit. Diode D7 gives reverse polarity protection. LED D8 'power' indicates that the 5 V supply is available.

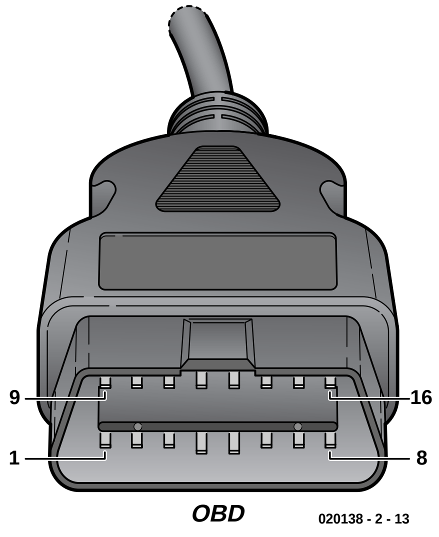


Figure 3. The 16-pin connector for the vehicle diagnostic connector.

The remaining two connections to the vehicle (OBD Pin 7 and 15) are the data lines described in the ISO 9141 and ISO 14230 standards. In accordance to the standards pin 7 of the connector is the K output while pin 15 is the L output. We will refer to these as the K line and L line of the OBD system. To comply with the specification, the ELM323 controls these two lines using NPN transistors with 510 Ω pull-up resistors.

The adapter circuit receives diagnostic data from the K line (pin 7 on the OBD connector). The data is inverted by transistor T3 before it is read by IC1 (pin 11). This transistor stage raises the threshold voltage to around 4 V instead of the standard 2.5 V for a CMOS input. The effect of this is to improve noise immunity on the input and the stage gain speeds up signal transition times.

For the interface to a computer there is a very simple RS232 implementation using just RxD (Pin 2) and TxD (Pin3) of the 9-pin sub-D connector. Most RS232 interface circuits require a voltage converter to produce a negative supply to allow the correct signal swing for RS232 signals. This design however, stores negative charge from the TxD line onto capacitor C3 to ensure that data output from the ELM323 will swing negative when T4 switches off. Resistor R12 limits the input current from the computer. R13 ensures that the RS232 input (IC1, pin 5) will be pulled low when the connector at K2 is disconnected. Transistor T4 drives RS232 data to the PC. The signal voltage will swing between +5 V (high) when T4 is conducting to –5.1 V (low) from the negative charge stored on C3 when T4 is off. Despite the simplicity of this RS232 interface it works very well.

Pin 4 of IC1 is tied high to force the microcontroller to send a line feed (LF) character after each carriage return (CR) character.

The four LEDs connected to pins 7, 8, 9 and 10 give a visual indication of data flowing at both the OBD and RS232 interface. The two groups of LEDs share a common current limiting resistor because data will only be flowing in one direction at any one time on either of the two interfaces (the ELM323 is not capable of true multitasking). The OBD bus may also be in its initialisation phase when RS232 data is sent to it so these limiting resistors are separate for the OBD and RS232 interface.

A crystal is fitted between pins 2 and 3 of IC1 along with two 27 pF loading capacitors. The capacitor values shown are typical but these may need to be changed depending on the crystal specification. The frequency chosen is that of the NTSC standard TV colour burst crystal and should be relatively cheap and widely available.

Construction and test

The layout for the circuit can be seen in **Figure 4**. Although the PCB is single-sided it does not need any wire links to be fitted. Connector K2 is a 9-way sub-D **socket** (do not make the mistake of fitting a plug).

As for the RS232 cable make sure that it is

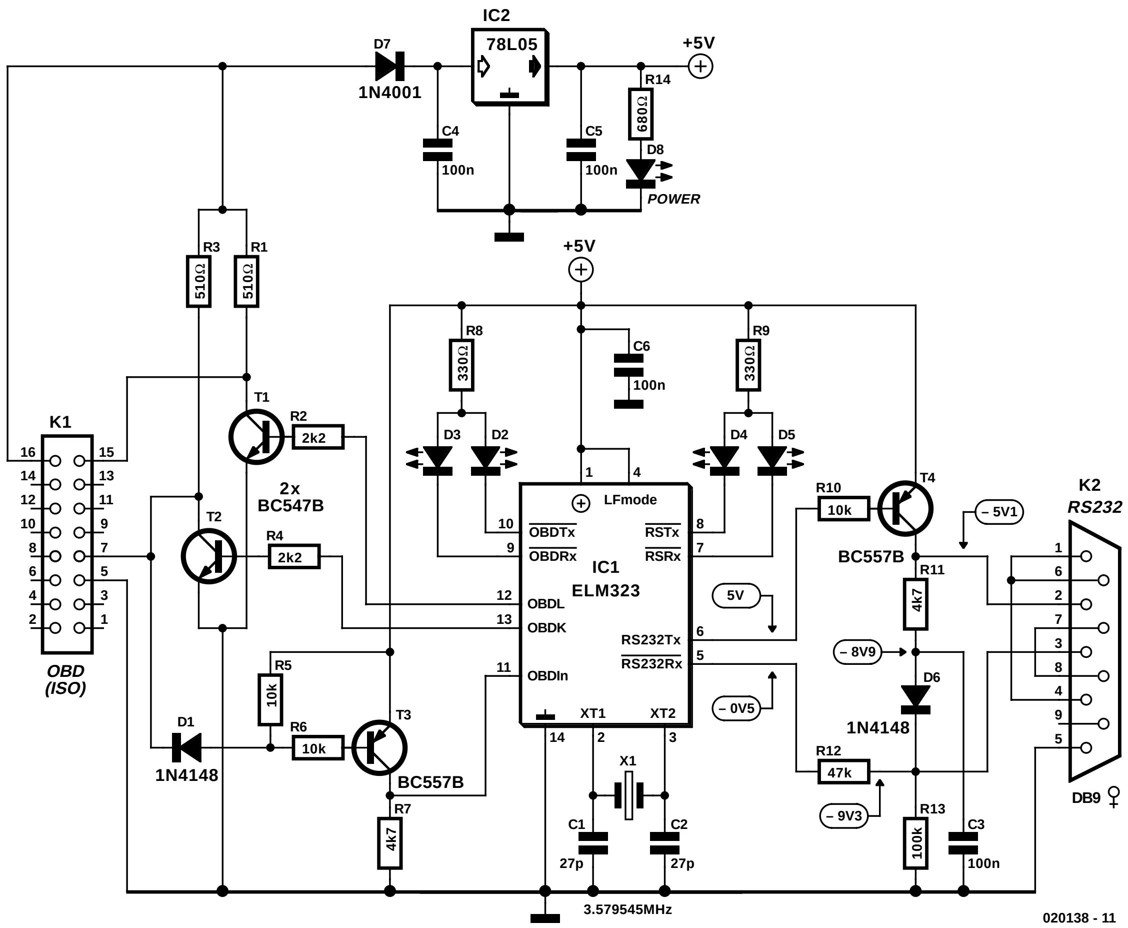


Figure 4. The OBD-2 to RS232 adapter circuit diagram.

a standard extension cable with the pins wired 1:1; **not** a null modem cable where the wires are connected quite differently.

Refer to the parts list for suppliers of the special components used in this design. (ELM323 and OBD plug). Hopefully a UK supplier emerges following publication of this article.

Once everything has been soldered in place check closely that all the components are correctly fitted and that there are no solder bridges on the track side. Now the circuit is complete you are probably anxious to try it out, but resist the temptation to plug it into the vehicle connector without first making some preliminary tests. Power up the circuit on the bench using either a 12 V mains supply or a 9 V battery together with a computer with a serial interface. Connect the power supply positive to pin 16 (+ 12 V) and the negative to pin 5 (ground). If everything is in order the green LED will come on to indicate that the circuit has power), The red LEDs will then briefly light. The 5 V supply can now be checked.

When everything is in order the next step is to connect the vehicle adapter to the serial port of a PC. It is now possible to check voltages around the circuit and compare them with the typical values given on the circuit diagram. The +5 V and –0.5 V shown on pins 6 and 5 of IC1 should be fairly close to these values but the voltage on pin 3 of connector K3 is derived from the PC and depends largely on the type of interface chip used in the external computer. It can be anything from about - 3 V to - 12 V. The voltage on capacitor C3 is dependent on the level of this voltage and should be about one diode drop (0.4 to 0.6 V) higher than the voltage measured at the TxD pin. In the circuit diagram TxD is assumed as –9.3 V which then gives -8.9 V after the voltage drop across D6.

Should you find that the voltage levels are substantially lower than the values shown on the diagram then it is important to find the problem before we can proceed any further. A short circuit on the RS232 interface will usually not result in damage to the PC because the signals are current-limited.

For a functional test of the serial interface we obviously need some software to run on the PC to send and receive serial data. The windows program accompanying this project (to be described in a follow-up article) would be a suitable tool for this job but if you are anxious to

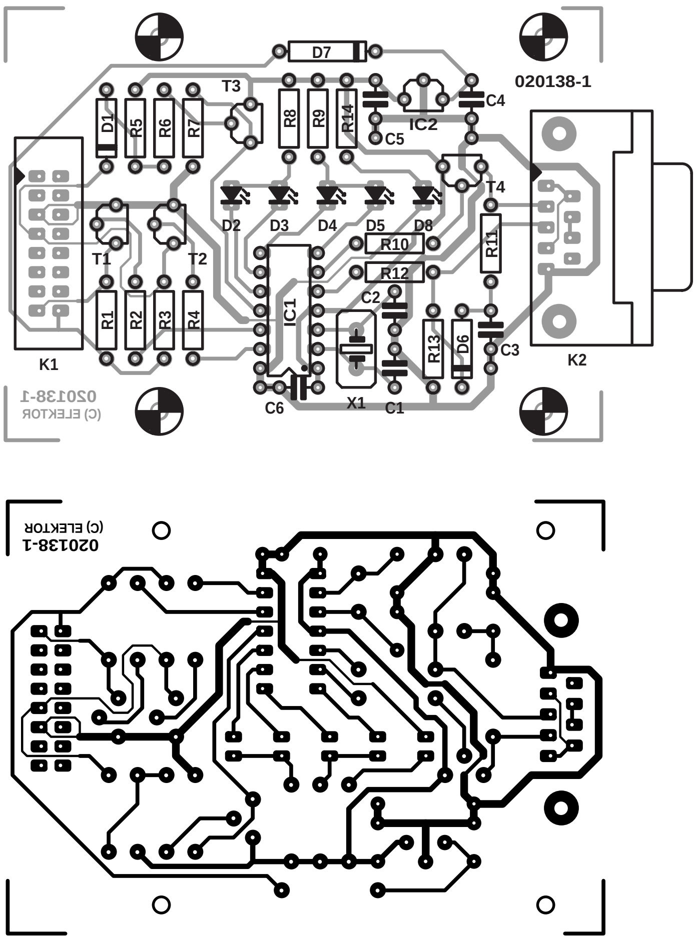


Figure 5. The PCB layout (board available ready-made).

COMPONENTS LIST

Resistors:
R1,R3 = 510Ω
R2,R4 = 2kΩ2
R5,R6,R10 = 10kΩ
R7,R11 = 4kΩ7
R8,R9 = 330Ω
R12 = 47kΩ
R13 = 100kΩ
R14 = 680Ω

Capacitors:
C1,C2 = 27pF
C3-C6 = 100nF

Semiconductors:
D1,D6 = 1N4148
D2-D5 = LED, red
D8 = LED, green
D7 = 1N4001
T1,T2 = BC547B
T3,T4 = BC557B
IC1 = ELM323 *
IC2 = 78L05

Miscellaneous:
K1 = 16-way boxheader
K2 = 9-way sub-D socket (female), angled pins, PCB mounting
X1 = 3.579545MHz quartz crystal (NTSC), 32pF parallel resonance
16-way OBD-2 plug *
PCB, order code **020138-1** (see Readers Services page)

* Suggested source for ELM323 and OBD plug kit:
Küster Datensysteme (KDS)
Geibelstrasse 14
D-30173 Hannover
Germany
Tel. (+49) 511 886059
Fax (+49) 511 8093329
E-Mail: OBD-Service@KDS-Online.com

* OBD connector parts also available from
www.scantool.net
www.autoxray.com

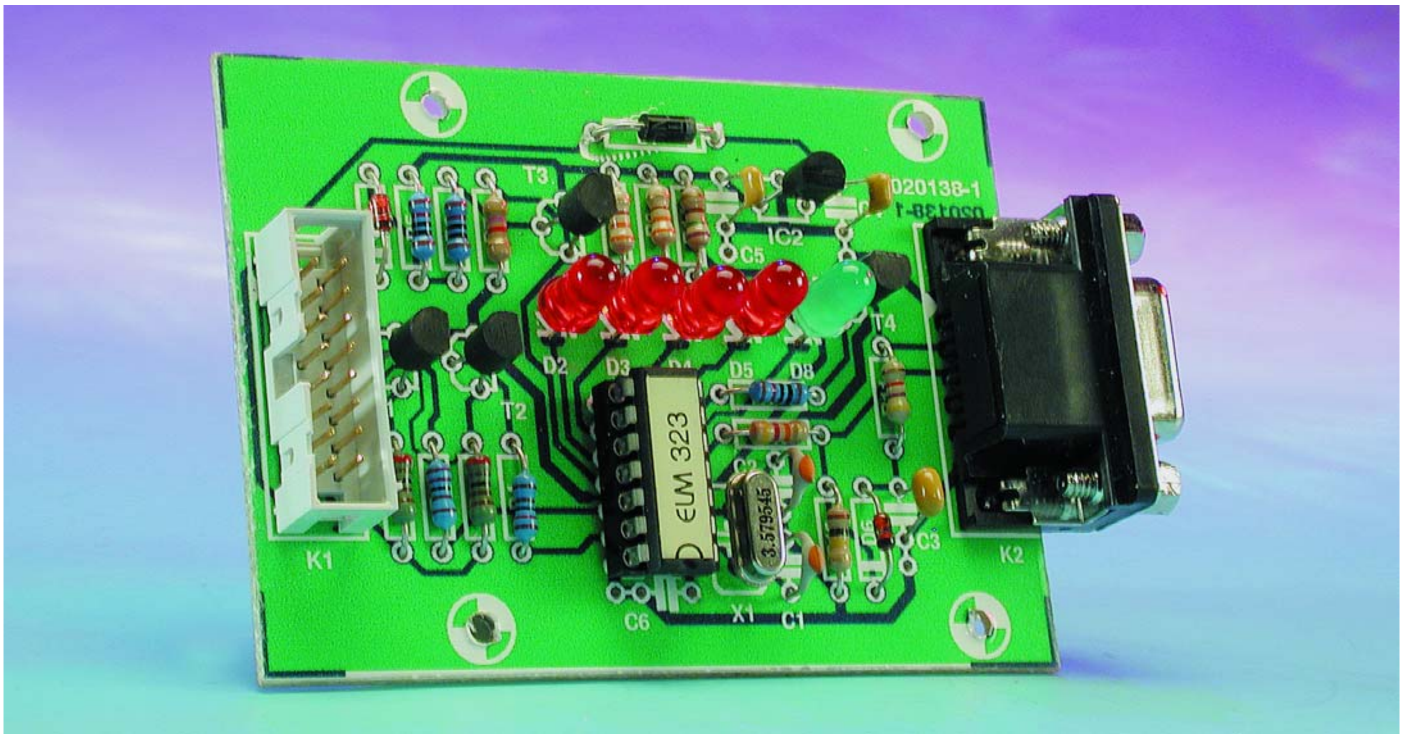


Figure 6.The fully populated PCB. The red LEDs indicate transmitting and receiving data through the ports.

test the interface then a terminal emulator program such as HyperTerminal will do. With any luck it should already be loaded on your Windows system, usually in 'Program Files' under 'Accessories'. If not then the program can be downloaded free of charge from: www.hilgraeve.com

HyperTerminal should be initialised with the following communication parameters: Data rate 9600 Baud, 8 Data bits, no Parity bit, 1 Stop bit and no handshake (no hardware handshake and no XOn/Xoff handshake). This is abbreviated to: **9600,8N1**.

Assuming that the interface is correctly connected you should see the four red LEDs light up and the following message will appear on the screen:

```
ELM323 v1.0
>
```

This gives the version number of the microcontroller software and also indicates that the IC is working correctly, and that the communication baud rate and receive path (the Rx input to the PC) is in order. The '>' character is a prompt issued by the ELM323 indicating that it is ready to accept data from the RS232 port. Messages from the PC can be intended for the internal use by the ELM323 or for the vehicles diagnostic system. The ELM323 handles all the communication and determines the message destination by analysing its character strings. Commands for the ELM323 are always prefixed with 'AT' commands just like the command set for Hayes compatible modems, while instructions for the OBD are ASCII coded hexadecimal numbers (0 to 9 and A to F). As a test, enter the command ATE1 (turn echo on) followed by the enter key, if there is no 'OK' response check the earth connection (Pin 5) and check the settings in the terminal emulator program to ensure that 'No handshake' is selected.

now for the car clinic

It is undoubtedly easier to communicate with the OBD system by using the Windows based program that we referred to earlier. This program will be featured in the next article on this project. If you can't wait, then the terminal emulator can meet out needs again; armed with a copy of the OBD-2 interface documentation and the ELM323 command set you should be able to unravel all of your vehicles secrets. We have gathered together some information to help you in this quest and all of this is available to download from the *Elektor Electronics* website under the following headings:

– Communication between the PC and the ELM323.
– AT commands.
– OBD bus initialisation.
– OBD commands.
– Diagnostic test modes.
– Read out and evaluation of fail codes.
– Clearing fail codes.
– ELM232 fail codes.

This detailed information should be of interest not only to anyone building this project but also to those of you thinking of developing an application based around the OBD system. In a forthcoming article we concentrate on the software for this project and include tips and code examples along with the other programs already mentioned above.

(020138-2)

Note:
Much of the material in this article is taken from a data sheet from Elm Electronics, Canada. The original can be downloaded from:
www.elmelectronics.com/dsheets.html

USB Driver Programming (2)

writing your own device driver

By M. Müller and C. Ehmer

In the previous issue of *Elektor Electronics*, we described how device drivers are used. Now it's time to modify a Cypress device driver. You only need a couple of programs for this, even if you have never worked with Microsoft Visual Studios. All of the necessary steps are described in minute detail.

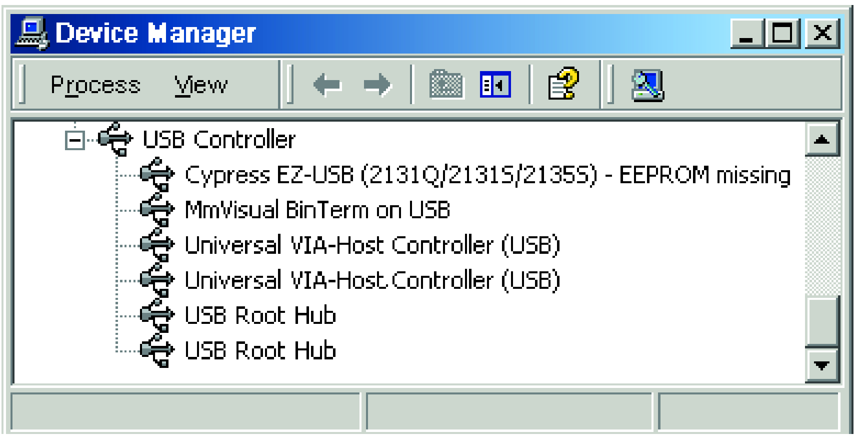
In the Windows Device Manager, you will see all currently connected USB devices listed under 'USB Controller'. **Figure 1** shows the entries for a USB device using the Cypress IC with no EEPROM and a USB data spy for BinTerm.

Writing your own complete driver 'from the ground up' is almost impossible for normal mortals. Fortunately, Cypress provides the EZUSB driver not only as an installable version, but also in the form of source code.

A device driver is divided into two parts, consisting of an INF file containing setup information and a SYS file, which is the actual device driver. The INF file can be edited using a simple text editor. A C++ compiler is necessary for generating the SYS file.

If you do not already know how to program in C, that's not such a big hurdle. You will need Microsoft Visual Studio 6 with the DDK (Driver Development Kit) addition for Windows 2000, and naturally you will also need the device driver source code from Cypress.

Be sure to install Visual Studio and related packages without modifying the installation paths. For the device driver, you will need the EZ-USB Development Kit, which you have



USB Controller
- Cypress EZ-USB (2131Q/2131S/2135S) - EEPROM missing
- MmVisual BinTerm on USB
- Universal VIA-Host Controller (USB)
- Universal VIA-Host Controller (USB)
- USB Root Hub
- USB Root Hub

Figure 1. The Windows Device Manager list with a user-generated driver.

Required items

System requirements:
Windows 2000 or Windows XP
Internet access

Programs and tools:
MmVisual BinTerm, version 2.2.2421 or later
Borland Delphi 6 or Microsoft Visual Basic
Cypress Semiconductor EZ-USB Development Kit
Microsoft Visual C++ 6
Microsoft DDK2000 Driver Development Kit for Windows 2000

Hardware
A working Cypress AN2131SC IC (with EEPROM) connected to the USB, or a BinTerm adapter connected to the USB. The latter circuit will be described in a coming issue of Elektor Electronics. *A BinTerm adapter connected to the USB is highly suitable for experimental projects, since it avoids the need to store program code in EEPROM. All programs are transferred to the device by BinTerm at runtime.*

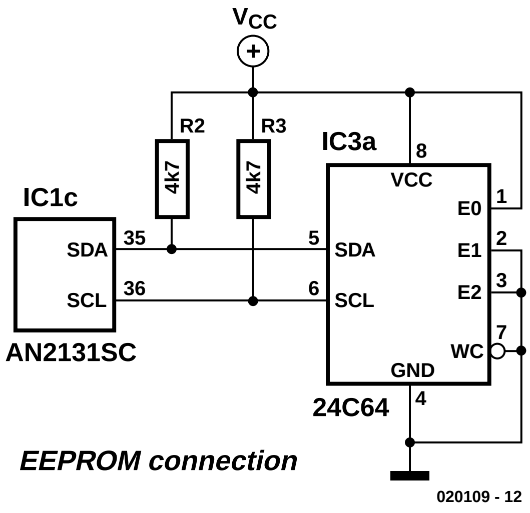


Figure 2. Circuit diagram of the EEPROM extension.

already installed and used in Part 1 of this series.

