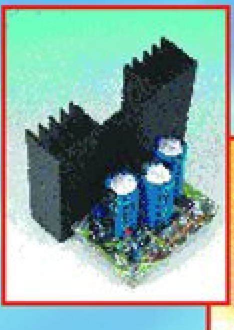
THE ELECTRONICS & COMPUTER MAGAZINE

FEBRUARY 2000
£2·95

ELEKTOR ELECTRONICS

PC TOPICS:

- running text display
- PC keyboard codes unravelled
- PC interface for Nintendo joystick



compact 50-watt in-car amplifier

2-metre band converter

IR remote control

for Sony MiniDisc players

parameter box for MIDI software

Awkward keyboard commands and uncomfortable mouse control can spoil even the most attractive PC game. The ideal solution would be simple operation, as offered by the Nintendo 64, combined with the processing power of a PC. This article describes how to connect a Nintendo 64 controller to the PC game port.

design by K. Schuster

PC interface for Nintendo joystick

using the Nintendo 64 to run PC games

This circuit allows a Nintendo 64 controller to be connected to the PC game port (or a sound card), without requiring any additional drivers to be installed. The Nintendo 64 controller is a widely-used unit that combines high quality with a low price. With this approach, you can run PC games with the comfortable Nintendo 64 controller instead of using the PC keyboard and mouse.

What the Nintendo 64 controller offers...

In addition to a few membrane switches, the controller contains a precise analogue electro-optical joystick module that works like a mouse. On demand, the controller unit reports the status of the switches and the position of the joystick. Bidirectional communication takes place over a single line that has a High level in the rest state. This line is used both to send commands to the controller and to receive the requested data from the controller. A command byte must be sent before data can be received from the controller. If the line is free, as indicated by a persistent High level, the command byte can be transferred. The controller responds to the command $01 with the status information for all pushbuttons and the position of the analogue joystick. The transmission time for each bit is 4µs in both send and receive modes. A Low bit is indicated by a 3-µs Low phase followed by a 1-µs High phase, while a High bit is indicated by a 1-µs Low phase followed by a 3-µs High phase. In order to delay the response to a command, the last transferred bit of the command can be held Low. If the line is returned High at the end of the command transmission, the response should occur within 2 to 3 microseconds. The response time is not fixed, since the controller and the N64C2PC IC operate asynchronously, each with its own clock.

The first experimental circuit, with a 8051 clocked at 12 MHz (corresponding to a 1 µs cycle time), was obviously too slow to meet the critical timing requirements of the Nintendo 64 controller. Reliable communication was only possible after the microprocessor was replaced by an AT89C2051-24PC with a 24-MHz clock. Regarding the hardware, you can see that two clock sources are shown in **Figure 1**, in addition to the microcontroller and a pair of current-limiting resistors. This is because 24-MHz crystals are normally only available for series-resonant operation. Such ‘overtone’ crystals are not suitable for this application! If you cannot obtain a fundamental-frequency crystal, you can use a self-contained 24-MHz oscillator (see the list of components).

Returning to the communications with the controller, the answer to the command $01 is four bytes of controller status information, transmitted MSB first, as shown in **Table 1**.

...is not what the game port expects

A simple PC game port does not need any active circuitry. The two pushbuttons simply make connections to earth. The PC game port, or a suitable sound card, simply polls the switch levels to see whether they are High or Low.

With the analogue joystick, the situation is a bit more complicated. The joystick contains two potentiometers (X and Y), whose resistances are around 100 kΩ, connected to the supply voltage. Capacitors located on the card are charged via these potentiometers. These capacitors determine the time constants of a pair of monostable multivibrators. The positions of the potentiometers can thus be derived from the lengths of the pulses produced by the monostables. All analogue elements are addressed or polled at the same time. A normal PC game port provides connections for two joysticks, which means that it has four 'digital' and four 'analogue' inputs. Sometimes only one joystick can be connected, but this is very rare.

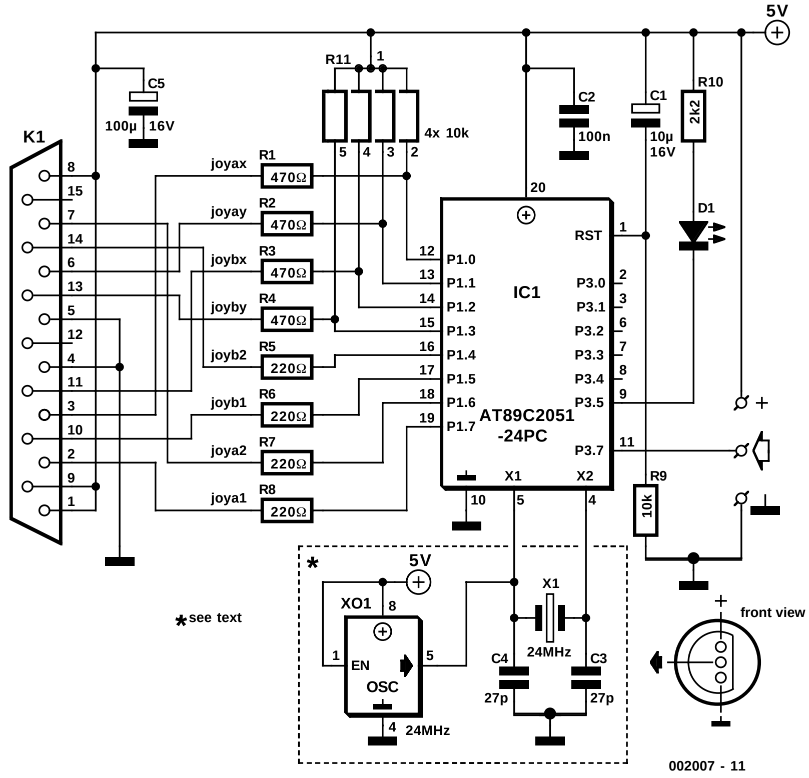


Figure 1. A microcontroller, a pair of resistors and an oscillator are all you need for the adapter circuit.