Setup information

Windows always needs an INF file for installing a device driver. Such a file contains the data specifying what should be installed, where it should be installed and how it should be installed. This file should be modified to meet your particular needs, since you don't want the Cypress device driver to conflict with your own driver. The items to be modified are shown in bold in **Listing 1**.

Before starting to edit the INF file, save it with a new name, and then enter the new name in the [xxx.Files.Inf] section. Next, change the company name ('Cypress') to your own name. The descriptors that Windows displays in the Device Manager window are entered in the [Strings] section. The device driver that you produce should also have a new SYS file. Give the existing SYS file a new name, such as 'MYSYS.SYS', and correspondingly change all 'EZUSB' identifiers to 'MYSYS'. Ensure that the file name of the device driver does not contain any non-standard characters or spaces — in other words, use only letters and numerals, with at most eight characters!

The most important items are the VID and PID numbers. They may lie in the range of 1 to 65534, and they are given in hexadecimal form. A VID number can be rented (for a fee) from the USB Organisation (www.usb.org) by anyone who wants to do so. If the VID is registered, the product is allowed to carry a USB symbol and the number is guaranteed to be unique. However, the fee is not exactly cheap.

The solution proposed here is to use freely

Table I

```
[Version]
provider=%Cypress%
[Manufacturer]
%Cypress%=Cypress
[Cypress]
%USB\VID_0547&PID_2131.DeviceDesc%=EZUSB.Dev, USB\VID_0547&PID_2131
[DestinationDirs]
EZUSB.Files.Ext = 10,System32\Drivers
EZUSB.Files.Inf = 10,INF
[EZUSB.Dev]
CopyFiles=EZUSB.Files.Ext, EZUSB.Files.Inf
AddReg=EZUSB.AddReg
[EZUSB.Dev.NT]
CopyFiles=EZUSB.Files.Ext, EZUSB.Files.Inf
AddReg=EZUSB.AddReg
[EZUSB.Dev.NT.Services]
Addservice = EZUSB, 0x00000002, EZUSB.AddService
[EZUSB.AddService]
DisplayName    = %EZUSB.SvcDesc%
ServiceBinary  = %10%\System32\Drivers\ezusb.sys
[EZUSB.AddReg]
HKR,,NTMPDriver,,ezusb.sys
[EZUSB.Files.Ext]
ezusb.sys
[EZUSB.Files.Inf]
ezusbw2k.Inf
[Strings]
Cypress="Cypress Semiconductor"
USB\VID_0547&PID_2131.DeviceDesc="Cypress EZ-USB (2131Q/2131S/2135S) - EEPROM missing"
```

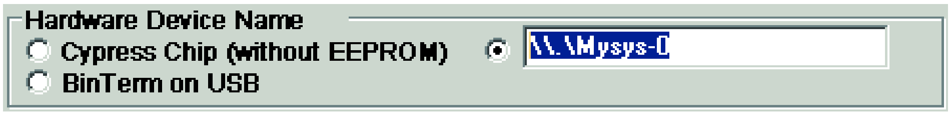

Hardware Device Name
Cypress Chip (without EEPROM) [✓ \\.\Mysys-0]
BinTerm on USB

Figure 3. Testing the device driver in BinTerm.

selected VID and PID numbers. This means that conflicts can arise with devices from other manufacturers. We suggest changing the VID number to '8B16' and the PID number to 'A001'.

The VID (vendor ID), PID (product ID) and DID (device ID) form a set of three 16-bit data words that the Cypress IC reports to the operating system. The VID and PID numbers are used to search for, install and load the device driver.

In order to be able to use your VID and PID with the Cypress IC, you will need an EEPROM. **Figure 2** shows how an 8-KB EEPROM can be added to the circuit used in Part 1 of this article series. Program the following sequence of numbers into the EEPROM, starting at address 0:

```
B0h       start byte
  copy IDs
16h, 8Bh VID = 8B16h
01h, A0h PID = A001h
01h, 00h DID = 0001h
```

The next time the USB device is plugged in, the Cypress IC will report this new identifier to the operating system, allowing Windows to install a suitable driver.

Using C++ to modify the SYS device driver

Start Visual C++ Studio, select 'Open Working Area' from the 'File' menu and open the file named

Table 2

```
NTSTATUS DriverEntry( IN PDRIVER_OBJECT DriverObject, IN PUNICODE_STRING RegistryPath)
{  NTSTATUS ntStatus = STATUS_SUCCESS;
   PDEVICE_OBJECT deviceObject = NULL;
   : :
   DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = Ezusb_ProcessIOCTL;
   : :
   DriverObject->DriverExtension->AddDevice = Ezusb_PnPAddDevice;
   : :
}
```

Table 3

```
NTSTATUS Ezusb_CreateDeviceObject(IN PDRIVER_OBJECT DriverObject,
                                  IN PDEVICE_OBJECT *DeviceObject, LONG Instance)
{  NTSTATUS ntStatus;
   WCHAR deviceLinkBuffer[]  = L"\\DosDevices\\Ezusb-0";
   UNICODE_STRING deviceLinkUnicodeString;
   WCHAR deviceNameBuffer[]  = L"\\Device\\Ezusb-0";
   : :
   deviceLinkBuffer[18] = (USHORT) ('0' + Instance);
   deviceNameBuffer[14] = (USHORT) ('0' + Instance);
```

Table 4

```
NTSTATUS Ezusb_ProcessIOCTL(IN PDEVICE_OBJECT fdo, IN PIRP Irp)
{  : :
   switch (ioControlCode)
   {
      case IOCTL_Ezusb_VENDOR_REQUEST: // = $00222014
         length = Ezusb_VendorRequest (fdo, (PVENDOR_REQUEST_IN) ioBuffer);
         if (length)
         {  Irp->IoStatus.Information = length;
            Irp->IoStatus.Status = STATUS_SUCCESS;
         } else
         {  Irp->IoStatus.Status = STATUS_SUCCESS;
         }
         break;
```

Table 5

```
case IOCTL_EZUSB_GET_DRIVER_VERSION:
{  PEZUSB_DRIVER_VERSION version = (PEZUSB_DRIVER_VERSION) ioBuffer;
   if (outputBufferLength >= sizeof(EZUSB_DRIVER_VERSION))
   {  version->MajorVersion = EZUSB_MAJOR_VERSION;
      version->MinorVersion = EZUSB_MINOR_VERSION;
      version->BuildVersion = EZUSB_BUILD_VERSION;
      Irp->IoStatus.Status = STATUS_SUCCESS;
      Irp->IoStatus.Information = sizeof(EZUSB_DRIVER_VERSION);
   } else
   {  Irp->IoStatus.Status = STATUS_UNSUCCESSFUL;
   }
}
break;
```

Table 6

```
typedef struct _VENDOR_REQUEST_IN
{    BYTE    bRequest;
     WORD    wValue;
     WORD    wIndex;
     WORD    wLength;
     BYTE    direction;
     BYTE    bData;
} VENDOR_REQUEST_IN, *PVENDOR_REQUEST_IN;
```

EZUSB.DSW. Next, check whether it is actually possible to generate the project. In the 'Generate' menu, select 'Generate ezusb.sys (F7)'. If everything is in order, '0 errors, 0 warnings' will appear in the message window. If the installation paths have been changed during the installation, all paths in the compiler and linker options must be modified accordingly. If the code cannot be generated without errors or warnings, the environment has not been configured properly.

Under C, it is nearly impossible to rename an entire project, so we will just leave it as it is. After everything is finished, assign your new name (MYSYS.SYS) to the SYS file.

The individual files

The most important file is *ezusbsys.c,* which contains all of the code for communications between the Windows USB device drivers and the application. In this article, we can only describe certain excerpts from the Cypress C source code.

The operating system first loads the SYS file, and then pointers to the other user-written functions are assigned to the entry label (*DriverEntry*), which is present in every device driver, to allow the operating system to directly execute the corresponding instructions. The operating system generates a new device using the function *Ezusb_PnPAddDevice*, which calls *Ezusb_CreateDeviceObject* and generates a virtual device name (similar to 'COM1' or 'LPT').

The main task is performed by the function *Ezusb_ProcessIOCTL*, which jumps to the Core32.dll function *DeviceIoControl* each time it is called.

You can modify the device name according to your own desires, as shown in **Listing 3**. Here you should change 'EZUSB' to 'MYSYS'. Since both of these names have the same length, the length declarations '18' and '14' need not be changed.

As soon as this driver is loaded by the operating system, the user-developed program can use *CreateFile* to establish a connection using the name \\.*Mysys-0.* In theory, the device driver can manage up to eight identical devices, which is why the name ends with a hyphen and a number. This routine is called for each *DeviceIoControl* instruction that communicates with the USB device (see **Listing 4**). A Request instruction for the USB device is handled using *Ezusb_VendorRequest*, but we don't want to discuss that in any more detail here.

All instructions that can be executed using the device driver are handled in the *switch (ioControlCode)* statement. Even version checking is handled using *DeviceIoControl*, as can be seen in **Listing 5**. The major, minor and build numbers are located in the *Version.h* file and can be modified as you see fit.

The sample Delphi program from the first part of the series now receives a new type declaration for the *_VENDOR_REQUEST_IN* record. This type is defined in the file *ezusbsys.h* from Cypress (**see** Listing 6). This structure, as well as other structures for *DeviceIoControl* instructions, can be directly incorporated into applications programmed in C, or

Table 7

```
#define Ezusb_IOCTL_INDEX  0x0800
#define IOCTL_Ezusb_VENDOR_REQUEST CTL_CODE(FILE_DEVICE_UNKNOWN,
                                       Ezusb_IOCTL_INDEX+5, METHOD_BUFFERED, FILE_ANY_ACCESS)
```

incorporated into Delphi applications in a derived form.

Links

Cypress Semiconductor	www.cypress.com
MmVisual BinTerm	www.mmvisual.de
Keil Software	www.keil.com
Borland Delphi	www.borland.com
	www.borland.com
Microsoft	www.micosoft.com
USB- Organisation	www.usb.org
Braintechnology	www.braintechnology.de
	(source for buying Cypress ICs)

All instructions for *DeviceIoControl* are defined further on in the same file (see **Listing 7**), with the definitions being computed using the macro *CTL_CODE*. The best way to find out how the numbers are put together is to study the Help file. If you want to manually check the index, it's worth knowing that the codes start at $00222000, with each subsequent code being incremented by $4. The code for *IOCTL_Ezusb_VENDOR_REQUEST* is thus $00222014.

Now press 'F7' to compile the source code. The result will be a file with the name *EZUSB.SYS*, which you should rename to *MYSYS.SYS*. Copy the modified INF file and *EZUSB.SYS* to a diskette. You have now produced a driver diskette for a Cypress IC with the VID and PID numbers you have defined.

Connect the USB device having the corresponding VID and PID identification to the PC. Windows will automatically recognise the new device. Use BinTerm to test the device driver (see **Figure 3**). If you select the 'USB Test' tab, you can enter the virtual device name and test communications with the Cypress IC.

(020109-2)

Stand-Alone EEDTS Pro

with controller addressing circuit

By S. van de Vries

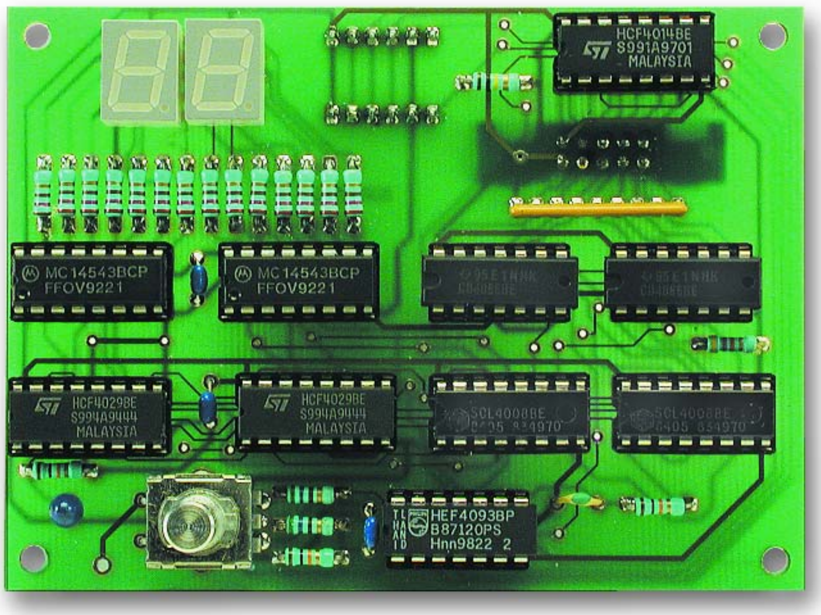
Not every user of the EEDTS Pro system needs to control his model trains using a PC. This article describes how the system can be used stand-alone and what is required. We also present a deluxe address programming circuit for your manual controllers.

Even shortly after the introduction of EEDTS Pro, it was evident that some users needed to be able to control the system without using a PC. One of the most frequently asked questions was whether the earlier EEDTS keypad could also be adapted to be used with the system and whether a keypad control would be coming. Another question that arose with regard to stand-alone operation of the controller was whether the 'fixed' addresses of the manual controllers could be changed without having to use a PC.

You might think that it shouldn't be so difficult to add these functions to the controller, but as can be seen from the old EEDTS address programmers and keypad circuit, such circuitry uses quite a few processor I/O lines or special I/O components.

That leads to another problem: practically all of the I/O lines of the EEDTS Pro controller are already in use, and the few that are not used are not routed to external connectors. The question was thus whether it was nevertheless possible to implement the above-mentioned functions within the current concept.

The solution was found in the S88 bus. This controller interface, which is quite busy collecting return data via the track when PC control is used, has precious little to do in stand-alone operation. That forms a good argument for using it for our purposes, with the additional advantage that absolutely no changes are necessary on the main controller circuit board. At the same time, we immediately gain access to 64 8-bit input channels, which is more than sufficient for our plans.



Starting up in stand-alone mode

The controller cannot decide on its own whether stand-alone or normal operation is desired. Its preference is for normal operation, in which all 64 EEDTS return signallers (or 32 Märklin S88 modules) are seen as return signallers (with addresses 1–64) that can be queried via the PC.

However, if the F0 key of manual controller 1 is held pressed while the main controller is started up or reset, the interpretation of the S88 bus changes as follows:

– the first 32 (8-bit) modules are

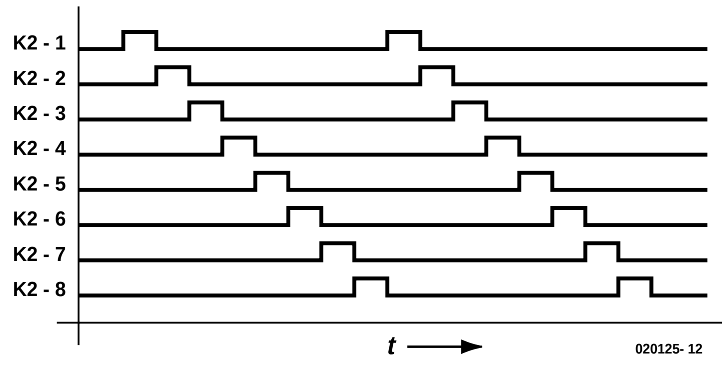


Figure 1. Timing diagram for the selection lines.

reserved for address programming and the keypad;

– the last 32 (8-bit) modules can still be used as needed as return modules (with addresses 1–32).

Controller addressing

Simple controller addressing

With the new master controller, locomotive addresses can be set within the range of 1 to 255. For a return module (8 bits), all of these addresses can be set in binary form, and in principle eight of them are needed to address all eight manual controllers. Although reserving eight modules for controller addresses would have the advantage of making it very easy to set the addresses, it would drastically reduce the number of possible keypad/return modules.

However, there are other options. Since address setting is intended to be done near the master controller, we can take advantage of the eight controller selection lines, which can be found on connector K2 of the master controller circuit board.

The controller selector lines are normally all Low, but when one of the eight controllers is selected, the processor sets one of these lines High (see **Figure 1**). During the selection interval, the function inputs are read, and thus it takes only a small additional effort to look at the S88 bus at the same time to see whether a new address for the controller may be present there.

To try out address setting, all you actually need is a single EEDTS detector module. The circuit shown in **Figure 2** shows this quite clearly.

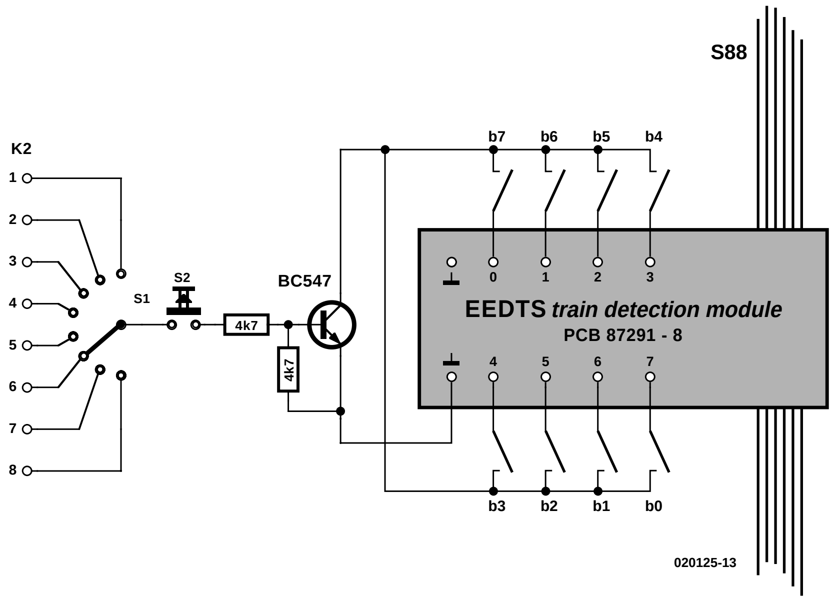


Figure 2. The simplest version of the address setting circuit.

In order to transform a normal train detector module into a genuine address programming module, you need a rotary switch with one wiper contact and eight positions. Although the well-known Lori switch has twelve positions, it can be converted into an eight-position switch by relocating the end stop.

This switch is then used to select one of the eight manual controllers, following which a binary address between 1 and 255 is set using switches b0–b8. Don't forget that the most significant bit (b7) must be connected to input 0!

Now hold pushbutton switch S2 depressed for a few seconds. After this, the address of the selected controller will have been modified. Make sure that the same address is not assigned to more than one manual controller. The master controller will allow this, but it will lead to confusing results on the track.

The purpose of the transistor with the two resistors is to invert the controller selection signal.

This train detector module should be the first module connected to the controller. If you do not need to modify the addresses of the manual controllers, but you do want to use keypad functionality, a 'dummy' train detector module (which is a train detector module with nothing connected to its inputs) must be connected at this location. (In Figure 2, everything except the train detector module has been omitted.)

When modified manual controller addressing is used, the addresses are always 'soft programmed'. This means that the settings are obliterated by a reset, so the master controller restarts with the default addresses.

Deluxe manual controller addressing

Quickly programming a binary address into your favourite manual controller so you can command an express train to stop before an impending collision is something best left to 'bit fanatics' with nerves of steel. However, for model railway owners with lower stress tolerance, we can offer a more luxurious version of the address programmer.

This version is shown in **Figure 3**. By and large, this circuit does the same thing as the simple version. An 8-position switch is used once again to select the controller, and even the pushbutton switch is still present, although here it is hidden inside the rotary switch that is used (contacts 4 and 5 are closed if you press on the knob).

This rotary switch (S1), which is called a 'rotary encoder', consists primarily of two changeover contacts with a common connection (pin 1). If the knob is turned to the right, as it moves between two positions contacts

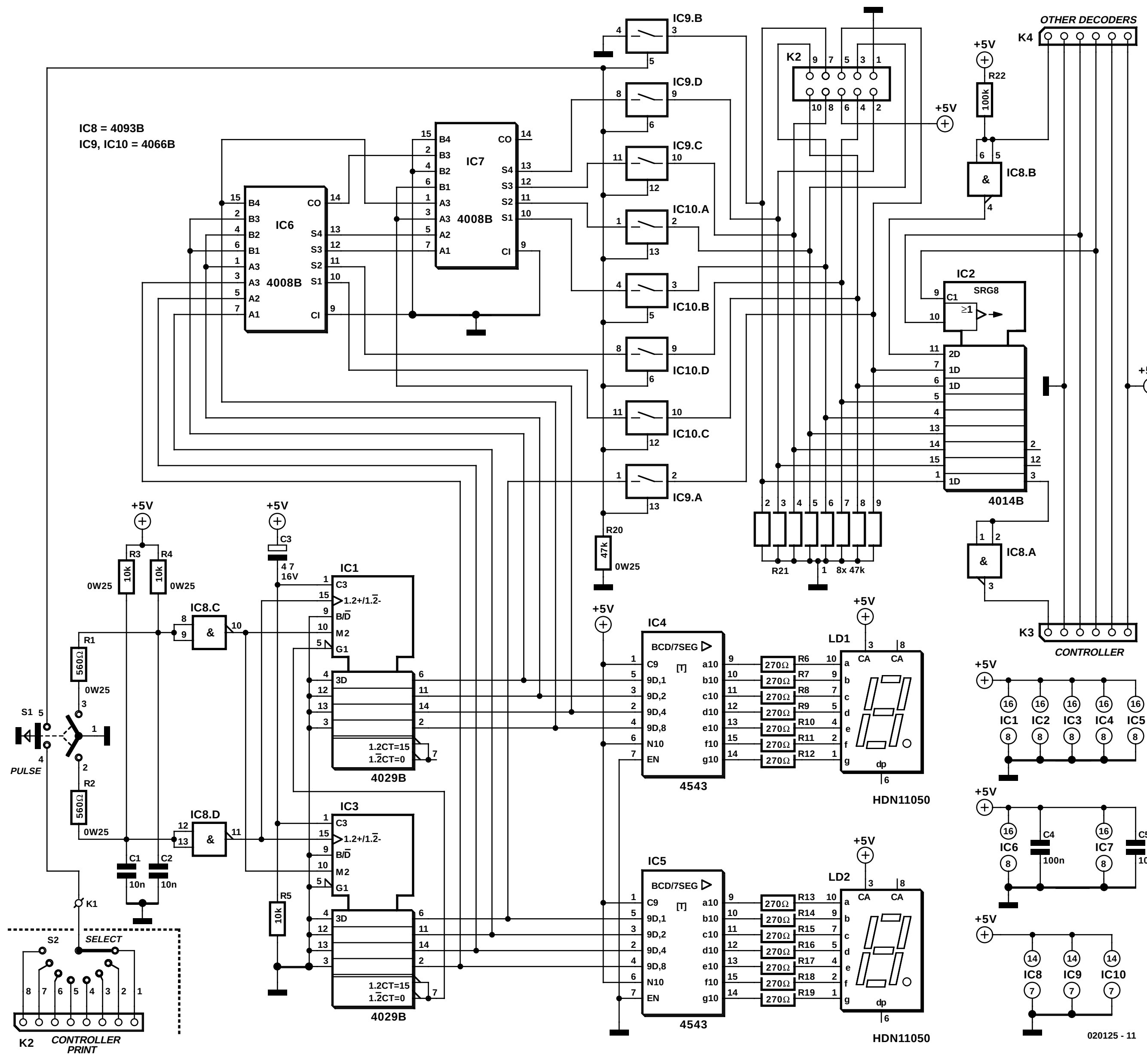


Figure 3. Schematic diagram of the controller address programmer.

1 and 3 close first, followed by contacts 1 and 2. Then contacts 1 and 3 open, followed by contacts 1 and 2. If the knob is turned to the left, as it moves between two positions contacts 1 and 2 close first, followed by contacts 1 and 3. Clearly, when the switch is being rotated to the right contacts 1 and 2 are still open when contacts 1 and 3 close, while with rotation to the left contacts 1 and 2 will already be closed.

Little more is needed to cause the 4049 CMOS synchronous counters (IC1 and IC3) to count up or down. R1–R4, C1 and C2, followed by Schmitt triggers IC8c and IC8d, are needed to produce reliable pulses from the bouncing contacts.

The two 4029 counters form a two-digit decade counter, so values between 1 and 99 can be set using the rotary switch. You might consider this to be a drawback, since after all, the simple version can set 255 addresses. However, this is an intentional choice, with the primary reason being that rotating the knob through 100 addresses (or at most 50 if you approach things right) is just about the limit of most people's patience. Also, since we're using a standard logic family, more than three times as many components would otherwise be required.

Two type 4543 BCD-to-7-segment decoder/drivers are used to convert the two decades from the 4029 counters into signals that drive two 7-segment displays (LD1 and LD2) via resistors R9–R16.

We now have easily readable addresses between 0 and 100 on a display, but what we ultimately need is to have the two 4-bit BCD codes from the 4029s converted into an 8-bit binary code that master controller can understand. This trick is performed by two 4-bit binary full adders (type 4008), whose operation we will not attempt to explain here. Suffice it to say that we have here is a sort of primordial computer IC.

The 8-bit binary code can only be

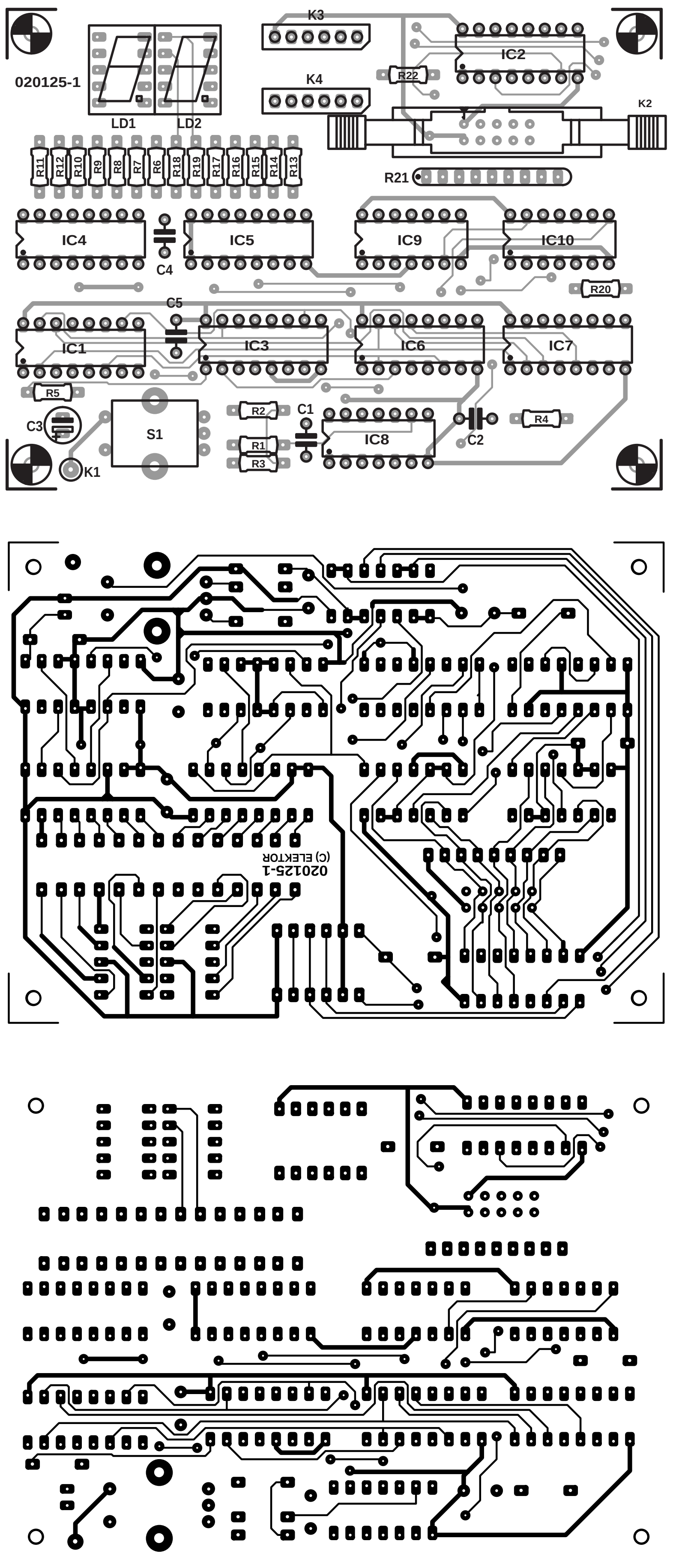


Figure 4. Double-sided printed circuit board layout for the circuit shown in Figure 3.

applied to shift register IC2 (a 4014) if the controller in question is being read out. The adder outputs are thus connected to the parallel inputs of the shift register via bilateral switches (IC9 and IC10).

To use the deluxe controller address programmer, first select a controller using 8-position switch S2. Then select an address using the rotary switch and press the knob to copy the address to the controller.

Super-deluxe controller addressing

The deluxe controller address programmer has advantage that the controller address can be easily seen while the controller is being programmed, but it also has some disadvantages, such as the fact that there is no permanent display of the addresses assigned to the manual controllers. It also requires a bit of extra work, since the controller must first be selected with a switch before it can be programmed.

The super-deluxe controller address programmer does not have these disadvantages, and it allows the addresses of up to eight

COMPONENTS LIST

Resistors:

R1,R2 = 560Ω
R3,R4,R5 = 10kΩ
R6-R19 = 270Ω
R20 = 47kΩ
R21 = 8-way 47kΩ SIL array
R22 = 100kΩ

Capacitors:

C1,C2 = 10nF, lead pitch 5mm
C3 = 4µF7 16V radial
C4,C5 = 100nF, lead pitch 5mm

Semiconductors:

IC1,IC3 = 4029
IC2 = 4014
IC4,IC5 = 4543
IC6,IC7 = 4008
IC8 = 4093
IC9,IC10 = 4066
LD1,LD2 = HDN1105 or LTS4801E

Miscellaneous:

S1 = rotary encoder, upright model, Conrad Electronics # 705594
S2 = rotary switch, 1 pole, 8 positions
K3,K4 = 6-way SIL header*
K2 = 10-way boxheader*
PCB, order code **020125-1** (see Readers Services)

* = mount at underside of board

manual controllers to be independently set while displaying the current locomotive address of each controller!

The only thing you have to do is to build two to eight copies of the deluxe address programmer board.

The selector switch can now be discarded, with each input instead being connected directly to one pin of connector K2 on the master controller board. In addition, all of the address programming boards are interconnected via the K2 connectors on the individual programming boards, using a 10-way flat cable and suitable headers (you can use a 1.4-m length of 10-strand flat cable with a total of eight headers attached every 20 cm). This cable supplies power to all of the programming boards and provides a sort of internal bus for the addresses to be set.

Naturally, only one address programming board (the master) can be fully populated and connected to the S88 bus. On all of the other address programming boards (the slaves), IC2 is not fitted, R21 must be omitted and connectors K3 and K4 may be omitted.

Many different combinations are possible, such as using only four address programming boards for four manual controllers (in which case it is recommended to 'hard code' the addresses of controllers 5 through 8 to 0, to avoid selecting an address that is already used by one of these controllers), or even a combination of one address programming board with a 4-position switch for four manual controllers (5–8, for example) and four address programming boards for direct address programming of manual controllers (for example 1–4).

The double-sided printed circuit board (**Figure 4**) is designed to have K2, K3 and K4 fitted on the back side (solder side), so the board with the rotary switch and display can be fitted directly behind a front panel (see **Figure 5**).

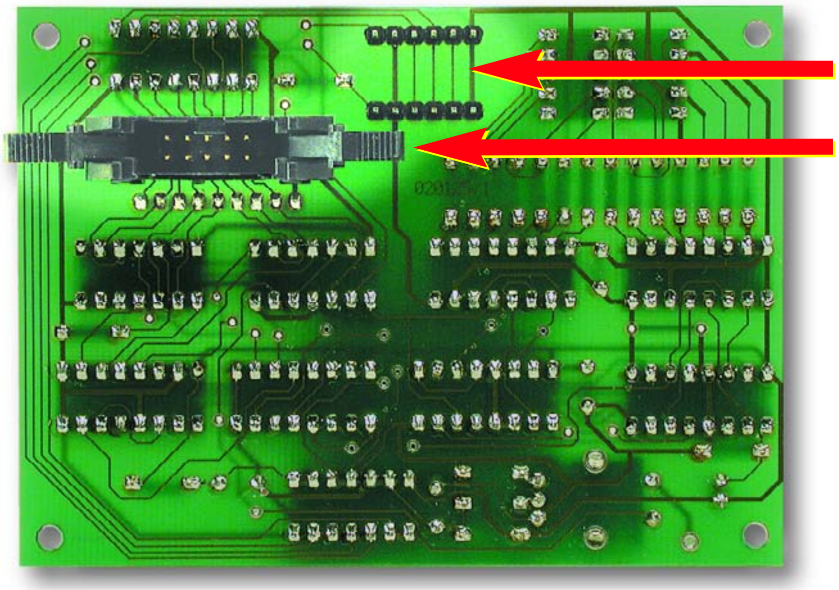


Figure 5. Detail of the connectors fitted to the solder side of the board.

EEDTS Pro keypad

Take four red pushbutton switches and four green pushbutton switches and connect them to an EEDTS train detector module (**Figure 6**) that is connected to the EEDTS Pro S88 bus. *Voilà* – you now have a genuine keypad!

This approach is primarily interesting for modellers who build 'real' control panels with physical pushbuttons, since in this concept the switches are not located on the printed circuit board, but instead are connected using wires. With the arrangement shown in Figure 6, up to 248 pushbuttons (31 × 8) could be built into such a panel for controlling loads. The entire control panel can be connected to the master controller using only five wires!

For people who regret the fact that Märklin no longer makes a switchboard, this represents a very attractive alternative.

As noted in the discussion of the address setting, the first module is reserved for setting addresses and must be present in all cases. After it, you can put together a chain of keypad modules.

Märklin S88 modules may also be used for the keypad modules (see **Figure 7**).

The first module following the address programming module has a fixed address of '1', the second '2' and so on, up to '31'.

This approach has been chosen for the sake of simplicity. Note that Märklin S88 modules occupy two

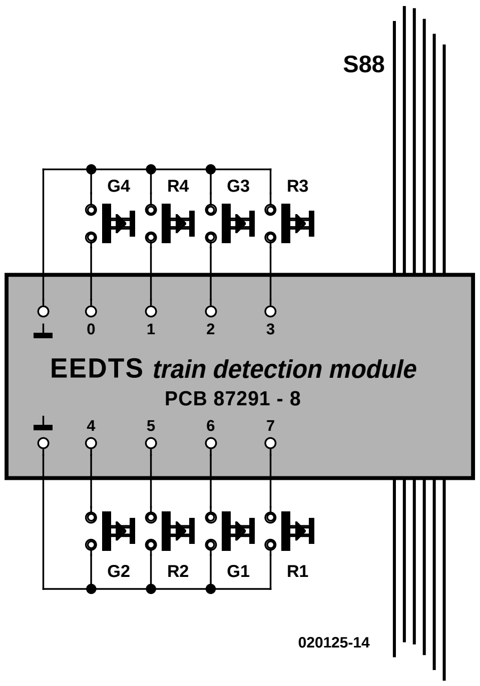


Figure 6. An EEDTS Pro keypad can be built using eight pushbutton switches and a train detector module.

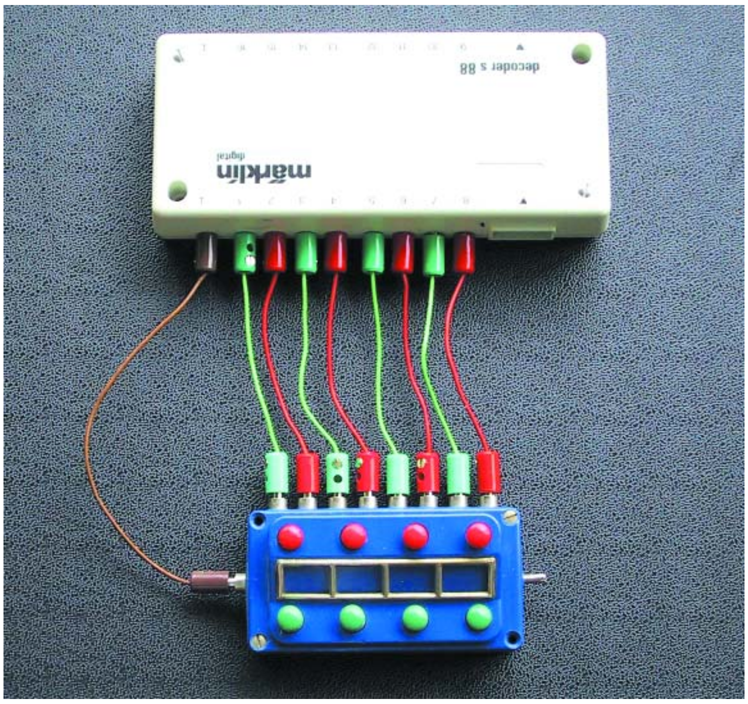


Figure 7. The Märklin 'keyboard' can serve perfectly well as an EEDTS Pro keypad.

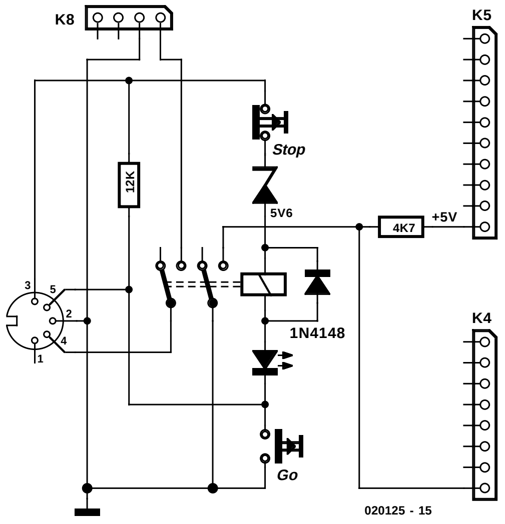


Figure 8. A booster interface with short circuit signalling via relay contacts.

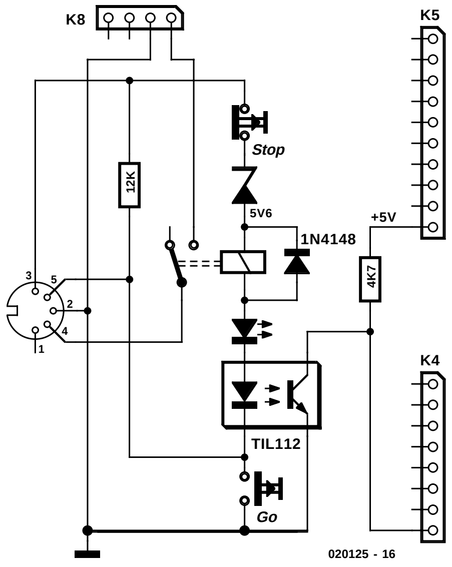


Figure 9. A booster interface with short circuit signalling via an optocoupler.

successive addresses. One thing that might be somewhat annoying is that the numerical sequence of the S88 bus modules does not correspond to the output sequence of the turnout decoders. Consequently, we have provided a summary in **Table 1**.

EEDTS Pro short-circuit signaller

We conclude with a small hardware bonus. In order to advise the EEDTS Pro 1.2 software of the fact that there is a short circuit on the track, the dropping out of the relay in the booster interface is reported back to the master controller using the only remaining free pin of K4 (pin 8).

Whenever a short circuit occurs, the EEDTS Pro software will stop sending commands, and it will only resume sending data after the short circuit has been remedied and the interface relay has again been engaged.

Two alternative booster interfaces circuits are shown in **Figure 8** and **Figure 9**. The first one uses a relay with two electrically isolated make or changeover contacts. If all of the contacts of the available relay are already being used, the circuit shown in Figure 9 can be used instead.

(020125-1)

Table 1. Output sequence of the turnout/signal decoders.

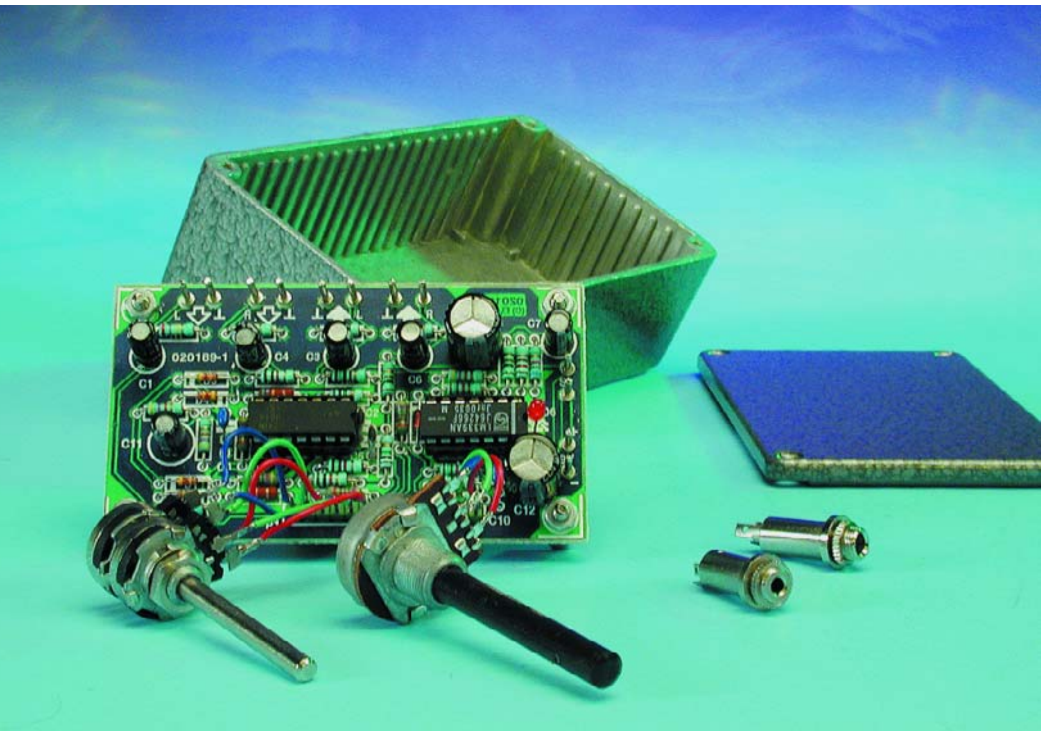
EEDTS Pro train address detector 87291-8	Märklin S88 lower address	Märklin S88 upper address	EEDTS turnout/signal decoder 87291-1	Märklin turnout/signal decoder K83
7	8	16	1a	1 red
6	7	15	1b	1 green
5	6	14	2a	2 red
4	5	13	2b	2 green
3	4	12	3a	3 red
2	3	11	3b	3 green
1	2	10	4a	4 red
0	1	9	4b	4 green

Audio Level Check for Line Input

for the PC sound card

Design by T. Giesberts

This simple control circuit allows the level of an external stereo audio source to be adapted to the sensitivity of, for example, a sound card. In addition, an LED indicates if any signal levels are present that can cause the input to be overdriven.



Standardisation is a long way off in many areas. This is certainly true for audio signal levels — a phenomenon that many of you will have encountered at one time or another, usually at the most inopportune moment. With equipment such as cassette decks, tuners, CD/DVD-players, one can never be quite sure what signal level to expect. Anything is possible from around 200 mV to 2 V, or so.

If you wish to connect a signal source to an input that does not have a good input level adjustment, then this can easily lead to unpleasant situations. A PC sound card, for example, is generally not provided with a level control. In addition, the maximum allowable input voltage of such a card is limited by the 5-V power supply of the chipset. This limits the input to signals of less than 1 to 1.5 V, before clipping occurs.