Two worlds join together

It is not difficult to see that these two worlds do not really fit with each other. Requesting and interpreting the status data from the Nintendo 64 controller should not be difficult, but how can the expectations of the PC game port be satisfied without a lot of complicated circuitry? Handling the pushbuttons is relatively easy; the relevant bits from the Nintendo 64 controller can simply be periodically output on the microcontroller leads. However, what should be done with the digital values for the analogue joystick? Here we can use a trick: the interface microcontroller waits for a short Low level on one of the potentiometer lines, which goes along with the cyclic charging of the capacitors of the PC game port card. Following this, the microcontroller holds all of the potentiometer lines Low, to prevent any further charging of the capacitors, and starts its timer. Each of the potentiometer lines is subsequently allowed to go High at a time that depends on the data received from the Nintendo 64 controller. The corresponding capacitors are charged briefly via the microcontroller outputs, and the associated monostables report what they assume to be the potentiometer positions. If you observe the relevant outputs of the AT89C2051 with an oscilloscope, you will see pulse-width modulated signals with a period of around 840µs and a duty cycle of 50% to 90%, depending on the potentiometer position. When the potentiometer is at the midrange position, the duty cycle is 70%.

Table 1. Nintendo 64 serial status information

Byte 1		
	Bit 7	button A
	Bit 6	button B
	Bit 5	button Z
	Bit 4	start button
	Bit 3	control cross up
	Bit 2	control cross down
	Bit 1	control cross left
	Bit 0	control cross right
Byte 2		
	Bit 7	unknown, always 0
	Bit 6	unknown, always 0
	Bit 5	button L
	Bit 4	button R
	Bit 3	button C up
	Bit 2	button C down
	Bit 1	button C left
	Bit 0	button C right
Byte 3		
	analogue stick x	
Byte 4		
	analogue stick y	

Details — the program

The software, including the source code, is available from *the Elektor Electronics* web site (www.elektor-electronics.co.uk). If you cannot program the microcontroller yourself, you can obtain a ready-programmed device from our Readers Services under order code **006504-1**.

The main loop of the program starts after the stack and the two timers have been initialized, the timers have been started and their interrupts have been enabled. First, the timing for the analogue joystick modules A and B (B is the control cross or C button) are established by the routines prepajoyt and prepbjoyt, respectively. Timers T0 and T1 are responsible for the timing of joystick A, with T0 used for the X axis and T1 for the Y axis. Timer T0 also manages the Timeout Mode, which prevents the program from getting stuck in a polling loop if the Nintendo 64 controller is unexpectedly disconnected or there is an intermittent contact. In such a situation, it would otherwise not be possible to initialize the Nintendo 64 controller once it was reconnected without first manually resetting the microcontroller. The entire program is synchronized with the slowest and least-flexible element, the PC game port. The instruction jnb JPYAX,* waits for the capacitors to be discharged. Once the game port has

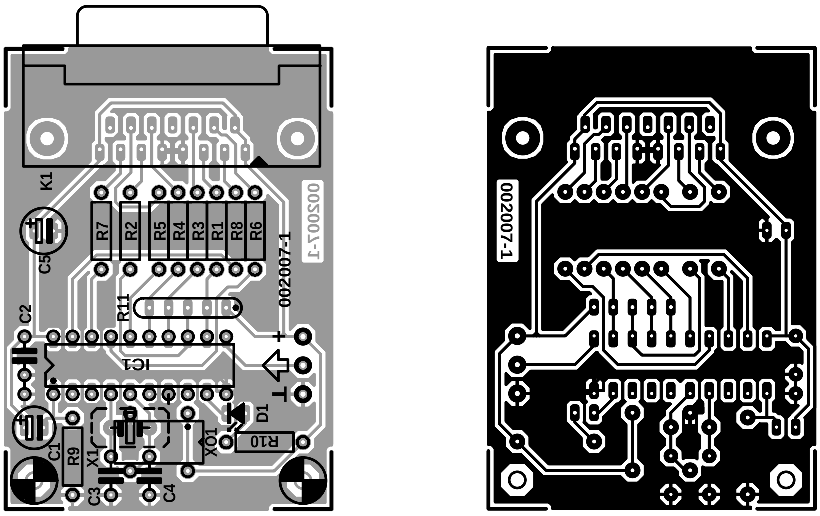


Figure 2. The printed circuit board for the Nintendo-64/PC adapter.

COMPONENTS LIST

Resistors:
R1-R4 = 470Ω
R5-R8 = 220Ω
R9 = 10kΩ
R10 = 2kΩ2
R11 = SIL-resistor array 4x10kΩ

Capacitors:
C1 = 10µF 16V radial
C2 = 100nF
C3,C4 = 27pF (*)
C5 = 100µF 16V radial

Semiconductors:
D1 = LED, low current
IC1 = AT89C2051-24PC (order code **006504-1**)

Miscellaneous:
K1 = 15-way sub-D plug fr board edge mounting
X1 or XO1 = quartz crystal, 24MHz, fundametal resonance or 24MHz-oscillator module (Seiko-Epson SG531P-24MHz) (*)

(*) = see text

done this, the microcontroller sets the four potentiometer lines JOYAX/Y and JOYBX/Y Low and starts timers T0 and T1 for JOYAX/Y, since these are assigned to the analogue joystick. The control cross or C button is assigned to JOYBX/Y. Analogue values are also expected here, so the timing is handled by the routine joybtiming, due to the lack of additional timers in the microcontroller. With the help of a few NOPs and nested loops, the game port receives what it expects here as well, and the JOYBX/Y lines are set high again after appropriate delays. The rate of advance in the Y direction can be set to one of three different levels by simple 'switch-on, switch-off' logic. If the control cross or the C button is used during a game for forward or reverse motion, the L button can be used to switch between 'creeping', 'walking' and 'running'. The duty cycle range is thereby switched from its default range of 58%–78% to either 48%–88% or 40%–97%.