The drawbacks that are caused in practice by such fixed input sensitivities and limited output signal levels can be mitigated by preceding the relevant line input with the control circuit that is described here. The signal levels of 2 V that modern CD- or DVD-players often provide can be reduced to an acceptable maximum. On the other hand, the output level of 200 mV that older equipment typically source, can be amplified, something that will improve the signal to noise ratio.

The adjustment range of our circuit amounts to no less than ±20 dB, so every conceivable signal adaptation should be possible. In order to avoid being unpleasantly surprised by excessive signal levels, an LED indicator has been added which will indicate immediately when the output signal exceeds a certain (adjustable) value. Because the current consumption is a mere few mA, the level adapter can simply be powered from a 9-V battery; that makes it pleasantly universally deployable.

Control circuit

If the circuit were only required to attenuate, then we could have made it easy for ourselves and a simple passive voltage divider would have sufficed. But in this case we also wanted to amplify and, in addition, we would also like a tidy circuit in which the input and output-impedance are not affected by the position of the potentiometers.

That is why we decided to go for a relatively 'mature' design. Each channel consists of an input buffer and an adjustable amplifier stage. There is a single indicator common to both channels. In order to power the opamps symmetrically from a 9-V battery, a virtual ground point has been created with voltage divider R27/R28/C11. **Figure 1** shows the complete schematic.

The indicator will be described later, we will first take a look at the buffer and amplifier stage.

The function of the input buffers IC1c and IC1b is to provide a constant load for the attached signal source. The values are such that the input impedance is fixed at the well-known standard value of 47 kΩ. The opamps are connected as voltage followers, while resistors R3 and R12 bias them to half of the power supply voltage. Because the circuit is powered from an asymmetric power supply, a coupling capacitor is present at the input. This capacitor is always charged by R1 and R10 to prevent any unwanted sound effects when the signal source is plugged in. The coupling capacitor is followed by an additional protection network, guarding against high peak voltages. This network consists of two diodes and a resistor.

For practical reasons, an inverting topology was purposely selected for the adjustable amplifier stages IC1a and IC1d. By using the inverting inputs, the load on the potential divider for the virtual ground (R27/R28) is smaller, allowing higher resistance values and consequently a lower total current consumption.

If, for the moment, we limit the description to the left channel only, we can establish that

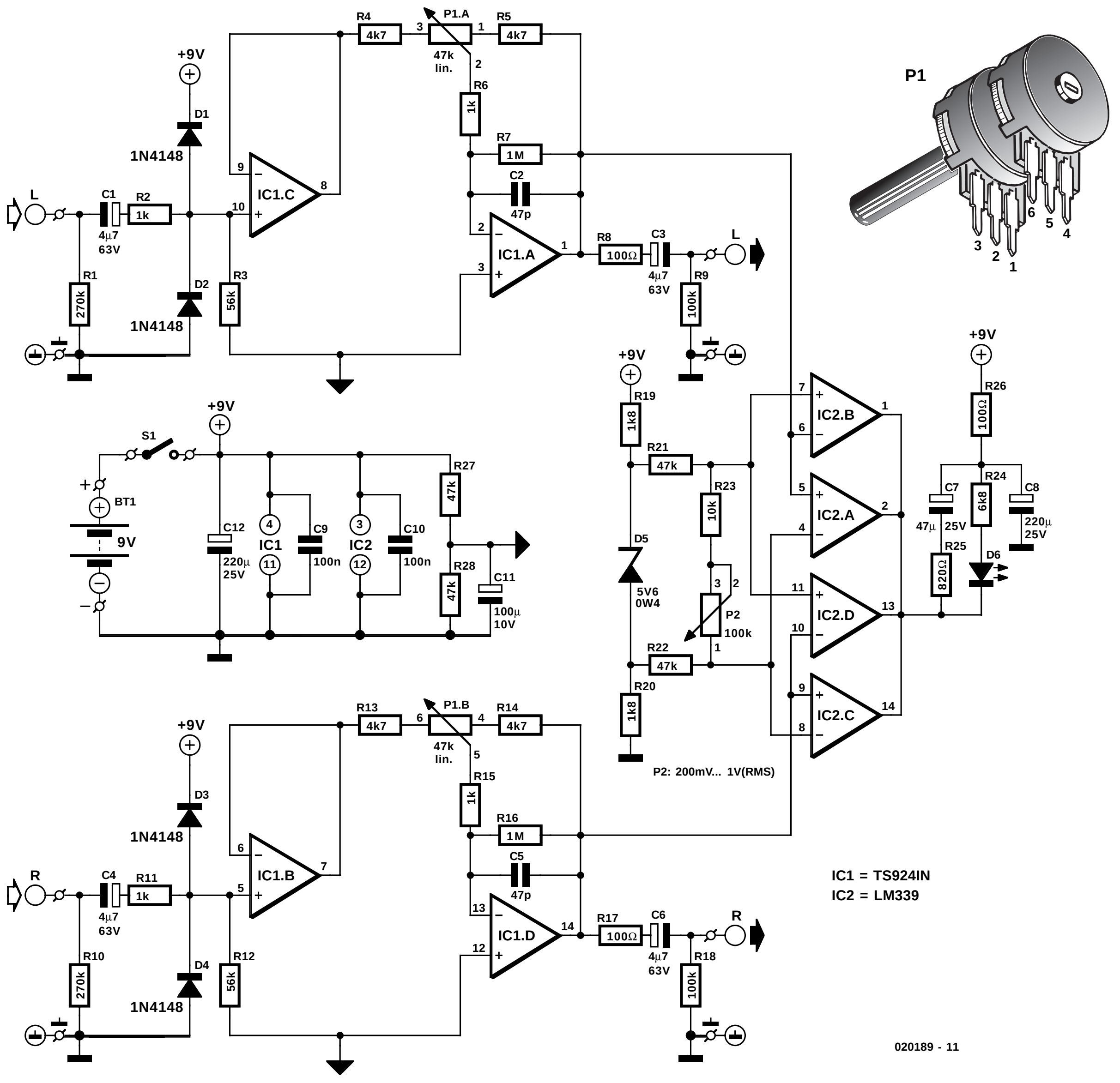


Figure 1. The level adapter actually consists of two separate circuits: the adjustment section and an overdrive indicator.

the maximum amplification of IC1a is equal to (R5+P1)/R4 and the minimum is equal to R5/(P1+R4). Because of small losses elsewhere, this results in an adjustment range from 10.5 to 0.09, which pretty much corresponds to ±20 dB.

Resistor R6 (R15) was added to prevent potential instability of the amplifier as a result of parasitic capacitance at the input of the opamp in case the wiring to P1 is a little bit long. Should the wiper of P1 be sporadically intermittent, R7 (R16) ensures that there is always negative feedback around the opamp and the output will always be nicely at half the power supply voltage. For stability reasons, C2 (C5) limits the bandwidth to 60 kHz at maximum amplification.

Another two components to complete the list: R8 (R17) ensures that the amplifier will be unconditionally stable even with a capacitive load at the output (because of long output cables, for example). R9 (R18) makes sure that the output electrolytic capacitor is always charged to prevent loud noises when the output load is plugged in.

It is worth mentioning that for IC1 we deliberately selected a rail-to-rail opamp, so that despite the low power supply voltage, reasonably high input voltages are still possible. The maximum is nominally more than 3 V_{RMS}, while with a nearly exhausted battery 2 V_{RMS} is still not a problem.

Indicator

The overdrive indicator is built around quad comparator IC2. This IC is used to build a window comparator for each channel. The window comparator compares the output signal of the adjustable amplifier stage with a variable reference. The outputs of IC2 are of the open-collector type and can therefore be simply connected together. If one of the comparator input signals exceeds the reference voltage, adjusted with P2, LED D8 will light up.

A few additional details regarding this circuit. The reference is set symmetrically around half the power supply voltage with the aid of R19, D5 and R20, so that both the negative as well as the positive half of the signal voltage is monitored. Because the zener current is set to a really small value of only 1 mA, the voltage drop across D5 is only about 5.3 V. R21, R22, R23 and P2 are dimensioned such that the reference voltage, adjustable with P2, corresponds to signal voltages from 200 mV to 1 V_{RMS}.

As can be seen in the schematic, LED D6 (and series resistor R24) is connected in parallel with an electrolytic capacitor (C7). This has been added to provide a certain amount of afterglow so that short duration signal peaks will become better visible. R25 limits the maximum charging current of C7 to a safe value for IC2. Finally, R26 and C8 decouple the power supply so that any potential noise from the switching of the LED has the least possible impact on the amplifiers.

Construction

An extremely compact circuit board has been designed for this circuit,

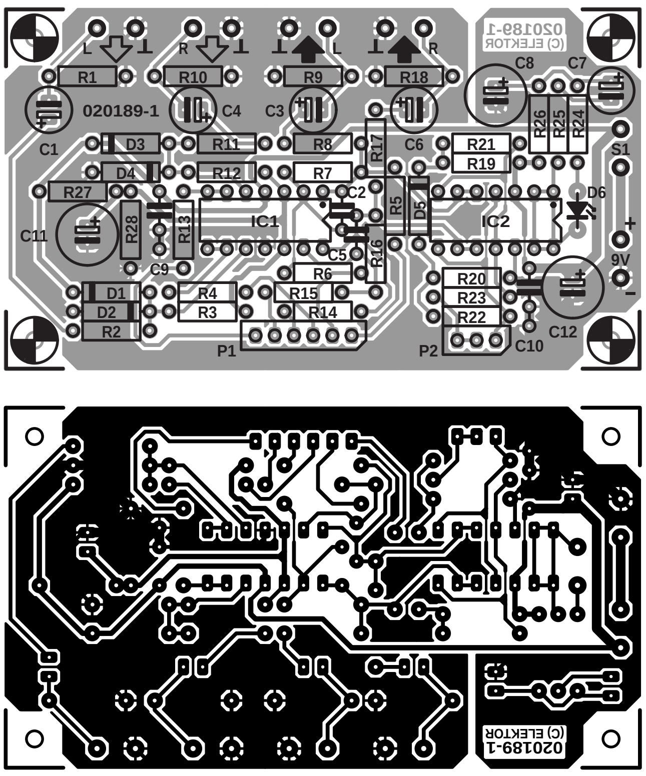


Figure 2. The dimensions of the PCB make an extremely compact construction of the level adapter possible.

COMPONENTS LIST

Resistors:
R1,R10 = 270kΩ
R2,R6,R11,R15 = 1kΩ
R3,R12 = 56kΩ
R4,R5,R13,R14 = 4kΩ7
R7,R16 = 1MΩ
R8,R17,R26 = 100Ω
R9,R18 = 100kΩ
R19,R20 = 1kΩ8
R21,R22,R27,R28 = 47kΩ
R23 = 10kΩ
R24 = 6kΩ8
R25 = 820Ω
P1 = 47kΩ stereo potentiometer, linear
P2 = 100 k mono potentiometer, linear

Capacitors:
C1,C3,C4,C6 = 4μF7 63V radial
C2,C5 = 47pF
C7 = 47μF 25V radial
C8,C12 = 220μF 25V radial
C9,C10 = 100nF
C11 = 100μF 10V radial

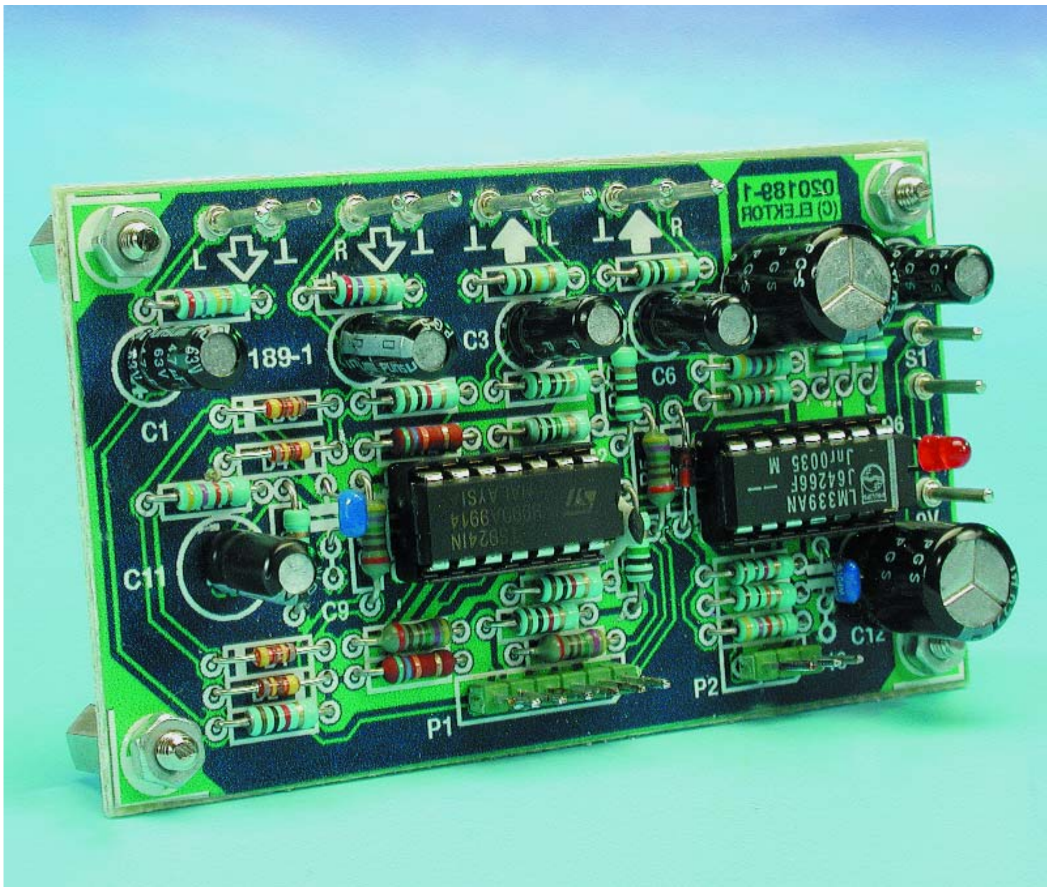
Semiconductors:
D1-D4 = 1N4148
D5 = zener diode 5.6V, 0.4W
D6 = LED, red, high-efficiency
IC1 = TS924IN ST (Farnell)
IC2 = LM339

Miscellaneous:
S1 = rocker switch, 1 make contact, for chassis mounting
BT1 = 9 V battery with holder
Two 3.5 mm stereo jack sockets, chassis mount
PCB, order code **020189-1** (see Readers Service page). PCB layout file also available from Free Downloads at www.elektor-electronics.co.uk

the layout and component overlay of which is shown in **Figure 2**. To make sure that the adjustment and indicator parts of the circuit don't 'bite' each other, there is a deliberate separation between the grounds of both sub-circuits. The circuit board is quite densely populated with parts, so construction should not be hurried.

The terminals for the inputs and outputs are nicely grouped on one side of the PCB, while on the opposite side two SIL headers can be found for the connections to P1 and P2. These components have deliberately not been fitted directly on the PCB so that everyone can determine for themselves what type of input and output connectors (e.g. mini-jack, cinch) and potentiometers to use. In addition, this also affords greater freedom in the selection of a suitable enclosure.

The wiring between the input and output connectors needs to be done with shielded cable. Standard hook-up wire can be used for the potentiometers, but with the channel separation in mind, it is best to keep the wiring to P1 as short as is possible. It is also recommended to use a potentiometer with a metal housing for this, the case of which is connected to ground with a short wire. Make sure that when wiring P1 and P2 that they are connected in such a way that the level increases when they are turned clockwise. The numbers of the connections shown on the schematic correspond with those of the SIL headers on the PCB. Once the circuit has been built into an enclosure, both potentiometers can be provided with a scale to indicate the amplification and indicator signal level respectively.



Some measurement results

Input impedance		47 kΩ
Input voltage max.	(THD = 0.1%)	3.2 V_{RMS}
Output voltage max.	(THD = 0.1%)	3.1 V_{RMS}
THD+N (B=80 kHz)	200 mV in, 20 mV out	0.027 % (20 Hz - 20 kHz)
	200 mV in, 200 mV out	0.005 % (20 Hz - 20 kHz)
	200 mV in, 2 V out	0.002 % (20Hz - 1 kHz)
		0.028 % (20 kHz)
THD+N (B=80kHz)	2 V in, 200 mV out	0.0026 % (20 Hz - 1 kHz)
		0.007 % (20 kHz)
	2 V in, 2 V out	0.002 % (20 Hz - 1 kHz)
		0.01 % (20 kHz)
Channel separation		> 66 dB (1 kHz)
		> 42 dB (20 kHz)
Current consumption	LED off	6.7 mA
	LED on	7.8 mA

The enclosure is preferably made from metal, because this minimises possible disturbances from any potential external noise sources. The metal of the enclosure needs to be connected to ground of the circuit. For example, a suitable enclosure is type Box1590N1 from the company H.O.D. However, the space inside this enclosure is just a little too small to fit the battery as well — in this case the next larger size will have to be used. For completeness' sake, we will mention that it was naturally the intention that D6 is mounted in such a way that it is visible from the outside.

As already mentioned, the current consumption of the circuit is so low (about 7 mA) that a power supply consisting of a 9-V battery can suffice. However, if you will be using the circuit in a more permanent application than a mains adapter is certainly more appropriate. A good adapter with regulated output is preferred. It is particularly important that the no-load voltage of the adapter does not exceed 12 V, because that is the maximum operating voltage of IC1.

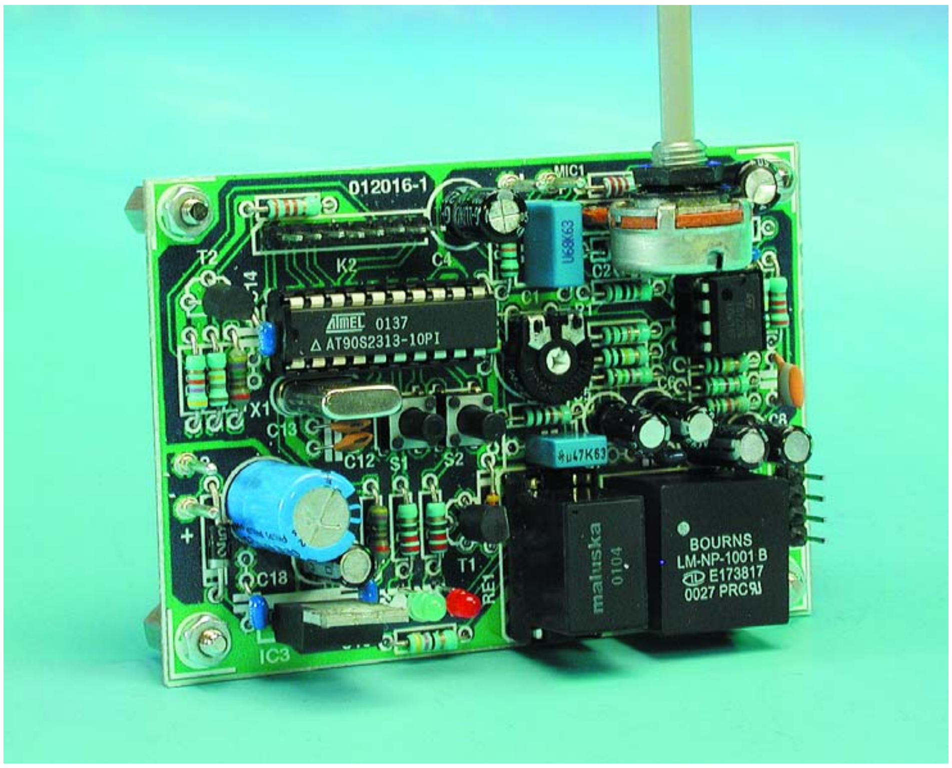
(020189-1)

Telephone Baby Monitor

cut through the static

Design by T. Finke

The usefulness of baby monitor alarms using (FM) radio has always been limited by poor range and its susceptibility to interference. This novel design uses a microcontroller to take a fresh look at the problem.



There are several different types of baby monitor on the market. The most common provides a short-range radio link (typically 100 m) between the transmitter unit in the nursery and the receiver unit carried by a parent. Another type uses audio to modulate a carrier frequency sent along mains wiring to the parents receiver plugged into a mains outlet. The RF model has a limited range and the mains modulation method requires that the parent cannot wander too far from a mains socket. Both systems are susceptible to interference.

The device described here goes about it in a different way: The unit is plugged into the house telephone socket and positioned in the nursery so that its in-built microphone can pick up sounds that the baby makes. The baby monitor microcontroller will analyse all sounds and determine if it hears a cry. When a cry is detected and persists for a given time, it will dial-out a pre-programmed number (typically the parents' mobile). There is no need to answer the call, a glimpse at the mobile's display will indicate that the house phone called you and junior is demanding attention!

Noise discrimination

The baby monitor circuit diagram is shown in **Figure 1**. The electret microphone MIC1 detects sounds in the nursery. Resistor R1 provides a bias from the supply rail while R7 and C4 are used to decouple any noise on the microphones supply. The AF (audio frequency) output signal is fed to the inverting input of IC2.A via capacitor C1. IC2 is a so-called rail-to-rail dual op-amp from ST (available from Farnell). This type of IC is necessary in this application because it allows us to set a high

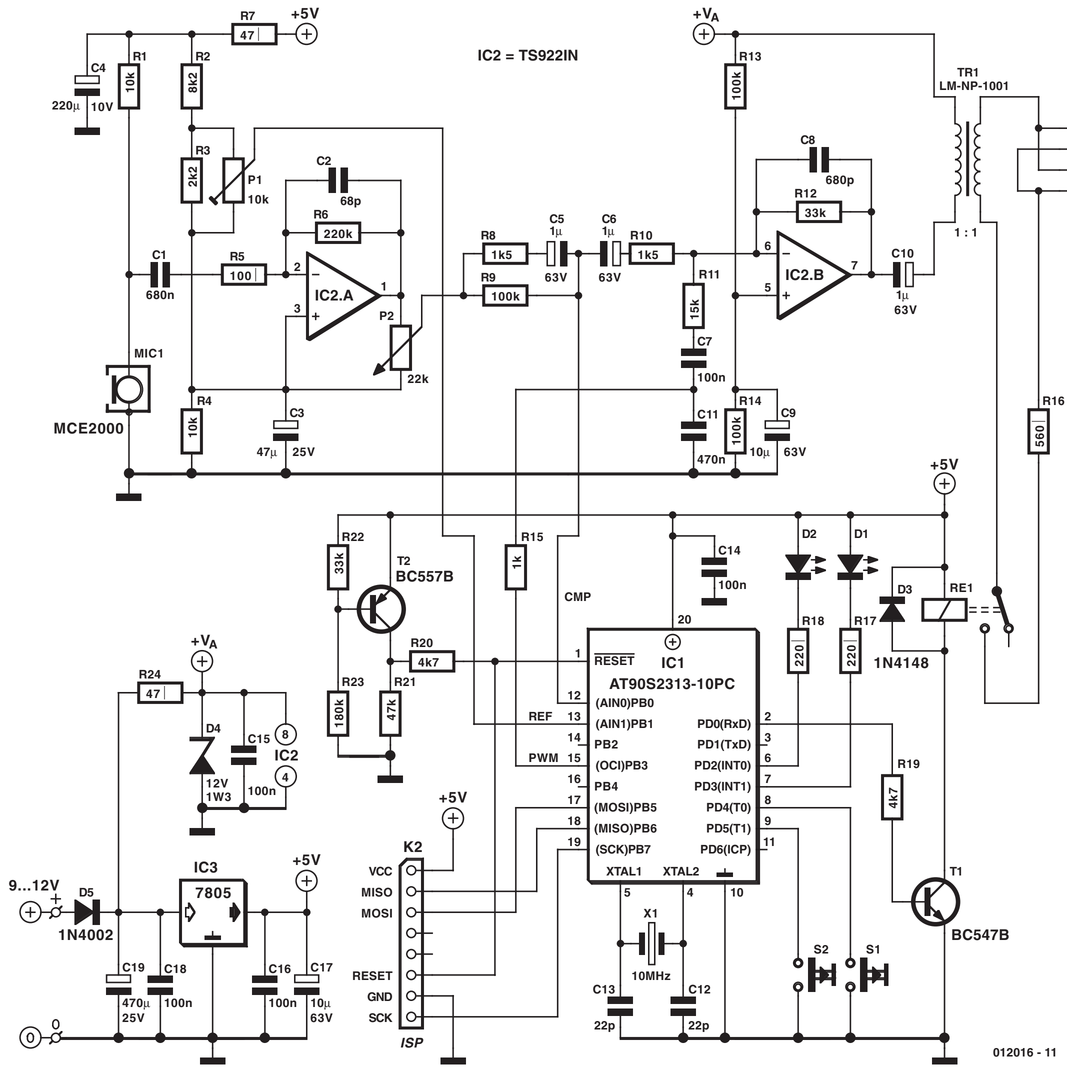


Figure 1. The circuit contains both analogue and digital circuitry.

level of amplification. The supply voltage to this IC is 12 V.

The audio signal is output from this stage on the wiper of preset P2. The signal quality is not especially hi-fi but it is not necessary for this application. Any high frequency ringing or oscillations are damped by capacitor C2.

The potential divider formed by resistors R2, R3, P1 and R4 produces a fixed voltage at approximately half-rail (2.5 V) for the non-inverting input of IC2.A capacitor C3 smoothes out this voltage.

The signal is now fed to the comparator input of IC1. **Figure 2** shows the internal features of this chip. One input of the comparator (AIN1) is connected to an adjustable reference voltage (REF). The other input (AIN0) is connected to the AF signal (CMP) and when this signal rises above the reference voltage the comparator output will switch and generate an interrupt in the microcontroller. If the microcontroller establishes that the sound is a baby crying it will change the AIN0 pin to an output and then output a zero to effectively short out the AF signal (R8 limits the short circuit current). Next the microcontroller sends out the DTMF signals on the PWM pin to dial the number. A more detailed description of this process is described under the heading 'DTMF signals'.

Amplifier IC2.B has a dual purpose to amplify audio or the PWM signal. The amplification factor required here is around two for DTMF signals and ten for audio signals. Resistor chain R13 and R14 provide a reference voltage of half V_A for the non-inverting input of IC2.B. Capacitor C9 smoothes any noise on this reference voltage. Both amplifiers have a bandwidth of 10 kHz, which is more than enough for this application.

Brown-Out protection

Brown-outs occur when the mains supply undergoes a brief interruption, just enough to cause your houselights to dim momentarily. These interruptions cause havoc in microcontroller systems that do not have brown-out protection. The microcontroller supply voltage starts to fall, producing unreliable operation but then recovers again before a reset can be generated. The processor can mis-read data values and instructions, ending up in an undefined state and systems using EEPROMs can have stored values corrupted. The brown-out protection circuit used here is built around transistor T2 and is taken from an ATMEL application note. This configuration is more economical than commercial brown-out protection IC's. The circuit quickly generates a RESET pulse when it detects a dip in the supply voltage.

Telephone connection

A length of telephone cable terminated with a line plug is suitable for connecting the baby monitor to a UK telephone socket. The type 431A connector is the UK standard line plug. The plug moulding is six-way but contacts are only fitted to the inner four positions numbered 2, 3, 4 and 5. Pins 2 and 5 should be connected to 'a' and 'b' on connector K1. The polarity is unimportant.

When the baby monitor dials out, transistor T1 conducts and the contacts of RE1 close, loop current is limited by R16 and the secondary winding of TR1. DTMF dialling tones are now output to the primary side of TR1 from IC2.B.

User interface

The baby monitor user interface consists of pushbuttons S1 and S2 to control the unit and enter data while LEDs D1 and D2 display its status. This interface is not the most elegant but you will seldom need to change system parameters so the additional expense of a dedicated keypad and LCD can hardly be justified in this instance.

A mains adapter unit supplying 9 to 12 V provides power for the baby monitor. A protection diode (D5) ensures that reversing the power supply leads cannot damage the input. C19 acts as a smoothing capacitor to reduce any supply ripple. IC3 provides 5 V for the circuit while the opamp supply is taken directly from the unregulated input from the mains unit. Diode D4 ensures that this supply does not exceed the 12 V maximum for the opamp, its purpose is to protect against the higher no-load voltage output from unregulated mains units. Do not expect it to be able to cope with a continuously higher input from a mains unit rated at more than 12 V.

It would be possible to power the unit directly from the telephone network but such practices are frowned upon by telephone authorities and besides on safety grounds it is better to maintain a good galvanic separation (no dc path) between the relatively high voltage of the phone network and the baby monitor unit.

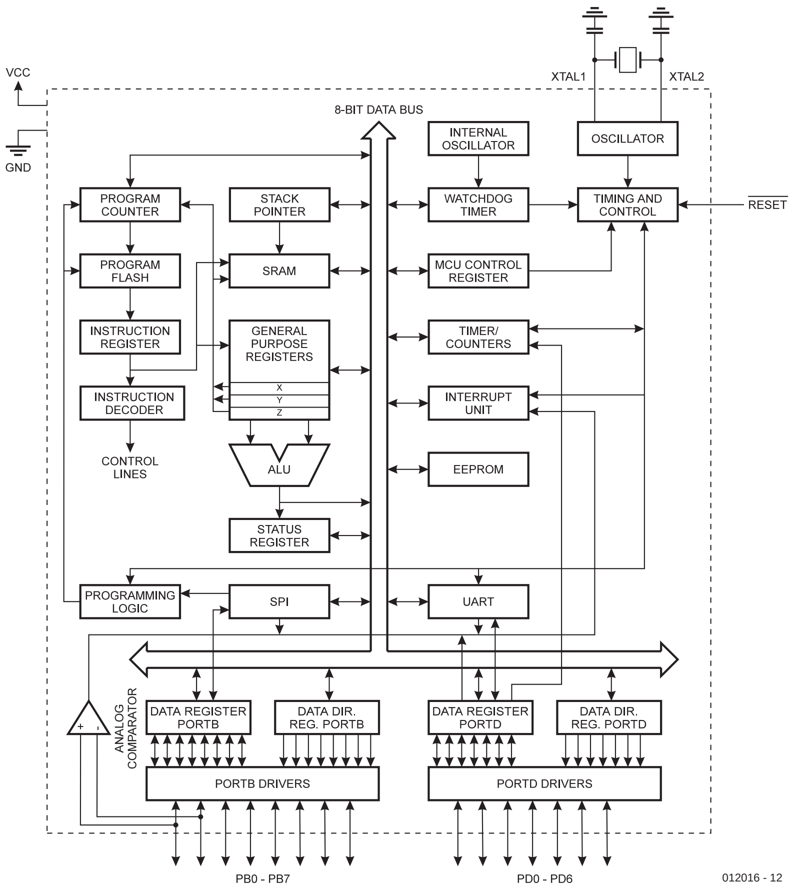
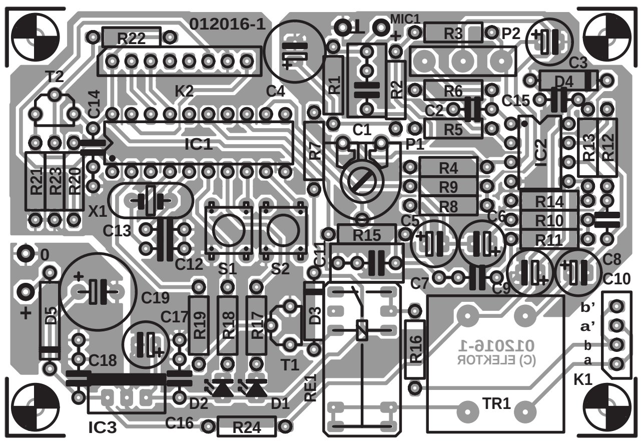


Figure 2. The controller uses an on-board comparator.



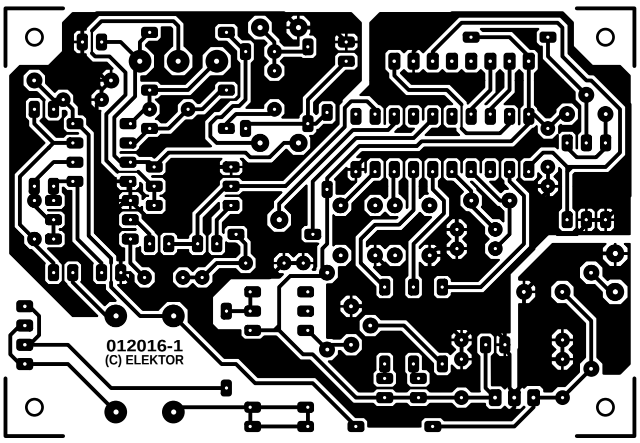


Figure 3.Single-sided PCB layout for the baby monitor.

The microcontroller's clock frequency of 10 MHz may seem high but it is necessary to achieve the accuracy for DTMF tone generation. The microcontroller can be programmed via connector K2. The pin-out of this connector corresponds to the standard used by Lattice to program their ispCPLDs. The same cable can be used to download the program to Atmel as well as Lattice chips. If you plan to order a pre-programmed controller from the Publishers (and have no need to re-program it) then you can omit connector K2.

Assembling and adjusting

The compact single-sided PCB is shown in **Figure 3.** Fit all the components to the board starting with the small items and finishing with the largest. There are no wire links necessary with this layout. For the pushbuttons S1 and S2 choose types with long buttons so that they protrude through the plastic case for easier access.

The microphone insert should be mounted under the case for protection but don't forget to make an opening in the case so that sound can reach the microphone. The telephone cable outlet from the case should be fitted with a strain relief exit grommet.

A final check of the PCB is always a worthwhile activity before the circuit is powered up. Make sure all components are correctly positioned and that there are no solder bridges between the PCB tracks. At power-up the green LED indicates that the unit is operational. If a sound is detected, this LED will start to flash. Pressing pushbutton S1 will turn off the unit and the green LED will go out. Preset P2 is used to adjust the sound level so that the cry from the

COMPONENTS LIST

Resistors:
R1,R4 = 10kΩ
R2 = 8kΩ2
R3 = 2kΩ2
R5 = 100Ω
R6 = 220kΩ
R7,R24 = 47Ω
R8,R10 = 1kΩ5
R9,R13,R14 = 100kΩ
R11 = 15kΩ
R12,R22 = 33kΩ
R15 = 1kΩ
R16 = 560Ω
R17,R18 = 220Ω
R19,R20 = 4kΩ7
R21 = 47kΩ
R23 = 180kΩ
P1 = 10kΩ preset H
P2 = 22kΩ potentiometer, linear, mono, miniature version

Capacitors:
C1 = 680nF
C2 = 68pF
C3 = 47µF 25V radial
C4 = 220µF 10V radial
C5,C6,C10 = 1µF 63V radial
C7,C14,C15,C16,C18 = 100nF ceramic
C8 = 470pF
C9,C17 = 10µF 63V radial
C11 = 470nF
C12,C13 = 22pF
C19 = 470µF 25V radial

Semiconductors:
D1 = LED, red, 3mm
D2 = LED, green, 3mm
D3 = 1N4148
D4 = zener diode 12V 1.3W
D5 = 1N4002
T1 = BC547B
T2 = BC557B
IC1 = AT90S2313-10PC, programmed, order code **012016-41**
IC2 = TS922IN (Farnell)
IC3 = 7805

Miscellaneous:
K1 = 4-ay SIL header
K2 = 8-way SIL header
S1,S2 = pushbutton, 6x6 mm
Re1 = subminiature relay 16x119x11.5 mm with SPDT contact, e.g., Maluska FRS1B-S DC 5V, (5 V, 56 Ω, Conrad Electronics # 505188)
X1 = 10MHz quartz crystal (C_{load} = 32pF, parallel resonance)
Tr1 = line transformer, Bourns LM-NP-1001 B)
MIC1 = electret Microphone element (e.g., Monacor/Monarch MCE2000)
PCB, order code **012016-1**
Disk, C source code and hex files, order code **012016-11**
(see Readers Service page)

DTMF Signal Generation

Simple DTMF generator IC's (like the TP5088 from NS) are practically no longer available. These days DTMF generators come with all sorts of bells and whistles that we do not need in this application. They are generally difficult to find and have lots of interface signals to take care of (The MT80444 needs the Φ2-Signal from the obsolete 6502 processor). A good solution to this problem is to generate these tones in software.

DTMF stands for Dual Tone Multi Frequency signalling (tone dialling). Eight frequencies are used comprising a group of 4 low tones and a group of 4 high tones. Each of the 16 possible keys on a telephone keypad is represented by a different combination of one high tone and one low tone. The reasoning behind this was that it is unlikely that these sounds would occur in normal speech patterns. The frequency of all the eight tones needs to be accurate to within 1.5 % otherwise the exchange equipment may not recognise the DTMF signal as a key press.

A D/A converter would be an ideal component to generate the tones necessary for DTMF signalling. Unfortunately the Atmel microcontroller used in this project does not have such a device on-board. The controller is however equipped with a Pulse Width Modulated (PWM) output and this can do the job just as well. This output has a resolution of 8 bits with a clock frequency of 19.6 kHz (fCK/510). The dual sine waves are produced by loading the PWM counter with values from a look-up table to control the width of the pulse. The spectrum of the resultant waveform shows peaks at the two sine frequency fundamentals and a peak by f = 0 (this dc component is removed by capacitors C7 and C11). Other frequency components of this signal are at the PWM switching frequency. The low-pass filter formed by R15 and C11 ensures that these square wave components on the output signal are removed to leave just the two sine waves. The corresponding amplitude components for the high and low tone groups are stored in a sine wave table. If the stored period does not result in the necessary frequency accuracy then two or three different values are used. Altogether the sine wave table contains approximately 200 bytes of data.

baby does not overload the input to IC2.B. This setting is best performed by viewing the output of IC2.B with an oscilloscope while adjusting P2. Lastly, the monitor trigger level is adjusted by preset P1. This adjustment ensures that the unit does not respond to background noises in the nursery. The baby monitor will not dial-out each time this comparator is triggered, the frequency of the sound must be above a threshold, and the sound must persist for a time greater than a minimum period. Both of these parameters are stored in EEPROM and can be altered at will.

When the microcontroller recognises the sound as a baby cry it will switch relay RE1 to connect the monitor to the telephone line (equivalent to lifting the receiver of a phone), the red LED will switch on continuously and then it will call the stored telephone number by using tone dialling. After a preset ring time (two to three rings) the monitor will hang up. The monitor will now remain in a 'standby' state for a pre-programmed period to prevent it from being continually triggered. The red LED will flash during this period, finally the unit will switch back to its operational mode.

Dial out

Pressing pushbutton S2 will immediately call the dial-out phone number stored in memory. External equipment (e.g. house alarm or temperature monitor) with a normally open type relay or open collector output can wire this output in parallel to S2 so that the baby monitor will dial out when the alarm condition is detected.

Enter the number

The method of entering the dial-out phone number is a little long winded but once it is stored in EEPROM the chances are that you will not need to change it too often. A long press of pushbutton S1 (longer than two seconds) will switch the monitor unit into programming mode (the red LED will come on and the green will go out). Now enter the first digit of the dial-out phone number: press pushbutton S1 corresponding to the digit (if the number is zero do not press S1 at all) after tapping in the digit press S2 to enter it and move on to the next digit of the number. Carry on until the complete number is entered and then a long press of S1 will finish the process. If you need to program a time delay in the dialling process (The number may involve a connection through an exchange) then just press S1 10 times where you want the delay in the phone number. If you make a mistake when entering a digit then two long presses of S1 will return you to the start of the number and you can make another attempt at re-entering it.

Checking the number

The stored dial-out phone number can be checked by pressing and holding pushbutton S2. The red LED will now light continuously and the green LED flashes to indicate the value of the first programmed digit, repeating after a short delay (the LED will remain off if the number is zero). Pressing S2 then moves on to the next digit. After the last digit the red LED will go out and the green LED will light continuously to indicate that the unit is ready for use.

Programming

All the baby monitor software is written in 'C'. Constants used in the program (the noise detector minimum frequency, various timer values and the dial-out phone number) may be changed without resorting to Compiler software but by using a simple programming device to directly overwrite the values stored in EEPROM:

Address	Function	Default
\$00-\$1F	Dial-out phone No. (ASCII coded). \$00 is the terminating character.	
\$20	Low frequency cut-off. Sounds above this frequency are classified as a baby cry. ($f_g = X \cdot 38$ Hz)	10
\$21	Length of the monitoring window after sound detected $2^X \cdot 6.5$ ms	10
\$22	The number of 'noisy' time periods (of 6.5 ms) in the monitoring window before the alarm is triggered.	80
\$23	Time between dialling and hanging up (ring time). $X \cdot 6$ s (approx.)	4
\$24	The baby monitor is in standby mode after dialling-out. It will wait for this period of time ($X \cdot 16$ s) before listening for noise again.	10
Where 'X' is the value stored in memory		

Pressing S1 and S2 together at power-up will load the default values to EEPROM.

(012016-1)

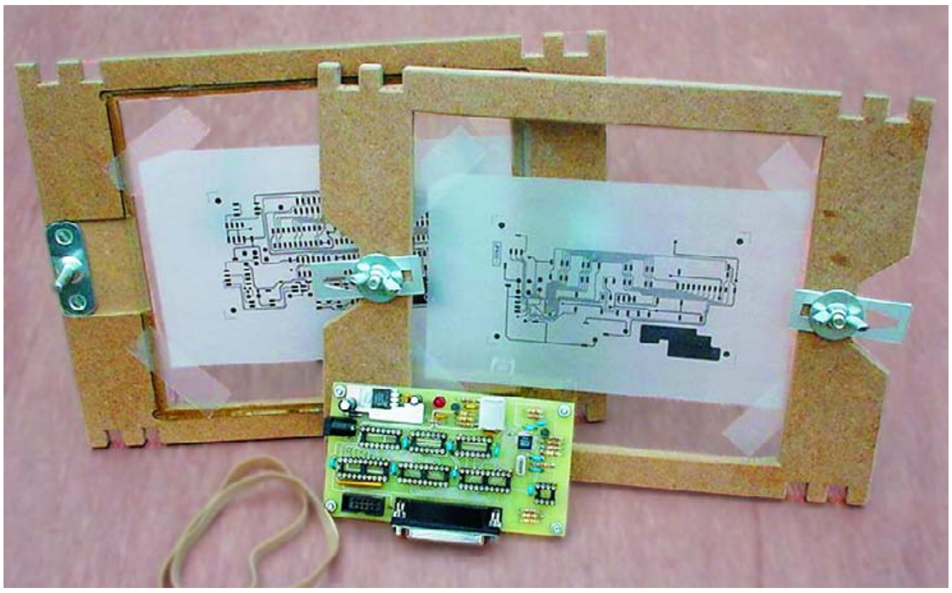
Double-sided PCBs

etch your own — trouble-free

By J. van de Kamer

jkamer@planet.nl

This light-box frame can be easily built by anybody and lets you accurately align films on double-sided PCBs.



Making your own PCBs is fairly straightforward when you make use of the readily available products on the market. When the board is single sided, its exposure is easy. But for double-sided boards the artwork films have to be aligned precisely on both sides of the board in order to make sure that the holes drilled will be neatly centred in the copper pads on both sides.

Most light-boxes are provided with alignment crosses to aid lining up the films. In practise they were not found to be accurate enough.

The light-box frame

The accuracy can be improved with the help of this easily built light-box frame, which allows both films to be lined up precisely.

The complete frame consists of two frames, each with a pane of glass onto which the film can be stuck (**Figure 1**). When both frames are on top of each other, the position of the top one can be adjusted in order to align the films. The position of the top frame is then fixed using two bolts and both frames can then be taken apart for the PCB to be put between them.

After that the PCB can be exposed, either both sides simultaneously or one side at a time, turning the frame over halfway through. The dimensions of the frame have been chosen to give a snug fit in the UV exposure box from Conrad Electronics (order number 53 06 89-33) that has its existing window removed. You can of course change the dimensions for use with a different type of exposure box.