After both software timers have timed out, the program waits until the hardware timers T0 and T1 have completed their jobs and generated interrupts. Once they have timed out, the JOYAX/Y outputs are again set to High. Since the program can easily get hung in the subsequent time-critical portion, the timer T0 interrupt is used as an 'emergency brake' timeout in the routine Inittom. If the Nintendo 64 controller does not respond within a predefined interval, the program is restarted from the beginning. The routine sendbyteA sends the command $01 (Status Information), and the following routine getbytes reads the four status bytes from the Nintendo 64 controller. Bytes 1 through 4 land in registers R4 through R7 for further processing. Before each byte is read, precise bit synchronization is established, following which the Timeout Mode of Timer 0 is again deactivated and the values that have just been read in are interpreted in the routine handlebuttons. This works according to the arrangement shown in **Table 2**.

Repeatedly pressing the L button changes the advance rate of the control cross up/down buttons or C button in three steps.

Once the switch states have been evaluated and their status has been passed on to the PC game port, the loop starts from the beginning with the evaluation of the analogue values that have been read in. The routine calctiming normalizes and scales these values in terms of processor cycles, and the resulting data form the inputs for the next round, which begins with the discharging of the capacitors.

Playing around

In order for the new joystick to be used with the PC under Windows 95/98, it must be made known to the operating system. You should find a joystick or game controller icon under

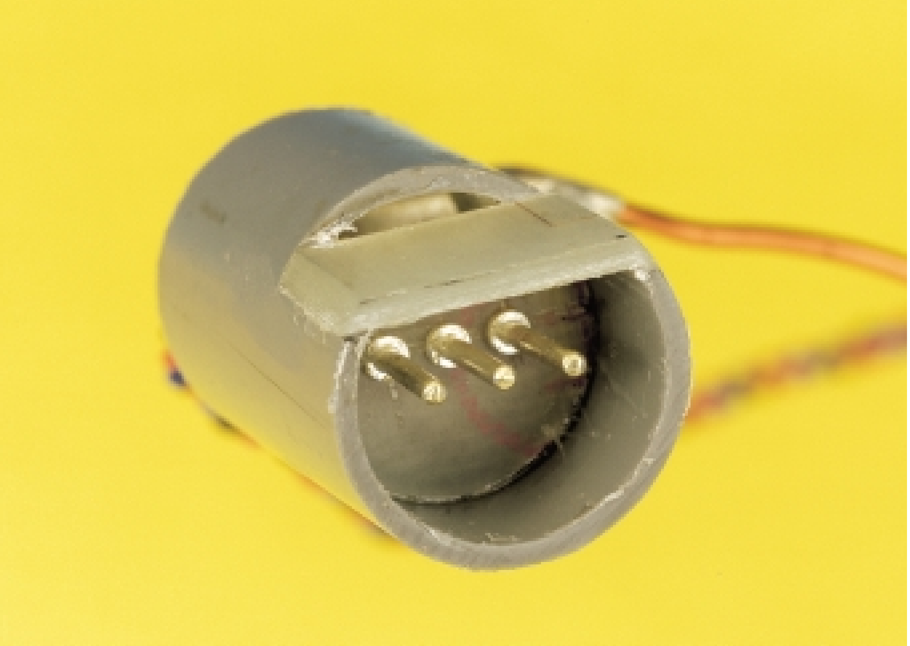


Figure 3. How to prevent an incorrect connection.

Settings/Control Panel. If you do not, the necessary software must first be installed. After this, the best approach is to configure a new joystick with four axes and four pushbuttons. During the subsequent calibration, make sure that the up/down buttons of the control cross, or the C button, have been set to the highest speed using the L button (recognizable by the largest displacement on the screen). The settings can be saved with the name ‘N64’, for example. This name may be needed later to configure certain games. Older (DOS) games only require calibration.
Some games (such as Unreal) offer an extensive range of joystick settings, which you will have to carefully study and try out. In some cases, such as with Half-Life, you will need a small joystick configuration file that contains the configuration data. The game looks for this file in a particular folder when it is started (for example, c:\Sierra\HalfLife\valve). You can usually find tips in the ReadMe files of the games as well. **Table 3** shows two typical configuration files.

Construction hints

Constructing the circuit, using the printed circuit board shown in **Figure 2**, should not present any difficulties. This PCB is unfortunately not available ready-made through our Readers Services. Mount the microcontroller in a good-quality socket. The choice between a quartz crystal and an oscillator module has already been discussed. If an oscillator module is used, omit capacitors C3 and C4 (and of course X1). Difficulties may arise with the (various) controller plugs, since matching sockets are hard to come by. There are three possible solutions: (a) cannibalize an old Nintendo 64 console, (b) cut off the plug and make up an adapter cable with a three-way DIN or Mini-XLR plug (with a mating connector on the end of the cable), or (c) improvise a solution using 1.3-mm diameter solder pins to which short lengths of wire are soldered, which in turn can be soldered to the inputs of the AT89C2051 (see **Figure 3**). To protect against a reverse-polarity connection, you should solder the pins to a piece of prototyping board with a hole spacing of 3.75 mm, and then use an additional part (for example, a piece of 3/4-inch plastic pipe, as shown) to prevent the plug from being connected incorrectly.