The parts

The frames are made from two pieces of 8 mm thick MDF. This material can be easily worked on without splintering. Cut out both frames using a jigsaw. The openings for the glass should not be cut until the grooves have been milled out. The dark grey areas in **Figure 1** have to be milled to a depth of 3 mm using a router with a straight cutting bit. The diameter isn't critical; 6 mm seems to do the job well. If you can't get hold of a router you could also use a sharp chisel and knife.

After milling the grey areas the openings can be cut which will later hold the glass panes. The holes for the various bolts can also be drilled now. All corners and edges should be filed at an angle to give a neat finish.

The next step is the mounting of the glass panes. This can be ordinary window glass, which can usually be cut to size free of charge at a hardware store or DIY store. The thickness of 4 mm is very important, since the glass has to stick out a little above the MDF frame. This way the films will be touching during the alignment process. If the glass were below the surface of the MDF frame there would be a gap between the films, which makes accurate alignment impossible. After the glass has been cleaned with white spirit or benzene, it can be glued onto the frame with a small amount of two-part adhesive.

Alignment

The most important part of the light-box frame is the alignment mechanism. This should not have any play, be adjustable over a certain range and remember its position when both frames are taken apart. This sounds more difficult than it is in reality.

To help with the alignment we can use a stay-peg from a window stay. There are various types of these available. What we need is a straight pin with a diameter between 6 and 8 mm, which has been welded to a base plate. The pin should have a rounded top, which makes it easier for the top frame to slide on.

The second part of the alignment mechanism has to be homemade out of a piece of aluminium. **Figure 2** shows all the relevant dimensions; the hole on the left fits over the stay-peg. This could have been a round hole, but then this should have had the exact size of the pin to avoid any play. This is practically impossible to make, hence the choice of a V-shape. The angled sides will take up any play in the X and Y directions.

The slot in the right-hand side allows the slider shown in **Figure 2** to move freely round the bolt in the top frame. Two of these sliders are required. The holes can be easily cut out of the aluminium with a fretsaw. Again you should de-burr all sides with a file.

Construction

All parts are now ready to be put together.

Mount the two stay-pegs onto the bottom frame, preferably with countersunk M4 bolts (about 10 mm long) and dome nuts. For the top frame you should use two M5 lock bolts with a length of 20 mm; an accurate tap with a hammer should be enough to fix them in their holes. If you don't want to risk breaking the glass you can use a small flat file to turn the round hole into a square.

A slider is placed over each of the bolts followed by a large washer and a wing nut, allowing this to be easily loosened and tightened. **Figure 3** shows the alignment mechanism of the prototype in detail. The light-box frame is now ready for use.

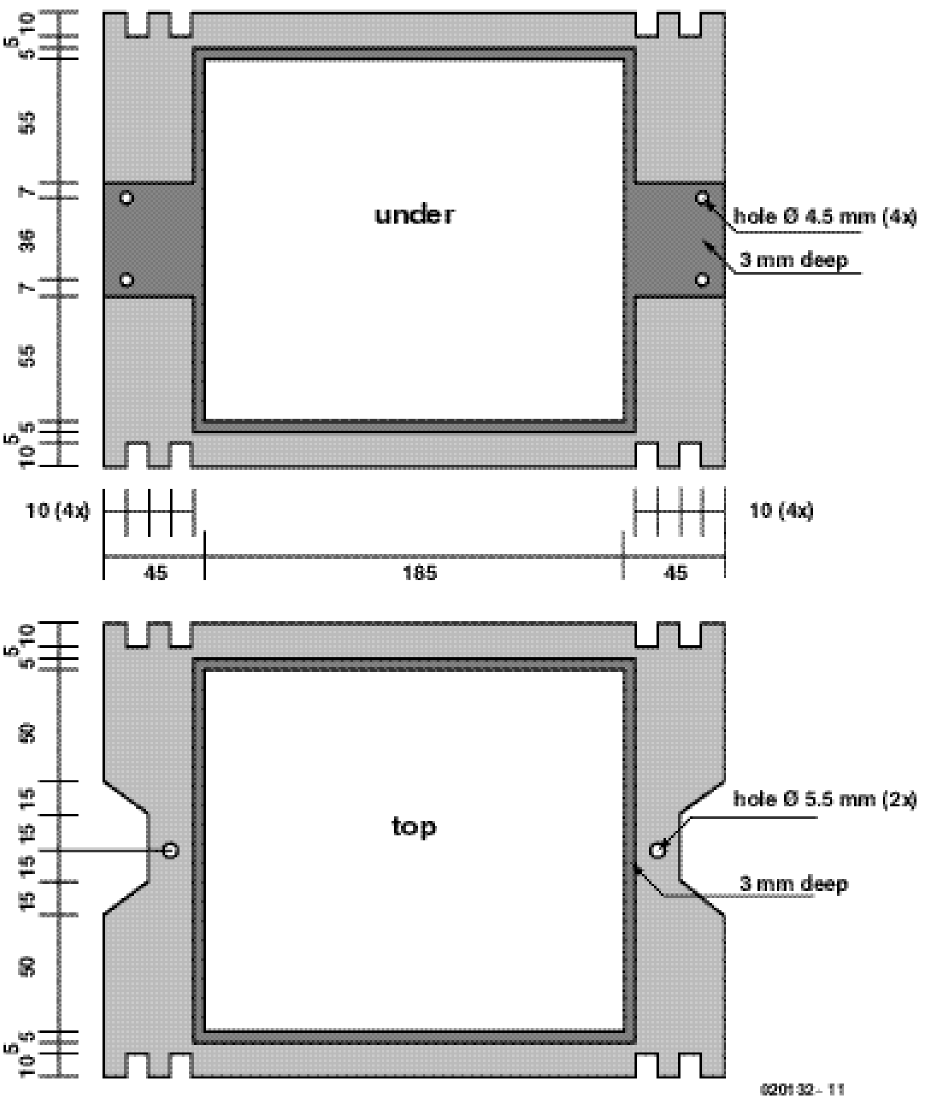


Figure 1. Dimensions of both frames.

The films

The artwork film for the component side can be made in the same way as that for the solder side. The film should however be printed in a mirror image. Apart from that, it is recommended to remove any unused pads. Especially with ICs it is difficult to see which pins are used once the socket is mounted on the board. When the unused pads are removed, only the remaining pads need to be soldered. Some PCB design software supports this feature. Layo1 for example has the facility to export the design as a

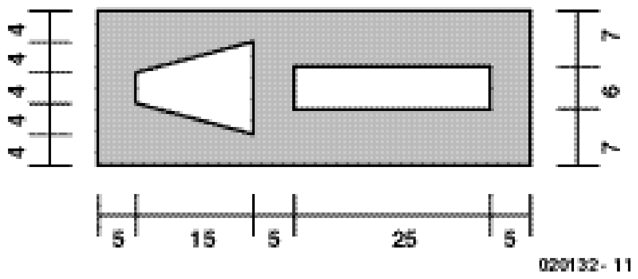


Figure 2. The slider of the alignment mechanism.

layer after which the unused pads can be removed manually.

Using the light-box frame

After both films have been printed they can be cut to size using a sharp knife. Leave enough of a margin round the actual board dimensions. The corners of the film are stuck to the glass using sticky tape. Something that should be avoided is for a corner of the PCB to lie on top of the sticky tape. In this case there will be a gap between the film and the light sensitive layer, causing the tracks to be over-exposed in that area.

With both films stuck to the glass panes the wing nuts are loosened such that the aluminium sliders can move freely. Put the frames on top of each other with the V-shaped holes of the sliders going over the pins. Now it's just the case of moving the top frame until the films are aligned. Once the correct position has been found the sliders should be moved tightly against the pins. The wing nuts are then tightened and the alignment is complete.

The protective films on the PCB should be removed in a dark room. The top frame can now be removed from the bottom frame, taking great care that the sliders don't move! Put the PCB onto the bottom film (avoiding the sticky tape) and replace the top frame again; this should automatically stay aligned with the bottom frame.

The films are pressed tightly to the board by using two thick elastic bands to force the frames together. That is the purpose of the fork shaped cutouts in the corners of the frames.

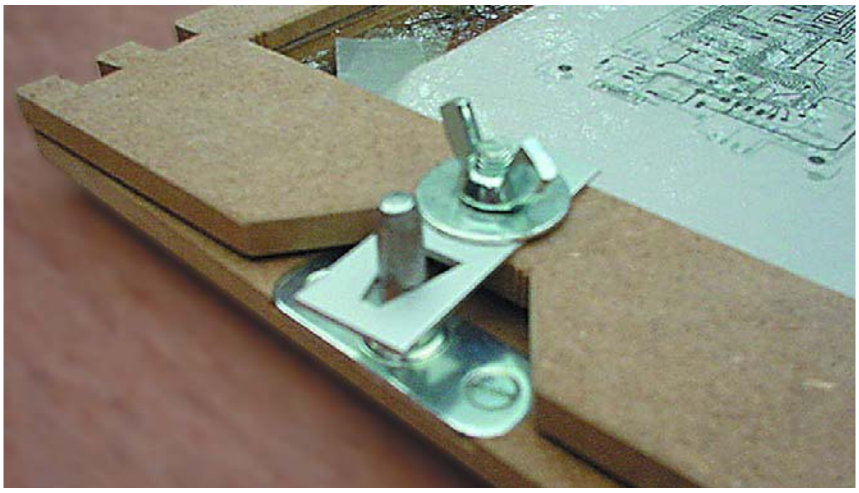


Figure 3. Close-up of the stay-peg/slider/wing nut.

And finally

After the exposing, developing, etching and drilling we end up with a PCB that has its drilled holes neatly in the centre of the pads on both sides. Through-plated holes can be made as usual, by soldering components on both the top and bottom side of the board at specific places.

(020132)

Fuel Cell Developments

further outlook: colder

By Reg Miles

Although fuel cells are still largely the stuff of the future, some recent developments look set to bring that rather closer and give a good idea of the way research is going.

Motorola researchers have further integrated the fuel cell processes. NEC researchers are making use of the highly fashionable carbon nanotubes. While those at Pennsylvania University have managed to make one run on conventional diesel fuel — thus obviating the need for 'reforming' fuels to hydrogen.

PEM cells

The fuelcells used by the Motorola and NEC researchers are of the proton exchange or polymer electrolyte membrane (PEM) type. A porous carbon anode containing a platinum catalyst splits the fuel into hydrogen protons (ions) and electrons. The electrons pass through an external electrical circuit, while the protons pass through the humidified membrane. The protons go to a cathode of the same material as the anode, and there recombine with the returning electrons and with oxygen from the air to form water. This is known collectively as the membrane electrode assembly (MEA).

Researchers at Motorola Labs have now demonstrated a prototype of an integrated, miniature direct methanol fuel cell (DMFC) system; and have also built several of the key components required for a miniature, reformed methanol to hydrogen fuel cell (RHFC).

Previous DMFC systems have used discrete components to process and deliver methanol to the fuel cell, to determine the methanol concentration within the fuel cell, and to separate the carbon dioxide gener-

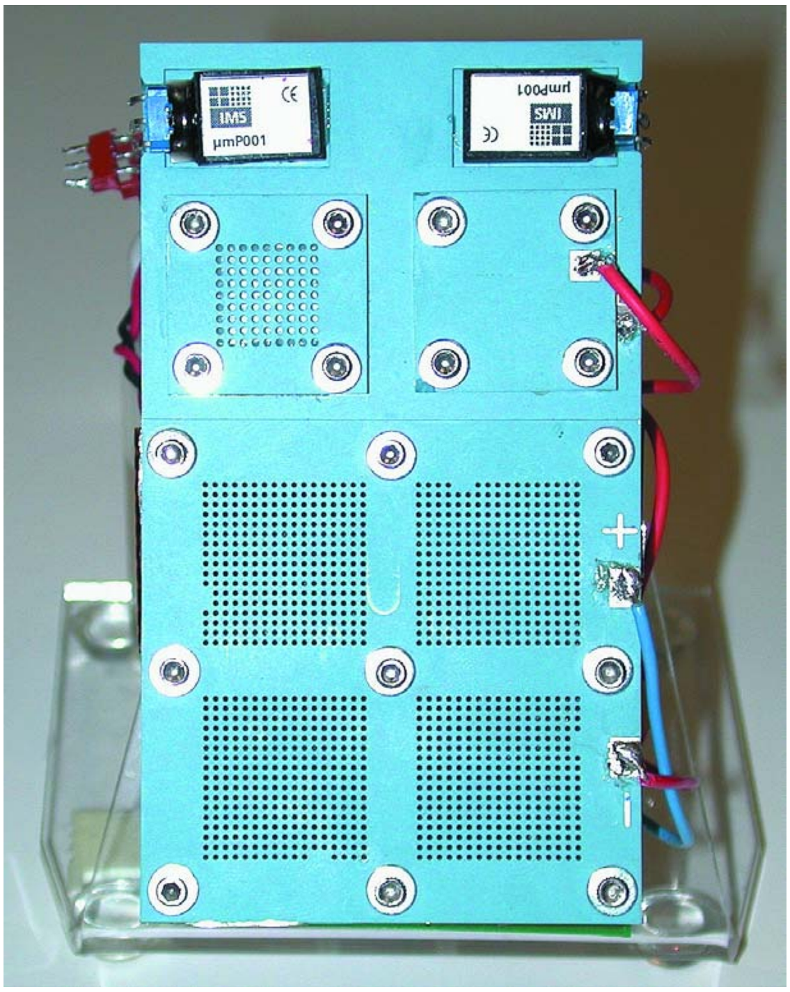


Figure 1. Motorola's experimental reformed methanol to hydrogen fuel cell (RHFC).

ated within the fuel cell from the liquid fuel. Last year, researchers announced a multi-layer ceramic technology for processing and delivering the fuel and air to the MEA. They have now integrated many of those other components, including a methanol concentration sensor and liquid-gas separation for CO_2 release, directly in the ceramic device. Miniature pumps and control and conversion electronics are also built into the device. The experimental assembly (see **Figure 1**) measures about 50×100×10mm (without electronics or fuel); and produces over 100 mW net power continuously.

Work on the RHFC is also dedicated to integrating all the parts. Previous systems have used discrete metal components to vapourise the methanol fuel, reform the methanol to hydrogen, and clean up the output of the reformer. Using the multi-layer ceramic technology, researchers have demonstrated an integrated vapouriser and miniature methanol steam reformer, and a separate chemical heater — three of the key components. The reformer assembly, measuring 38×13×1mm, integrates both the fuel vapouriser and methanol steam reformer. The chemical heater, with the same dimensions, converts a percentage of the methanol fuel into heat to drive the reformer reaction. The plan is to produce an integrated device giving 1 watt or more.

Nanotubes and nanohorns

The NEC development was achieved in a joint effort with the Japan Science and Technology Corporation and the Institute of Research and Innovation. In this, the porous carbon electrodes are replaced by ones made from carbon nanotubes, or, more accurately, carbon nanohorns — a variation in shape. Both were discovered by Sumio Iijima, an NEC Research Fellow. The main characteristic of the nanohorns is that when many of them group together an aggregate of about 100 nm is created. This not only gives an extremely large surface area, but also makes it easy for the gas and liquid to permeate through it.

Nanohorns are easily prepared to high purity, so it is expected to become a low cost raw material. In addition, because a nanohorn is produced by the laser ablation method, if a platinum catalyst is also simultaneously evaporated, a platinum particle will naturally adhere to the surface of a nanohorn, thus obviating the conventional and costly wet process, and compensating for the costly platinum — at least to some extent.

Diesel-powered?

Turning now to Pennsylvania University, Dr. Raymond J. Gorte, professor of chemical engineering, and colleague Dr. John M. Vohs, professor and chair of chemical engineering, developed a butane powered fuel cell — the first to run on anything other than hydrogen. Now they have developed the first to run directly on a readily available liquid fuel. It is a solid oxide fuel cell (SOFC). This differs from the preceding type in having a solid oxide electrolyte (through which oxygen ions migrate — giving a greater choice of fuels), and ceramic electrodes — normally, with nickel as the catalyst. However, the hydrocarbon fuel makes a nickel-based SOFC coke up, so this cell has a copper cermet anode with a ceria catalyst to directly oxidise the fuel.

According to Dr. Gorte: “In our earlier work, we were unable to feed liquid diesel to the fuel cell because we did not have a means for vapourising fuels that have a low vapour pressure at room temperature. This paper demonstrated that we could feed these liquids to a fuel cell using a method analogous to a fuel injector in an internal combustion engine and still get stable operation of the fuel cell.”

The experimental fuel cell operates in a furnace set at 700 degrees Celsius. A commercial, self-contained fuel cell would ideally generate that heat itself using the fuel placed in it. Such a temperature obviously precludes its use for notebook computers, mobile phones, etc. (unlike the Motorola and NEC devices which operate at much lower temperatures); but it still leaves a wide range of applications, such as automotive and portable power generators. Although the laboratory device is tiny — approximately 3 mm square in area, as Dr. Gorte said: “There are no intrinsic reasons for us to work with such small cells, but it is easier for us and the materials questions remain unchanged.”

(020123-1)

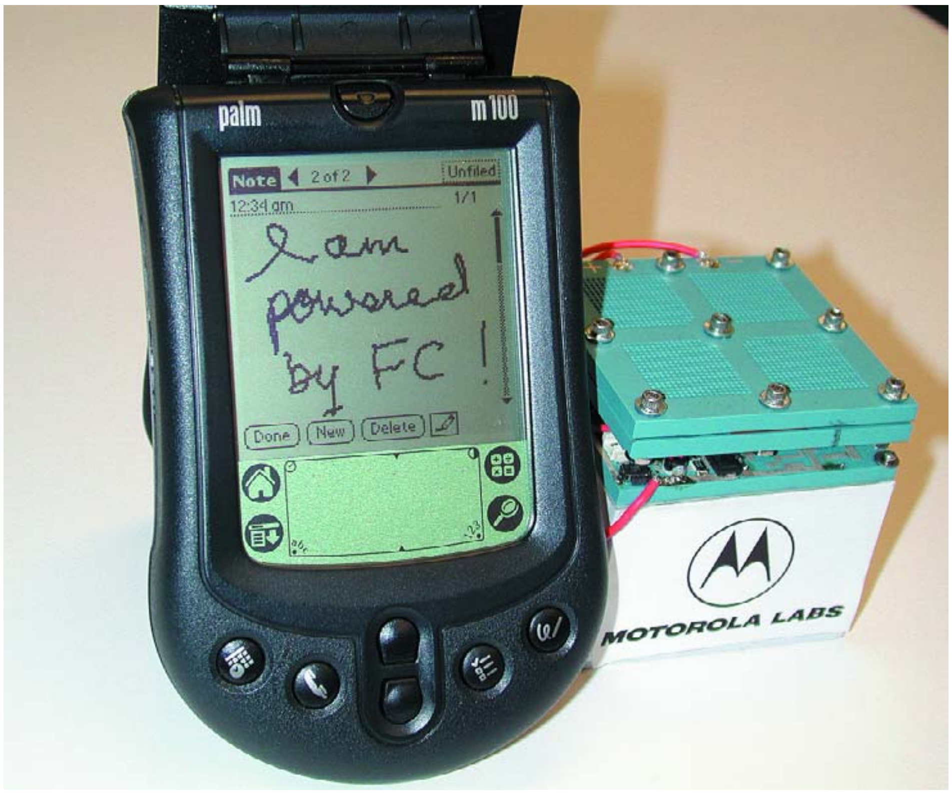


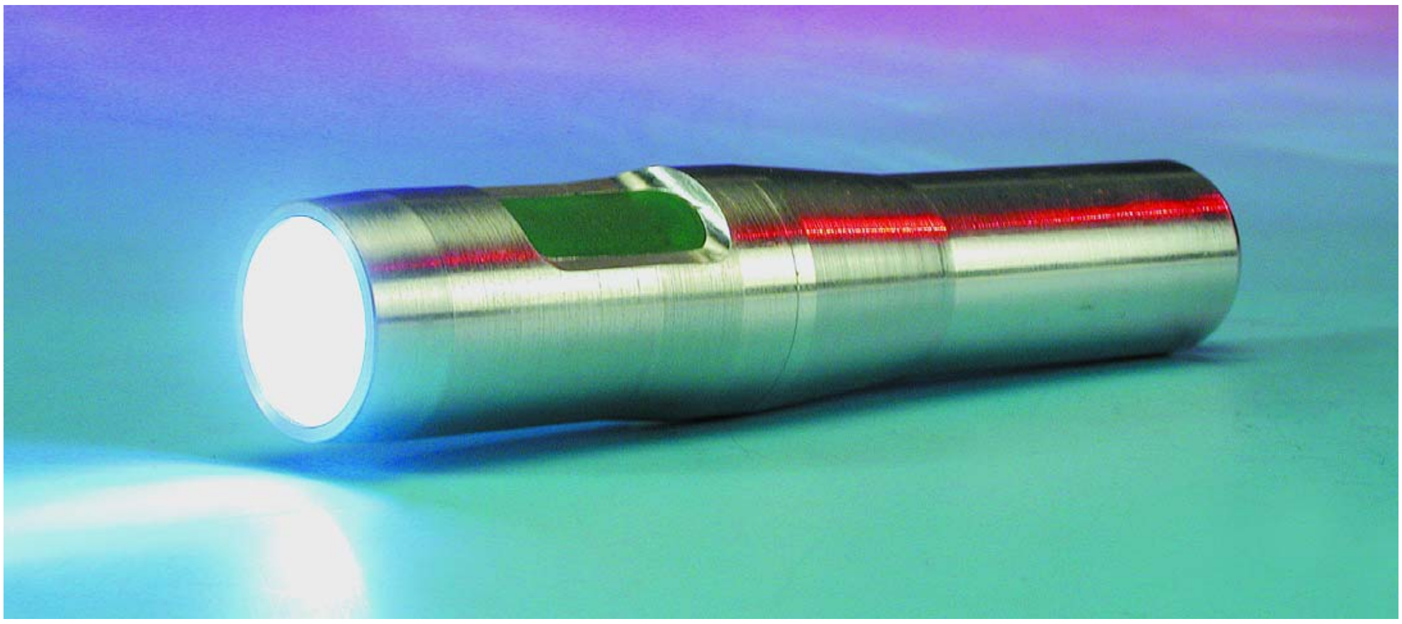
Figure 2. Demonstrating that today’s low temperature fuel cells, while not exactly compact, have the potential to power small electronic devices like a PDA.