(002007)

Table 2. Arrangement of the Nintendo-64/PC game port signals

N64 Controller	PC Gameport	Line
button A	Joy A button 1	JOYAB1
button B	Joy A button 2	JOYAB2
button Z	Joy B button 1	JOYBB1
button Start/R	Joy B button 2	JOYBB2
analogue X-axis	Joy A analogue x	JOYAAX
analogue Y-axis	Joy A analogue y	JOYAAY
K/C left/right	Joy B analogue x	JOYBAX
K/C up/down	Joy B analogue y	JOYBAY

K=control cross, C=buttons

Table 3. Two typical joystick configuration files

```
// name joystick.cfg
// analog turn and look version
//
// x analog turn left/right
// y analog look up/down
// C move left/right
// C move forward/backward
// configure in game: A alternate fire, B duck, Z fire, R/Start jump
//
joyname ”N64”
joyadvanced 1
joyadvaxisx 4
joyadvaxisy 2
joyadvaxisz 1
joyadvaxisr 3
joyadvaxisu 0
joyadvaxisv 0
joyforwardsensitivity -1.0
joysidesensitivity 1.0
joypitchsensitivity -1.0
joyyawsensitivity -1.0
joyforwardthreshold 0.1
joysidethreshold 0.1
joypitchthreshold 0.1
joyyawthreshold 0.1
joyadvancedupdate
```

Alternative version:

```
// name joystick.cfg
// analog turn and move version
//
// x analog turn left/right
// y analog move forward/backward
// C look up/down
// C move left/right
// configure in game: A jump, B alternate fire, Z fire, R/Start duck
//
joyname ”N64”
joyadvanced 1
joyadvaxisx 4
joyadvaxisy 1
joyadvaxisz 2
joyadvaxisr 3
joyadvaxisu 0
joyadvaxisv 0
joyforwardsensitivity -1.0
joysidesensitivity 1.0
joypitchsensitivity 1.0
joyyawsensitivity -1.0
joyforwardthreshold 0.1
joysidethreshold 0.1
joypitchthreshold 0.1
joyyawthreshold 0.1
joyadvancedupdate
```

It's hard to keep track of how many running text displays you run across nowadays. You can find them used as decorations in shop windows, as programmable signboards and as simple eye-catchers. If you buy one ready-made, however, it's fairly expensive, and they are usually too complex for DIY construction. The running text display project in this article combines a simple and inexpensive design with repeatable construction and ease of use.

Design by K. Wohlrabe

running text display

controlled by a COP-8 microcontroller

Technical specifications

Stored text:	508 characters maximum
Displayed text:	6 characters visible using 5 x 7 matrix elements
Transmitter range:	approximately 10 m
Operating voltage:	12 V (transmitter and receiver)

The objective of the running text display project was to develop a simple, inexpensive design that would not be too difficult to build and would be easy to use. We intentionally decided not to make the display as large as possible or to implement a lot of different display modes, since these would require a powerful microcontroller or a single-board computer. The result is a circuit that is controlled by an inexpensive National Semiconductor microcontroller with 4 kB of ROM, and which can be built using readily obtainable components.

Conventional running text displays normally have keyboards that are directly cabled to the display units. These keyboards are usually not laid out the same as standard keyboards, so programming is awkward and time consuming. The keyboard matrix also requires a relatively large printed circuit board with expensive keys, which makes it unsuitable for a DIY project. The basic idea of this project is to use a standard PC keyboard with an infrared data link to the display unit. The running text can then be conveniently programmed, with all the advantages of using a standardised PC keyboard — and this can be done up to 10 metres away from the display.

The transmitter, with its attached keyboard, can also be employed as a general-purpose unit for other (future) projects, in order to simplify the con-

COMPONENTS LIST
(transmitter)

Resistors:
R1 = 10Ω
R2 = 100kΩ
R3 = 470Ω
R4 = 1MΩ

Capacitors:
C1,C2 = 33pF
C3 = 100nF
C4 = 220μF 16V

Semiconductors:
D1 = TSUB8201
T1 = ZTX603
IC1 = COP8782 (order code **996527-1**)
IC2 = 7805
IC3 = ZSM560

Miscellaneous:
K1 = 5-way DIN socket 180° or PS2-socket
X1 = 10MHz quartz crystal

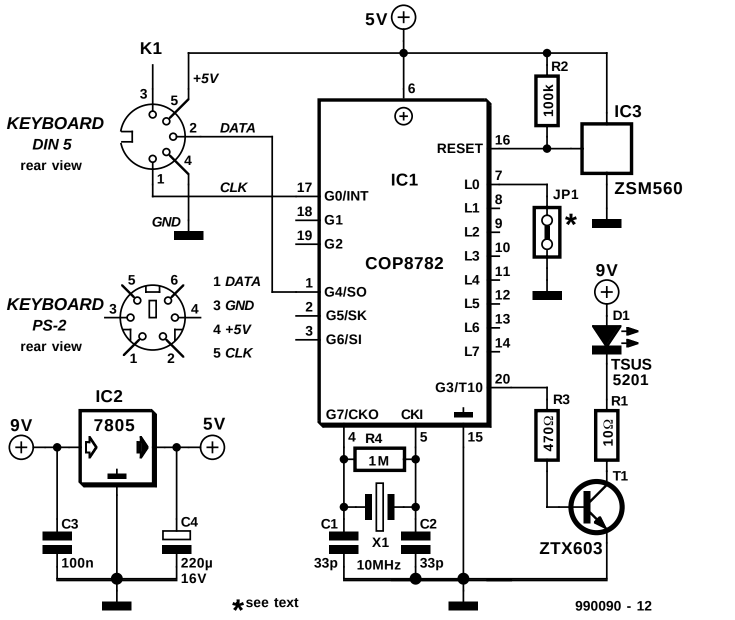


Figure 1. The input data transmitter contains only a COP-8 microcontroller and an infrared transmitter module.

struction and operation of the equipment. Only one port pin of the microcontroller on the receiver side is needed for decoding the information from up to 128 keys.

The transmitter

The microcontroller in the transmitter unit, whose schematic diagram is shown in **Figure 1**, receives the serial digital signals from the PC keyboard and converts them into a protocol that is sent to the display unit via infrared light. The decoding of PC keyboard signals is described in another article, elsewhere in this issue. The microcontroller in the transmitter unit selects scan code set 3 after it has been reset, switches on the Scroll LED of the keyboard as an indication that it is active, suppresses the Break code for the upper-case (shifted) keys and transfers key codes to the display unit. The data transfer employs a modulated 36 kHz carrier, in order to provide noise immu-

The microcontroller

The same type of microcontroller is used in the transmitter and the receiver. The specifications of this National Semiconductor IC make it an outstanding choice for this project:

- 4096 x 8 OTP EPROM
- 128 bytes of RAM
- 1 µs cycle time at 10 MHz
- 16-bit timer with the following operating modes:
- auto reload
- external event counter
- timer with capture function
- 16 I/O leads, of which 14 can individually be programmed as inputs or outputs
- •selectable pin configuration: tri-state, push-pull or pull-up
- Microwire interface
- interrupt sources: external with selectable edge, timer or software

The COP8782 microcontroller now has a successor, with the type designation COP8SAC7. This has improved characteristics, but it is essentially pin-compatible and functionally compatible with the older version. There is a starter kit available, which unfortunately does not allow real-time emulation, but which does allow OTP devices to be programmed. It also provides comprehensive insight into the possibilities of this inexpensive and technically interesting microcontroller family. For somewhat more demanding projects that require the real-time behaviour of the microcontroller to be tested, you have no other choice than to buy an emulator if you do not want your projects to turn into endless trial-and-error sessions.

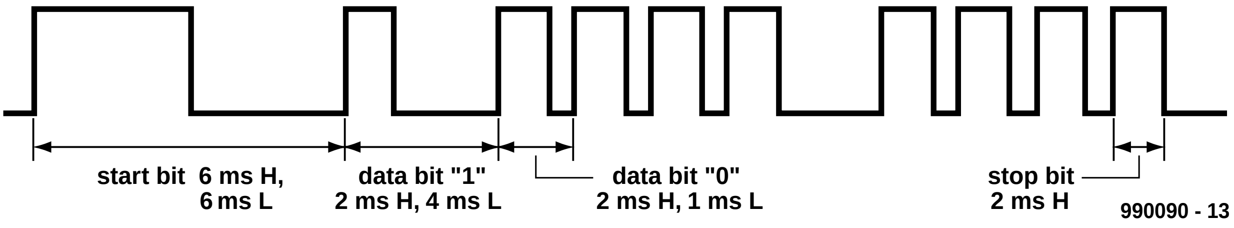


Figure 2. Timing diagram of the transmitter signal that modulates the 36-kHz carrier (this example is for the code 88_H).

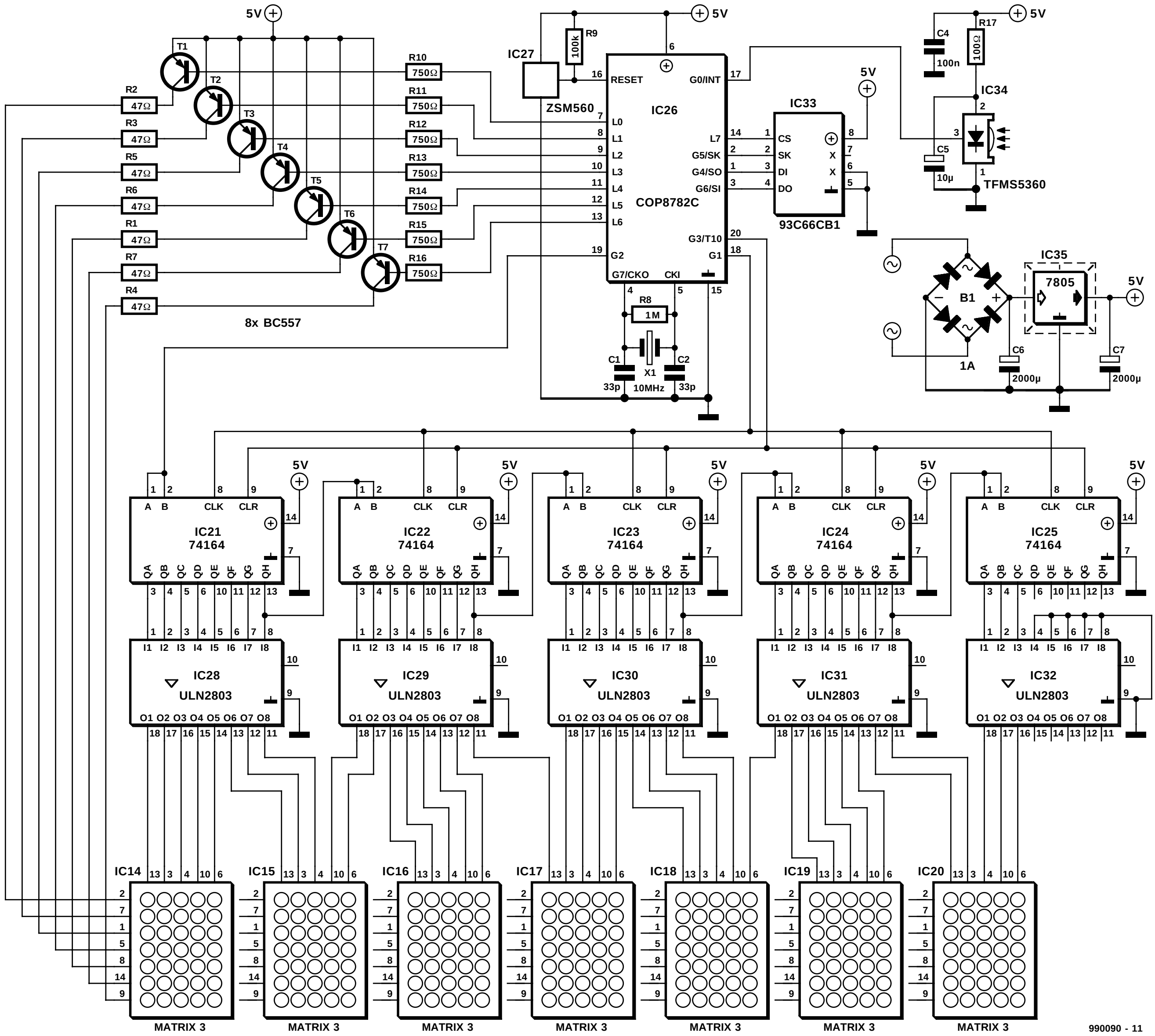


Figure 3. Circuit diagram of the receiver and seven-position LED display matrix.

nity. One start bit, eight data bits, one parity bit and one stop bit are transmitted. The microcontroller is clocked at the relatively high rate of 10 MHz. This enables it to correctly decode the serial data stream from the keyboard, and to generate the 36 kHz carrier frequency for the infrared diode, using only software.