LED Torch

flashlight goes high-tech

Design by H. Reisinger

You would be forgiven for thinking that a torch is probably the ultimate minimalist circuit: all you need is a battery, switch and bulb and you're in business but add a microcontroller and you can build in some interesting features that make the humble torch rather more useful…



If you made a wish-list for the ideal torch it would probably start with the need for it to be small enough to fit comfortably in your pocket or rucksack but at the same time be powerful enough to provide enough light at a couple of metres distance to work by for many hours. The environmentally aware would also vote for a rechargeable power source that would hold its charge even if the torch had been lying in a drawer, unused for a couple of years. An indicator showing the amount of charge left in the battery would also be nice together with an automatic cut-off to guard against accidentally draining the battery. It would also be useful if the light level could be efficiently regulated to allow continuous operation for several nights and when the battery gets really low (to 1% of its capacity) still provide a useable emergency light level for a few hours.

To fulfil these criteria we have chosen to use white LEDs instead of a halogen lamp. Contrary to popular belief, the efficiency of a white light LED is not much better than a low power tungsten bulb and in fact worse than higher wattage (> 3 W) bulbs. The LED however offers a significant advantage over filament lamps; they are can be dimmed down to 1/1000th of their maximum power without any loss of efficiency. A tungsten filament, by contrast will glow less brightly and energy will be lost in producing heat but little light so that at half power its efficiency drops to zero. A further advantage is the improved life expectancy of an LED. You can expect it to last more than 1000 times longer than a tungsten lamp so spare bulbs are unnecessary. The LED package also incorporates some optics to direct the light beam so that the torch head

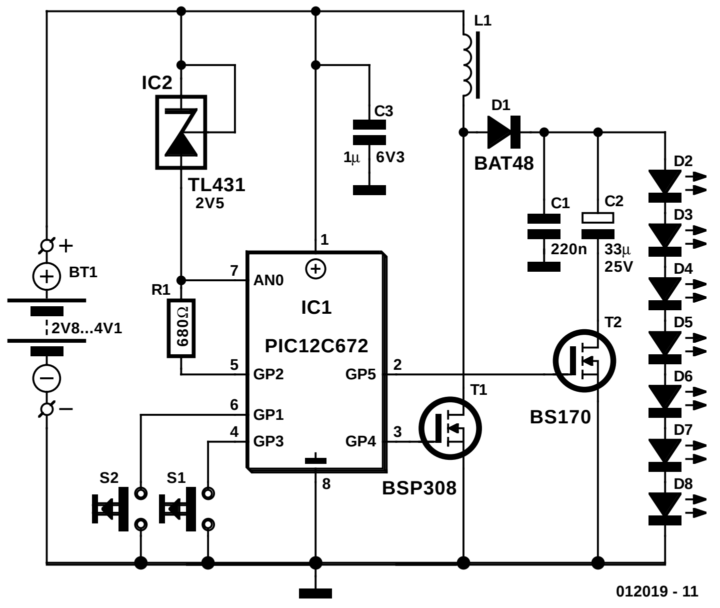


Figure 1. The voltage converter uses a microcontroller.

can be made smaller than a conventional bulb and reflector arrangement.

For the power source a lithium-ion rechargeable battery has been chosen. These cells are more expensive than NiCds but offer a number of advantages: A NiCd cell of the same capacity will weigh three times as much as a Li-Ion cell. Li-Ion also offers a much lower self-discharge of less than 10 % per year compared to 20% per month (!) for NiCds. They also do not suffer from the so-called 'memory effect'.

The circuit

The voltage of a Li-Ion cell varies from 4.1 V fully charged to 2.8 V when empty. The forward conduction voltage of a single white light LED is 3.6 V so it is not possible to limit the current to the LEDs with a simple resistor as you would normally for a red indicator LED. The circuit diagram in **Figure 1** uses a step-up voltage regulator formed by L1, MOSFET T1, diode D1 and smoothing capacitor C1 to supply the LED chain with a controlled power source. The efficiency of the circuit is approximately 94%.

Unusually for a torch, the step-up regulator is controlled by a PIC12C672 microcontroller (IC1). This device has an internal 4 MHz RC oscillator and an 8-bit A/D converter. Analogue input AN0 measures the reference voltage produced by IC2. To conserve energy IC2 is switched on (by pulling GP2 low) only during the measurement time. Capacitor C3 buffers the battery voltage (V_{BATT}) and the A/D converter measures this voltage at the VDD pin. The microcontroller adjusts power to the LEDs by altering the mark-space ratio of the output on pin GP4. The frequency of the signal on this output is a maximum of 30 KHz, less at lower power settings. In the very low power mode the microcontroller would use more power than the LEDs so in this case the microcontroller operates for most of the time in sleep-mode and is woken up every 18 ms to switch the power. In the low power setting capacitor C2 is used to store energy and smooth out the 18 ms flicker on the LEDs. In all other power settings transistor T2 is turned off to disconnect C2. C2 would otherwise prevent a clean switch-off of the light when the torch is set to flash mode.

Two push button switches on the torch allow the selection of six possible commands.

Functions

An advantage of using a microcontroller in this design is that the torch features are defined in software so if you need to change some aspect of the control it is relatively simple to do so (assuming you have the correct programming tools available). Looking a little more closely at the main features of the torch we have:

- The torch brightness can be adjusted in six steps; each step increases power by about three.
- At its lowest setting the light output is sufficient to illuminate close work, ideal for reading comics under the bedcovers! Astronomers will also find this setting useful for reading star maps without seriously impairing their night vision. This setting will use just 0.5 mA and means that one charge will last 120 days (one year if the torch is used for 8 hours per night).
- A built-in time out for each power level is included (see **Table 2**). This feature ensures that only 3% of the battery charge is lost if the torch is accidentally switched on. The torch will give a warning one-minute before it switches off by modulating the light level.
- When the battery voltage falls below 3.3 V the controller will automatically reduce power to the lamp ensuring that the battery has sufficient charge for at least 15 minutes illumination. When the battery voltage falls to 2.8 V the torch switches off to ensure that the battery is not damaged by too deep a discharge.
- After a recovery period the Li-Ion battery regains about 0.5% of its capacity and this will be sufficient for 1 hours operation at the 20 mW setting or 3 hours at 8 mW.
- The torch can be set to act as a warning flashing light with adjustable on/off ratio to conserve power.
- Standby mode uses the least energy. The light flashes at a low setting and can be useful for example for finding the torch in a dark tent. Before the lamp goes into standby mode it will flash to indicate the remaining energy left in the battery. Each flash represents 10% of the cell's capacity. When the cell has less than 10% of its energy left it will indicate after a short delay how much energy remains in steps of 1%.

Table 1 Shows how the torch is controlled using push buttons S1 and S2. **Table 2** gives the parameters of the torch at different light level settings. The column showing duration

Table 1. Function

Key pressed	Function
S1 briefly	Brightness level +
S2 briefly	Brightness level –
S1 and S2 briefly	Off
S1 longer	Flash mode +
S2 longer	Flash mode –
S1 and S2 longer	Battery check, then Standby, Reset Continuous mode

(per charge) for a fully charged battery ignores the effects of cell self-discharge.

Power control

The main function of the step-up regulator is to provide a regulated LED supply voltage despite the falling battery voltage. Taking a closer look at the regulator circuit we find that the energy (E) stored in the inductor (L) by the current (I) can be expressed in the formula:

$$E = \frac{1}{2}\cdot L \cdot I^2 \quad ..1)$$

The current I will increase linearly from 0 and is a function of the time that transistor T1 is switched on (T_{ON}).

$$I = \frac{V_{BATT}\cdot T_{ON}}{L} \quad ..2)$$

Substituting into equation (1) we get the average power (P) for the period:

$$E = \frac{1}{2}\cdot\frac{V_{BATT}^{\;2}\cdot T_{ON}^{\;2}}{L}$$

$$P = \frac{1}{2}\cdot\frac{(V_{BATT}^{\;2}\cdot T_{ON}^{\;2})}{L\cdot\left(T_{ON}+T_{OFF}\right)} \quad ..3)$$

It can be seen that the power to the LEDs is a factor of the battery voltage so that if no regulation were used the power would vary by a factor of 2 as the battery voltage drops from 4.1 to 2.8 V. The average current is P/V_{BATT}. The on to off ratio of the transistor (T_{ON} und T_{OFF}) is altered by the microcontroller to keep P constant.

In order to regulate the power it is necessary for the microcontroller to know the battery cell potential V_{BATT}. To reduce the component count the supply voltage is not directly measured, instead the battery voltage is used as the reference input (= full scale) and the input at AN0 is measured. The TL431AC Bandgap-reference produces a constant 2.495 V. For the 8-bit ADC the measurement result stored in the internal register AD_{RES} is given by:

$$AD_{RES} = \frac{V_{BATT} - 2.495V}{V_{BATT}}\cdot 255 \quad ..4)$$

Rearranging:

$$V_{BATT} = \frac{2.495V}{1 - AD_{RES}/255} \quad ..5)$$

V_{BATT} is measured every 100 ms to regulate the output power.

The microcontroller uses this battery voltage measurement to point to a stored look-up table for one of 16 possible values to control the T_{ON} and T_{OFF} times at output pin GP4.

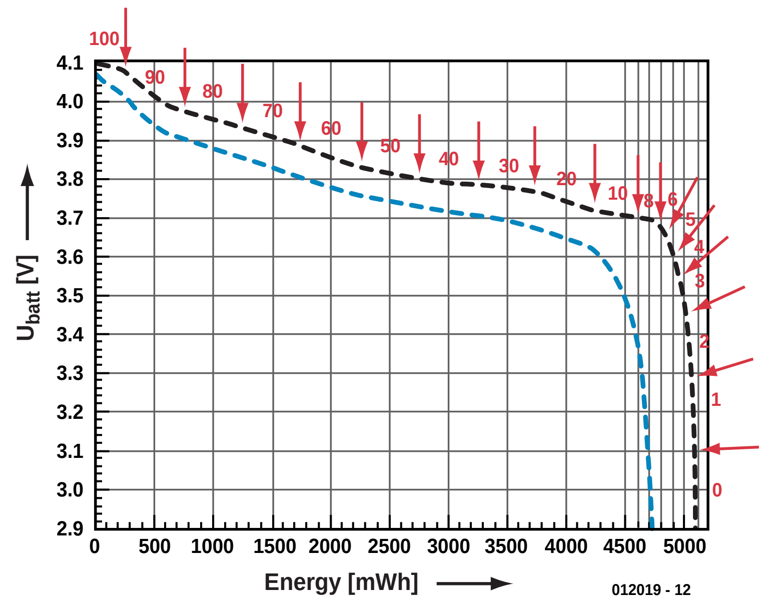


Figure 2. Typical Li-Ion cell discharge characteristics.

Table 2. Light level settings

Setting	Power	Current drain V_{batt} = 3.6 V	Duration (1.3 Ah)	Timeout period
0	0.5 W	140 mA	10 hr	20 min
1	0.2 W	55 mA	24 hr	40 min
2	65 mW	18 mA	3 days	2 h
3	20 mW	6 mA	9 days	5 h
4	8 mW	2.4 mA	22 days	10 h
5	4 mW	1.0 mA	55 days	10 h
Standby	–	30 µA	5 years	24 h
Off	–	2 µA	70 years	–

The value of 'back e.m.f.' (electromotive force) generated across inductor L1 is approximately $6\times V_{BATT}$ when the transistor T1 is switched off (T_{OFF}). The transistor off time will be $T_{OFF} > 1/6 \times T_{ON}$ this ensures that the current I has time to fall to zero.

Internal resistances of the Schottky diode, inductor and battery produce losses in the circuit and reduce the circuit efficiency. The efficiency at maximum power setting reaches about 94%, but even LEDs from the

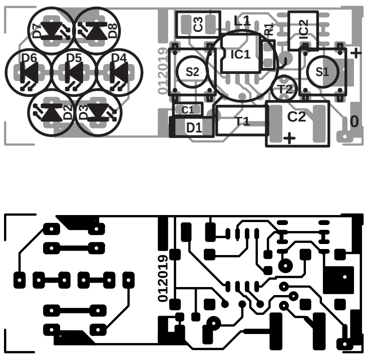


Figure 3. The two-part PCB uses SMD components.

COMPONENTS LIST

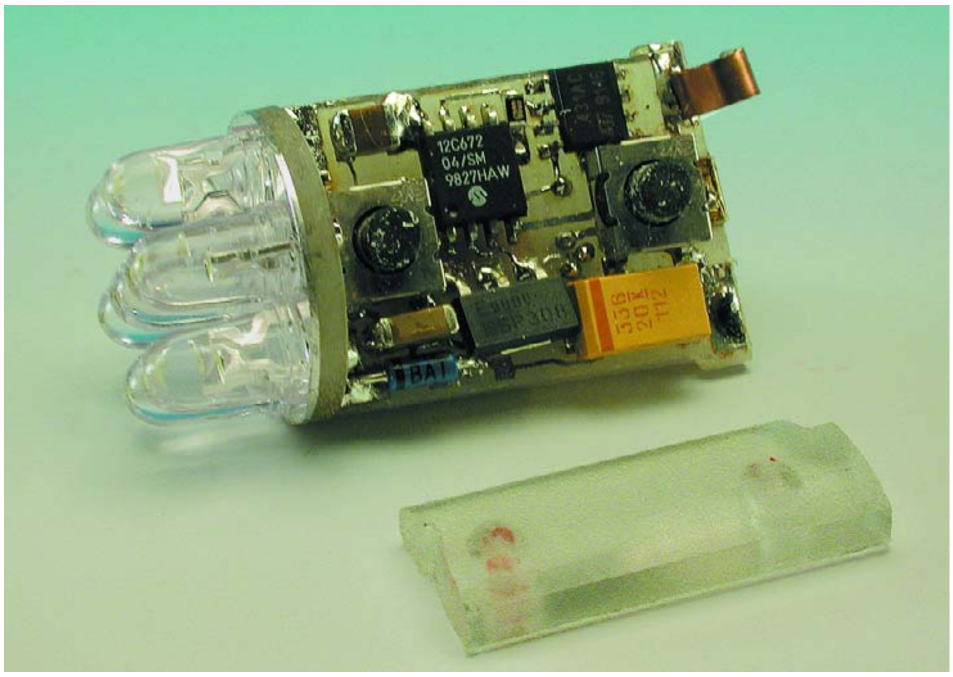
Resistor:
R1 = 680 Ω, SMD size 0805

Capacitors:
C1 = 220nF ceramic, SMD size 0805
C2 = 33µF 20V, SMD size 2220
C3 = 1µF 6.3V, SMD size 1210

Inductor:
L1 = 233µH (100kHz), ESR = 0.27 Ω
Pot core P9.0x5.0/N26 (A_l = 250) with 30.5 turns of 0.22mm dia. ECW

Semiconductors:
D1= BAT48 or 1N4148
D2-D8 = LED, 5mm, white, 6400mCd (Nichia, NSPW500BS) (www.nichia.co.jp/lamp-e.htm), available from Conrad Electronics
T1=BSP308, BSP319 (R_{ON}=50mΩ)
T2=BS170, BSS138 (R_{ON}=5Ω)
IC1=PIC12C672 04/SM, programmed, order code **012019-41**
IC2=LM9140-2.5 or TL431

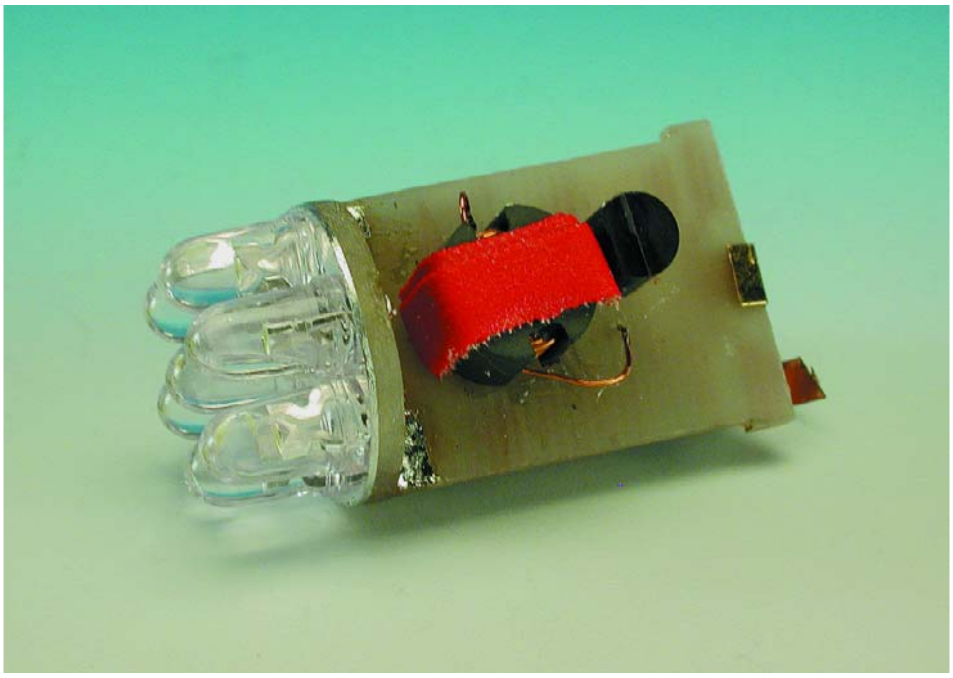
Miscellaneous:
S1,S2 = SMD switch, 1 contact (Mentor 1254.1007 or 1301.9314 or Omron B3FS-1052 from Farnell)
Battery = Sanyo UR18650 Li-Ion, 1350 mAh (18mm, length = 65mm, weight 40g) or Sanyo UR18500 Li-Ion, 1100 mAh (18mm, l = 50mm)



same manufacturer can have poorly matched efficiency values. At the present time the most efficient white light LEDs are made by Nichia. These LEDs specified are only intended for use by OEMs but it is possible to find some outlets that stock them.

Charge indication

An important design feature of the torch is that the user should never be left unexpectedly in the dark. It is therefore necessary for the controller to be aware of the charge remaining in the cell. This charge is dependant on the battery voltage, the discharge current, previous discharge history and cell temperature. In the torch we simply determine the charge remaining by measuring the cell voltage some time (1,000 s) after the light has been switched off. The discharge characteristics of a Li-Ion cell are shown in **Figure 2.** A feature of these cells is that when the load is removed the cell voltage rises slowly (1,000 s approximately) from the lower curve to the upper curve. The arrows and numbers on the



graph show the percentage of charge remaining in the battery and correspond to the figure indicated by the microcontroller. The measurement resolution is approximately ±30 mV, which corresponds to a worse case measurement error of about 4%.

This method of charge measurement is not particularly accurate but is acceptable for our application here. The cell self-discharge is typically less than 5% per year may so the torch will still be ready for use even if it has not been used for some time.

A time-out feature ensures that if the torch is accidentally left on it will not drain the battery. The time-out will occur (independent of power) after approximately 3% of the maximum cell capacity of 5 Wh (watt-hour) is used. When the charge in the cell falls below 10% the microcontroller will reduce power to the LEDs to the next lowest level. The cell voltage will rise slightly. When the remaining charge falls below a threshold the LEDs will be turned off. In this case it is still possible to switch the torch back on at a low setting for a short period.

With just 1% charge left in the cell the torch will function for 5 hrs at brightness level 4 or more than 12 hrs at level 5 so you should never be left completely in the dark.

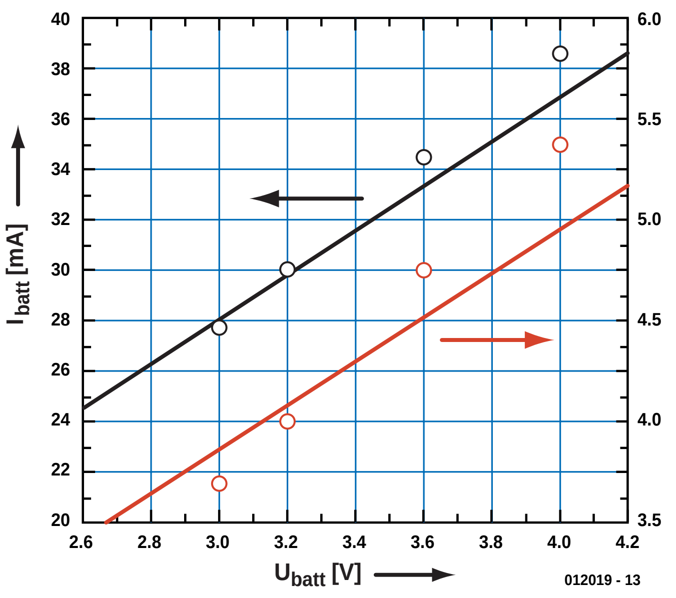


Figure 4. Current consumption of the LED torch.

The battery and coil

The battery specified in the parts list has a capacity of 1.3 Ah (approximately 5 Wh). This is a little bit generous for the torch but the cell size is widely used for Laptops so it is relatively economical. Mobile phone batteries are more difficult to obtain and are therefore more expensive.