Figure 2 shows the timing diagram of a sample character (with the code 88_H) before modulation. The infrared diode D1 is driven by the Darlington transistor T1. In order to give the transmitter a wide range, the value of the current-limiting resistor R1 is intentionally chosen to be on the low side, and a high-efficiency LED is used. However, in principle any type of infrared LED can be used.

The short data packets, and the resulting short 'on' times, prevent the transistor from becoming overheated. The reset IC (IC3) ensures that the microcontroller always starts up properly. The current consumption of the transmitter, including the connected keyboard, is around 110mA. Since it is used only infrequently, it can be powered from a 9 V battery, although a mains adapter can also be used.

The receiver

The transmitted information is demodulated by the infrared receiver IC4, shown in **Figure 3**. This very sensitive IC, which is specially tuned to work at the 36 kHz carrier frequency, contains a photodiode, an amplifier stage, a filter and a demodulator. Resistor R17 and capacitor C5 form a supplementary

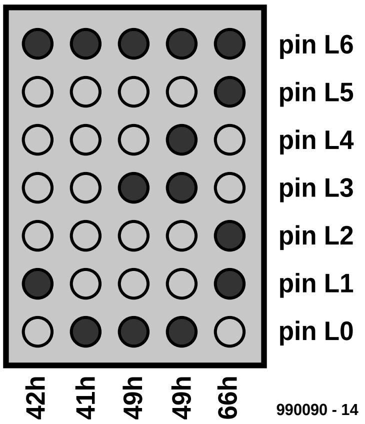


Figure 4. How the numeral '3' is represented on a 5 x 7 matrix display element.

COMPONENTS LIST
(receiver)

Resistors:
R1-R7 = 47Ω
R8 = 1MΩ
R9 = 100kΩ
R10-R16 = 750Ω
R17 = 100Ω

Capacitors:
C1,C2 = 33pF
C4 = 100nF
C5 = 10µF 16V
C6 = 2000µF 25V (or 2200µF 25V)
C7 = 2000µF 16 V (or 2200µF 16V)

Semiconductors:
B1 = bridge rectifier B80C1000 (80V piv, 1A)
T1-T7 = BC557
IC14-IC20 = see text
IC21-IC25 = 74164
IC26 = COP8782C
(order code **996527-2**)
IC27 = ZSM560
IC28-IC32 = ULN2803 (Sprague)
IC33 = 93C66CB1 (ST-Microelectronics)
IC34 = TFMS5360
IC35 = 7805

Miscellaneous:
Mains adaptor socket
Small heatsink for IC35
X1 = 10MHz quartz crystal
Disk. source code file, order code
996032-1

low-pass filter, which guarantees error-free reception. The microcontroller (IC26) samples the signal every 400 µs. Any transient interference that might be present is suppressed by a special software algorithm, which evaluates the lengths of both the pulses and the intervening gaps and compares them to reference values. Finally, the software computes the parity of the received data and compares this to the state of the received parity bit.

LED matrix display devices with a 5×7 matrix are used for representing the characters. Although a LED matrix display costs marginally more than a set of 35 separate LEDs, it is significantly easier to handle. Since not all of the matrix diodes can be driven at the same time, they must be switched on sequentially using a multiplexing process, which is not noticeable to the user. Seven LEDs at most in one matrix column, are illuminated at any one time. Since the eye cannot respond as fast as the individual segments are switched on and off, it sees an image consisting of 245 points (7×35).

Each display cycle starts with a high level on the data input of the first shift register (IC21). This high level lasts for one clock period. The five cascaded shift registers are clocked simultaneously every 400 µs under interrupt control, so that the active column moves stepwise from QA of IC21 through to QH of IC25. IC28 through IC32 are simple driver ICs, which provide sufficient current for the matrices. Every column of the matrix is assigned to a RAM location in the microcontroller. The information to be displayed appears for 400µs on the microcontroller outputs L0 through L6, according to which column is being driven. Transistors T1 through T7 are used here as drivers. Due to the use of multiplexing, the LEDs must be driven at higher than usual current levels in order to make them bright enough to be seen in daylight. To obtain sufficient brightness, you should use matrix displays with an optical efficiency of at least 3 mcd at 20 mA.

The information for the running text display is stored in a non-volatile 512-byte serial EEPROM (IC33). In order to allow the data to be quickly recalled, the EEPROM is addressed via the Microwire interface of the microcontroller at a clock frequency of 500kHz. The PC keyboard delivers the scan code of each key via the infrared interface, for example the code 26_H for the numeral ‘3’. In order to display this numeral on a 5×7 matrix, as shown in Figure 4, the scan code is converted using a look-up table, which in this case yields the values 42_H, 41_H, 49_H, 59_H and 66_H. These are applied to each column in turn to display a ‘3’. IC36 is a ZSM560, which produces the power-on reset pulse for the microcontroller. The same type of IC is used in the transmitter circuit.

The current consumption of the receiver is around 25 mA when all displays are dark, and around 100 mA (average) to 200 mA (peak) when the display is operating. Here the use of a small battery is not such a good idea. A 12-V mains adaptor is a suitable power source, or a small 12-V sealed lead-acid battery (or a car battery) can be used if the display must be independent of the mains.