More information on Li-Ion cells can be found on the Web at

www.sanyo.com/industrial/batteries/industrial_liion.html

or

www.panasonic.com/industrial/battery/oem/chem/lithion/index.html

Coil L1 has an inductance of 220 to 250 µH with I_{MAX} = 0.5 A and can be purchased off the shelf or alternatively hand-wound using the EPCOS core specified together with 0.2 to 0.22 mm diameter enamelled copper

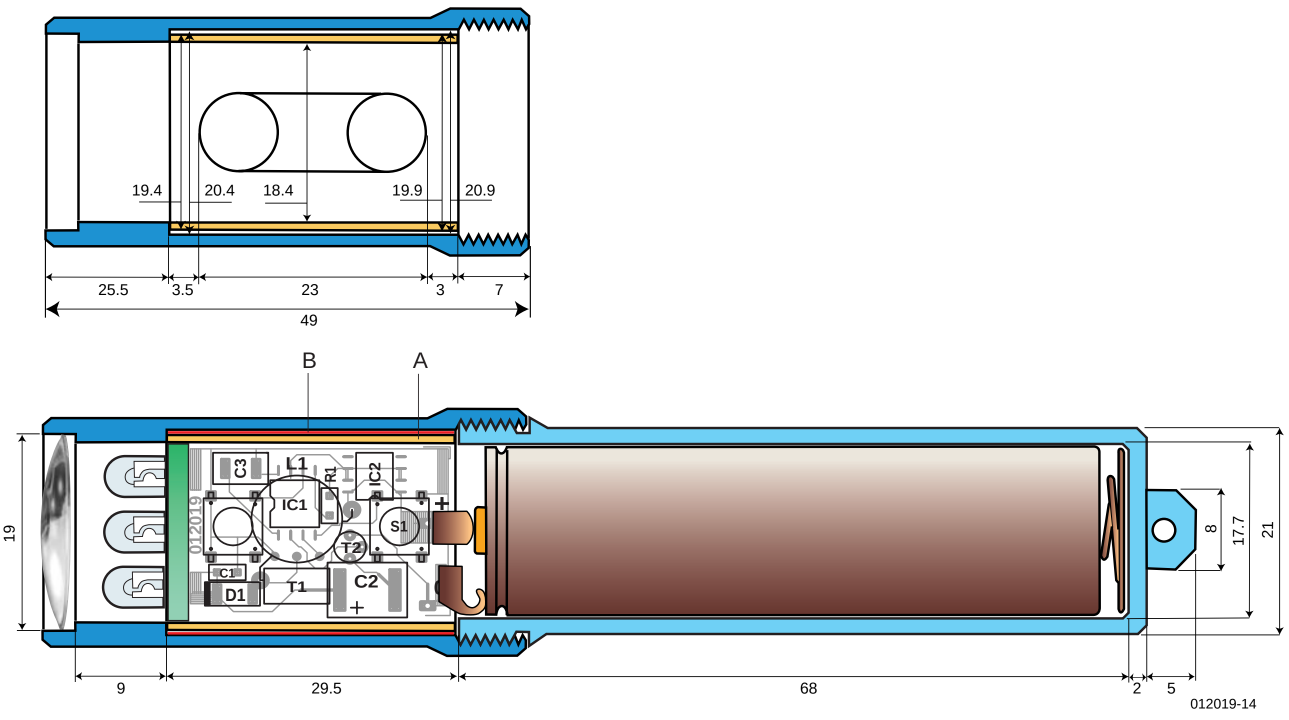
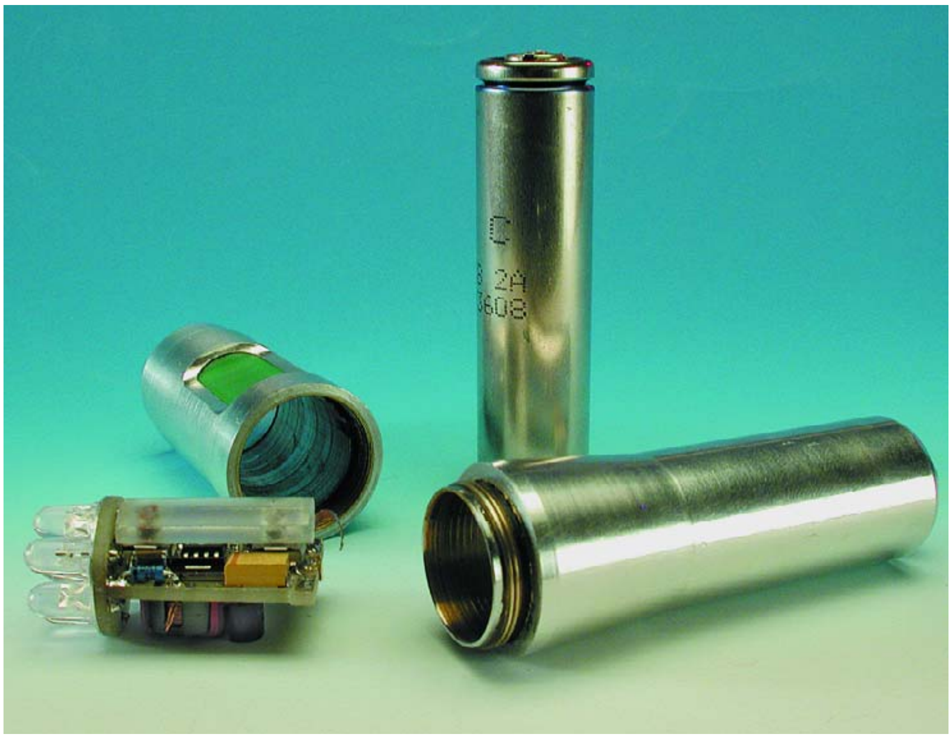


Figure 5. Mechanical layout for a splashproof case.

wire. Each winding layer should be separated by two turns of 0.1 mm thick tape. This will reduce the capacitance of the finished coil and also provide a smooth surface for the next layer of windings. The EPCOS core has an A_l value of 250 so to produce an inductance of 233 mH requires 30.5 turns:

$$L = 30.5^2 \text{ x } 250 \text{ nH} = 233 \text{ mH}$$

To be on the safe side add an extra turn (L will increase by 6%). After completing the coil it can be measured directly on an inductance meter or tested in-circuit as described under the next heading. If you are planning to fit a magnet to the torch casing to enable it to clamp on to metallic surfaces make sure that the magnet is sited at least 2 cm away from the coil otherwise it will reduce its inductance by about 10%.



Let there be light

The PCB layout is shown in **Figure 3.** Divide the PCB into its two parts and solder all the surface-mount components and LEDs onto the boards. The coil is the only 'conventionally' packaged component and should be soldered to the component side of the PCB. The two PCBs are then soldered together at right angles so that the pads serve both to secure the PCBs and carry power to the LEDs.

To test the circuit it is best to use a variable Power Supply Unit (PSU) with a current indicator (alternatively a low impedance ammeter in series with the supply lead will suffice). It is important to note that the microcontroller technical data sheet specifies a minimum ramp-up time for the supply voltage. If the supply voltage rises too slowly the internal reset does not function correctly and this can cause the FETs to conduct too heavily. This is not a problem when the circuit is battery powered but from an external PSU make sure you adjust the supply voltage (above 2.8 V) before switching the output to the circuit. Before powering-up for the first time set the PSU current limit to 0.4 A and add a 10 Ω resistor in series with the supply.

At power-up (providing no buttons are pressed) the PIC will begin the first of two test routines. These routines are invoked only when the battery is fitted. They will check that the clock frequency is 4 MHz (corresponding to a machine cycle of 1 µs), test the inductor and check the A/D converter.

During the first test routine the lamp will flash on and off with a period of 200 ms and a 50:50 duty cycle. If the ON time of the light is much greater than the OFF time (more than about 20%) then this indicates that the value of L1 is too high and it will be necessary to remove one or two turns from the winding. The lamp will go out after 1,000 periods (200 s) providing none of the buttons are pressed first. During ON times the lamp brightness is set by a T_{ON} time of 15 µs and a (T_{ON}+T_{OFF}) time of 100 µs. Current consumption should correspond to the lower line of the graph in **Figure 4**.

When everything is in order, briefly press one of the pushbuttons this will turn off the lamp and start the second test routine. In this routine the lamp is lit continuously at increasing power levels. Doubling the T_{ON} time increases the power four times. The current consumption of the circuit should correspond with the upper line in Figure 4.

The torch will switch off after 50 s if none of the keys are pressed. This time-out is necessary because you may not always be aware that a reset to the microcontroller has occurred. The torch may, for example undergo a mechanical shock sufficient to momentarily disconnect the battery and generate a reset. Pressing and releasing either push-button will return the torch to normal operation at brightness level 2.

Making a good case

Li-Ion cells are not the same size as standard AA cells so it is not a simple matter just to cannibalise a standard torch case.

Machine drawings are available for this project (**Figure 5**), detailing the construction of a suitable casing for the torch. Those of you who do not have a lathe or milling machine lying around at home may wish to take the drawings along to a local machine shop for a quotation. The drawings are produced in Micrographx format and can be downloaded from this month's Free Downloads at the *Elektor Electronics* website. Note that the PCB slides into tapered slots in the head of the torch, this ensures that the PCB and the splashproof pushbutton covers (boots) are securely held in place.

Solder two copper battery contacts to the PCB and fit a suitable spring in the base of the torch to ensure that the Li-Ion cell makes a good connection with the contacts.

(012019-1)

LCD Connections & More

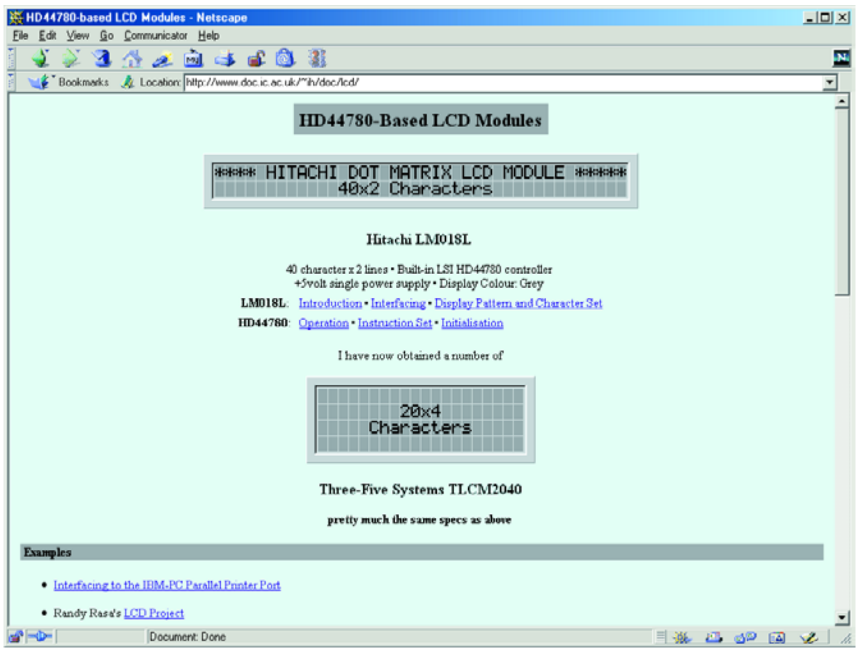
Technical details and applications

By Harry Baggen

Small alphanumeric LCD displays are applied in many electronic circuits, particularly when microcontroller-driven. These displays are easy to control and not expensive. However, you will always need to know their connection details and other salient features. Fortunately the Internet abounds with technical information on LCDs, as well as interesting application briefs.

When designing a microcontroller system, a terminal or any other intelligent circuit that needs to 'show' us something, an alphanumeric LCD is often the easiest way to obtain a readout device. A good choice, really, because LCDs are affordable, they offer a clear readout and are relatively easy to drive in software and hardware. Most commercially available LCDs of the smaller variety use the same controller IC, whether the display has 1, 2 or 4 lines of 8, 16, 20, 24, 32 or 40 characters. In all of these displays you will typically find the Hitachi HD44780 controller or a second source equivalent. The datasheet of this 'evergreen' chip is available in the form of a pdf (Acrobat Reader) document from the **Hitachi** website [1]. However, because of the widespread use of these displays in electronic equipment, there are many other sites on the web that deal with this chip and way it is applied.

A fine page with plenty of information is Ian Harris' **HD44780-Based LCD Modules** [2]. On it Ian provides comprehensive information on 2- and 4-line displays, including their interfacing, display patterns and character sets. You will also find many useful links leading to application examples and other sources of information. Another address for information on this controller, connection details of displays based on the HD44780, as well as on LCD backlighting is that hosted by **Lampex** [3], a manufacturer of all kinds of LCDs.


HD44780-based LCD Modules - Netscape
File Edit View Go Communicator Help
Bookmarks
Location: http://www.doc.ic.ac.uk/~ih/doc/lcd/
Document: Done

HD44780-Based LCD Modules

**** HITACHI DOT MATRIX LCD MODULE *****
40x2 Characters

Hitachi LM018L

40 character x 2 lines • Built-in LSI HD44780 controller
+5volt single power supply • Display Colour: Grey

LM018L: Introduction • Interfacing • Display Pattern and Character Set
HD44780: Operation • Instruction Set • Initialisation

I have now obtained a number of

20x4
Characters

Three-Five Systems TLCM2040

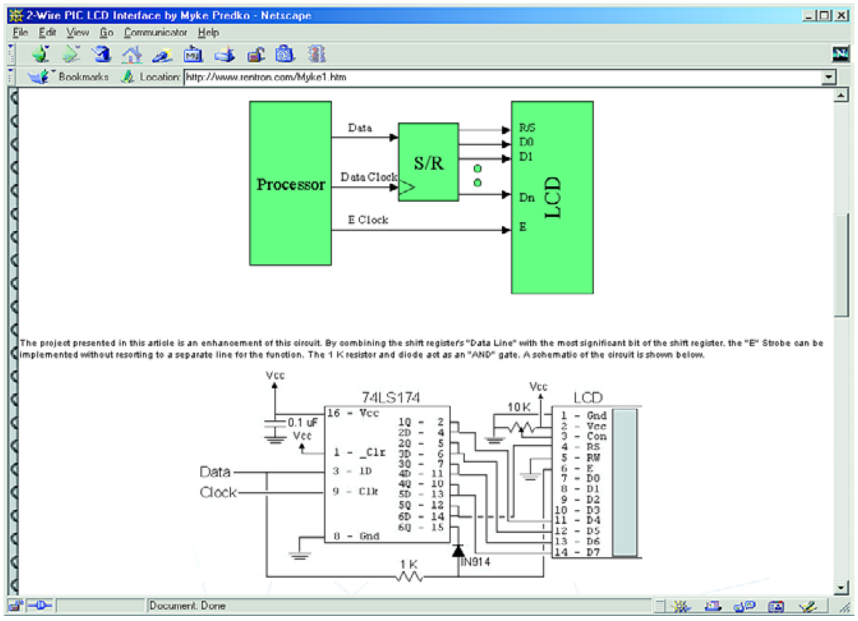
pretty much the same specs as above

Examples

- Interfacing to the IBM-PC Parallel Printer Port
- Randy Rasa's LCD Project

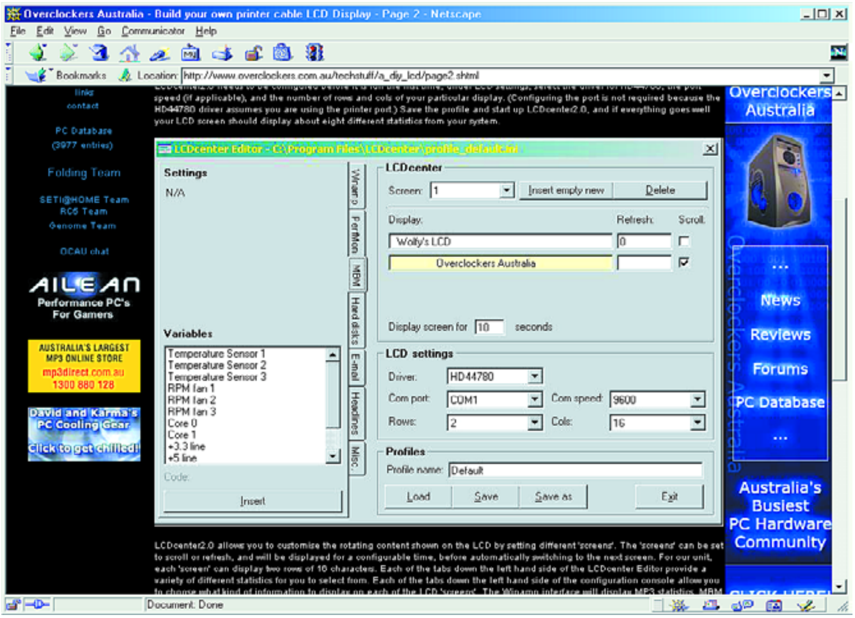
If you need information on displays that do not fall in the 'standard' category, you should really visit the LCD page at **EIO** (Electronics Information Online) [4]. This site holds datasheets and specifications of LCD products from most manufacturers in the field.

Driving a 44780-based LCD is not difficult once you've absorbed some of the relevant manuals that may be picked up from the Internet. At **Peer's LCD pages**, published by Peter Ouwehand, an extensive trea-



2-Wire PIC LCD Interface by Myke Predko - Netscape
http://www.rentron.com/Myke1.htm
Processor
Data
Data Clock
E Clock
S/R
R/S
D0
D1
Dn
E
LCD
Vcc
74LS174
0.1 uF
10K
1K
1N914
Data
Clock

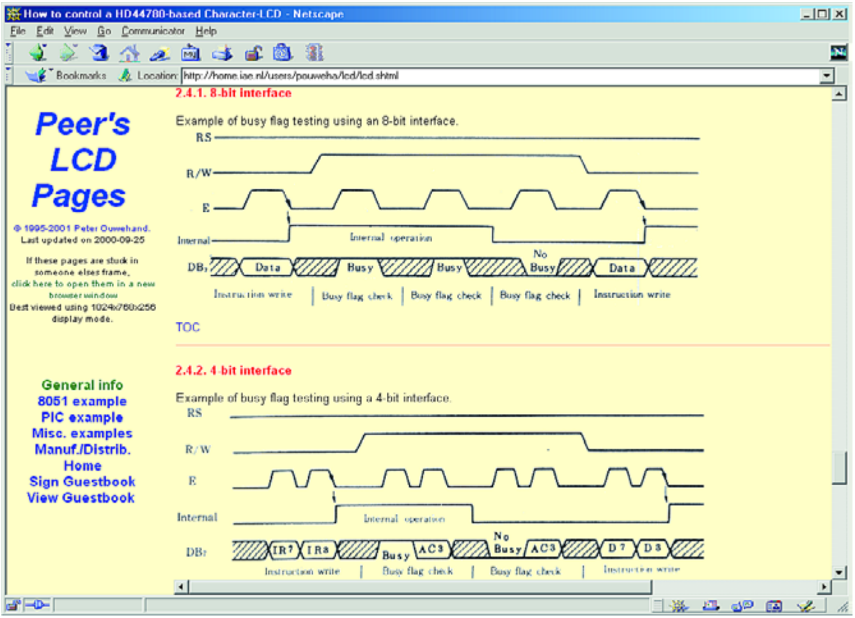
The project presented in this article is an enhancement of this circuit. By combining the shift register's "Data Line" with the most significant bit of the shift register, the "E" Strobe can be implemented without resorting to a separate line for the function. The 1 K resistor and diode act as an "AND" gate. A schematic of the circuit is shown below.



Overclockers Australia - Build your own printer cable LCD Display - Page 2 - Netscape
http://www.overclockers.com.au/techstuff/a_diy_lcd/page2.shtml
LCDcenter Editor - C:\Program Files\LCDcenter\profile_default.ini
Settings
N/A
Variables
LCDcenter
Screen: 1
Insert empty new
Delete
Display:
Refresh:
Scroll
Wolfy's LCD
Overclockers Australia
Display screen for 10 seconds
LCD settings
Driver: HD44780
Com port: COM1
Com speed: 9600
Rows: 2
Cols: 16
Profiles
Profile name: Default
Load
Save
Save as
Exit
Insert
News
Reviews
Forums
PC Database
Australia's Busiest PC Hardware Community

speed (if applicable), and the number of rows and cols of your particular display. (Configuring the port is not required because the HD44780 driver assumes you are using the printer port.) Save the profile and start up LCDcenter2.0, and if everything goes well your LCD screen should display about eight different statistics from your system.

LCDcenter2.0 allows you to customise the rotating content shown on the LCD by setting different 'screens'. The 'screens' can be set to scroll or refresh, and will be displayed for a configurable time, before automatically switching to the next screen. For our unit, each 'screen' can display two rows of 16 characters. Each of the tabs down the left hand side of the LCDcenter Editor provide a variety of different statistics for you to select from. Each of the tabs down the left hand side of the configuration console allow you



tise may be found on driving 44780-based LCDs [5]. The material includes application examples for the popular 8051 and PIC microcontrollers.

Another interesting application is **Build your own "2-wire LCD Interface using the PIC16C84" microcontroller** produced by Myke Predko [6]. As indicated by the title, we're dealing with a kind of serial drive of an LCD by means of a PIC. Also worth mentioning is Myke's **LCD Interfacing Reference Page** [7].

If you want to hook up an LCD to a computer, consider the parallel (printer or LPT) port for that purpose. An example (including source code) of how it may be done is available from Craig Peacock's **Beyond Logic** website [8].

Those of you who can not boast to much experience in this area may want to start off at an extensive description containing lots of photographs of an LCD connected to a printer cable, on the **Build your own printer cable LCD display** [9] webpage provided by Overclockers Australia. This project allows all LCD settings to be carried out on the PC using a clearly laid out Windows program.

(025067-1)

Internet addresses

[1] Hitachi:
http://semiconductor.hitachi.com/hd44780.pdf

[2] HD44780-Based LCD Modules:
www.doc.ic.ac.uk/~ih/doc/lcd/

[3] Lampex:
www.lampex.com/prod.htm

[4] EIO LCD-pagina:
www.eio.com/datashet.htm

[5] How to control a HD44780-based Character-LCD:
http://home.iae.nl/users/pouweha/lcd/lcd.shtml

[6] Build your own "2-Wire LCD Interface" using the PIC16C84 microcontroller:
www.rentron.com/Myke1.htm

[7] LCD Interfacing Reference Page:
www.myke.com/lcd.htm

[8] Beyond Logic:
www.beyondlogic.org/parlcd/parlcd.htm

[9] Build your own printer cable LCD Display:
http://www.overclockers.com.au/techstuff/a_diy_lcd/