Operation

When the power is switched on, the running text that was last entered is automatically displayed. If no text has yet been programmed, ‘ELEKTOR’ appears on the display. Connect the transmitter to a PC keyboard and then apply power to it. If the transmitter is working properly, the Scroll LED should be illuminated on the keyboard. Press the F2 key to clear the display and cause a cursor to appear. You can now enter the desired text. Press the Shift key briefly to switch between lower-case and upper-case characters. This will cause the appearance of the cursor to change. Incorrectly entered characters can be deleted using the Backspace key, up to the first character entered. It is not possible to erase previously entered characters; if this is necessary, press the Esc key to end the current entry session and then press F2 to start anew. Press Enter when you have finished entering the text. After this, the running text display will start automatically.

After each text display cycle, the time of day is automatically displayed for about 15 seconds. The time can be set using the F1 key. If you do not want this alternating display mode, you can use the F3 and F4 keys to select a different mode. The meanings of the keyboard keys are explained in the ‘Keyboard Input’ box. Since the microcontroller does not have a real-time clock with a separate 32-kHz crystal, the clock keeps relatively poor time, due to the high clock frequency and the tolerance of the crystal. A small trimmer capacitor in place of C1 can help to improve the situation.

(990090-1)

Design editing: K. Walraven

Keyboard input

Esc:	Cancel input
F1:	Enter the time of day
F2:	Enter the running text
F3:	Running text only (on/off)
F4:	Time of day only (on/off)
Shift:	Switch between upper and lower case
Return:	End the entry session and start the running text display
Delete:	Erase the character in the input window

This design was originally 'knocked together' to test out a prototype DAC circuit to allow the digital codes to be entered manually. The circuit was later modified to test an opto-isolated low side switch which required an open collector transistor driver. The final circuit combines the virtues of both designs.

Hardware design: Adrian Grace

byte generator

for testing DACs and digital controls

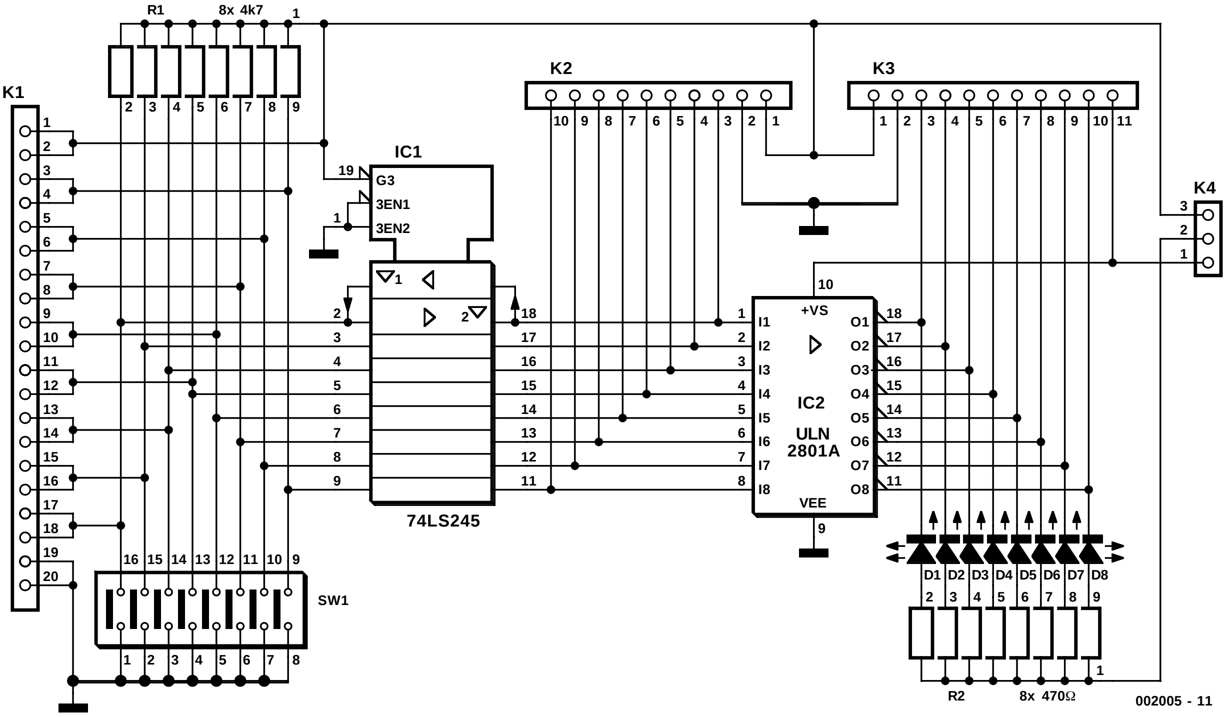


Figure 1. Circuit diagram of the byte generator.

In the circuit diagram, **Figure 1**, SW1 is a 16-pin 8-way DIL switch and is fitted into a 16 way DIL socket (more about this later). The common side of the switch is grounded and the switched side is pulled up to +5V via a 4.7 kΩ SIL resistor network (R1). This is then connected to K1 (which is a doubled up 10-way SIL header or 20-way IDC header), and from there to the inputs of IC1, a 74LS245 which is configured as a buffer. The outputs of IC1 are connected to both K2 and the inputs of IC2. IC2 is a ULN2801A, which is an octal Darlington driver chip with open collector outputs. The outputs of IC2 are available on K3.

Operation of SW1 will result in a TTL output on K2, or an open collector output on K3. As can be seen from **Table 1**, the pin-out for K2 and K3 are virtually the same with the exception of the extra terminal (pin 11) on K3. The ULN2801A (IC2) incorporates internal protection diodes for driving inductive loads — like relays. These internal diodes are commoned together on pin 10 of IC2 and should be connected to the voltage supply of the load. This will 'shunt' any inductive kicks created by switching the load, back into the load's power supply away from the circuit itself.

The circuit can be powered via pins 1 (+5 V) and 2 (0 V) of K2 or K3 and pin 11 of K3 as required, depending upon the application.

The main circuit also includes a simple logic indicator. If the circuit to be monitored is connected to K1, a series of eight LEDs connected to the open-collector outputs of IC2 shows the circuit's status. Connector K4 allows the LED supply voltage to be selected. A link between pins 2 & 3 for +5 V operation, and between pins 1 & 2 for an external voltage source.

Extensions

With a simple extension, this circuit can be modified to include an external clock source — see **Figure 2**.

If the 8-way switch is removed (or ensured that all switches are open) and a daughter board is plugged into K1, the main circuit can be driven by a clock source, rather than manually.

The external clock source is connected to the main board via a 20-way ribbon cable. I found it easier to use a 20-way IDC ribbon cable connection (2 × 10) even though the 10 signal lines are doubled-up, than attempt to use a 10 way crimp connector version (1 × 10). The cable is terminated at the clock source board by a 20-way transition, and at the main board end in a standard 20-way IDC connector.

The clock source itself is based around IC1, a 74HCT4040. This is a +5 V TTL output version of the standard CMOS 4040 chip. Eight sequential outputs, Q0 through Q7 (CT0 through CT7) are fed to the ribbon cable connection whilst Q8 through Q11 (CT8 through CT11) are not connected. IC1 is reset on power-up via R1-C1, and D9 discharges C1 on power down.

The (TTL-level) clock source is connected to GND and CLOCK. Depending on the frequency required, connecting a length of wire to CLOCK may be used as a simple clock source by relying on mains pick-up.

(002005-1)

Article editing: Jan Buiting

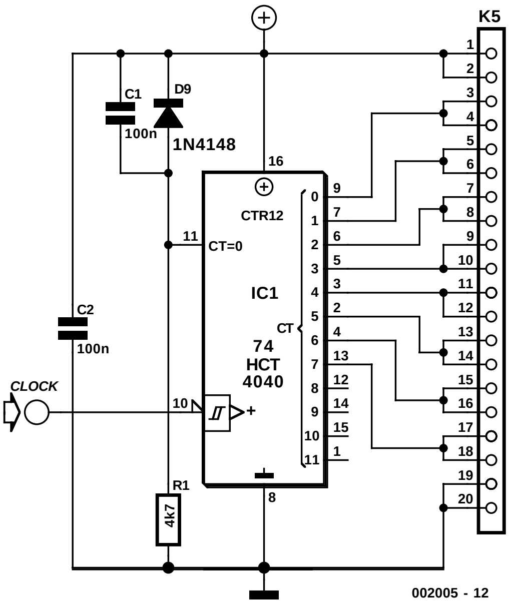


Figure 2. Optional clock extension circuit for connecting to K1 of the byte generator circuit.

COMPONENTS LIST

Resistors:
R1 = 8 × 4K7Ω SIL resistor pack
R2 = 8 × 470Ω SIL resistor pack

Integrated Circuits:
IC1 = 74LS245 or 74HCT245
IC2 = ULN2801A

Miscellaneous:
D1-D8 = 5 mm ⇔ 2mm wide LED, high efficiency
K1 = 20 way IDC connector
K2 = 10 way SIL pin header
K3 = 11 way SIL pin header
K4 = 3 way SIL connector with jumper
16 way turned pin DIL socket

COMPONENTS LIST
Clock divider extension

Resistor:
R1 = 4k7Ω

Capacitor:
C1 = 100nF

Semiconductor:
D9 = 1N4148

Integrated Circuit:
IC1 = 74HCT4040

Miscellaneous:
K4 = 20 way DIL transition
K5 = 20 way IDC
20 way ribbon cable

Table 1. Connector pin functions

K1 pin #	Function	K2 pin #	Function	K3 pin #	Function
1,2	+ 5 V	1	+ 5 V	1	+ 5 V
3,4	DI-1	2	0 V	2	0 V
5,6	DI-2	3	D0-1	3	D0-1
7,8	DI-3	4	D0-2	4	D0-2
9,10	DI-4	5	D0-3	5	D0-3
11,12	DI-5	6	D0-4	6	D0-4
13,14	DI-6	7	D0-5	7	D0-5
15,16	DI-7	8	D0-6	8	D0-6
17,18	DI-8	9	D0-7	9	D0-7
19,20	0 V	10	D0-8	10	D0-8
				11	V+

…and couldn't find anywhere, you will find in this Elektor article (and where else would you find it?). It's a natural idea to use a PC keyboard for developing microcomputer applications, for example, in order to send commands or respond to specific actions. Why go to the trouble of building your own keyboard when you can use a ready-made (and inexpensive) PC keyboard? The only problem is that you first have to know exactly what signals a PC keyboard supplies.

By F. Wohlrabe

PC keyboard encoding

Everything you ever wanted to know about the signals from a PC keyboard…

In addition to the ready availability, low cost and accustomed manner of use of a PC keyboard, connecting a PC keyboard directly to a microcontroller system has the advantage that it makes valuable port pins available that otherwise would be used for polling a keyboard built from individual components. A PC keyboard, by contrast, produces a serial signal, and is thus an ideal complement to a microcontroller project. Of course, the manner in which the signal from the PC keyboard is constructed has a few special features. Two lines are used for the serial data transfer. One of these, labelled data, transfers the data, while the second one transfers the clock. The serial data transfer protocol, which is fairly complex, is explained below.

Key codes

The most widely-used type of keyboard is the MF2 model ('multi-functional version 2'). It was originally developed by IBM for computers in the XT, AT and PS/2 series. This model has become an industry standard in the meantime, and almost all PCs are equipped with it. The keyboard itself contains a 'keyboard controller', which generates the key codes and provides for communication with the keyboard interface of the PC. The keyboard controller is usually a mask-programmed microcontroller. Data are sent and received according to the IBM protocol. Commands can be used to control the LEDs, specify the delay and rate for key repetition, and select the scan code set. The MF2 keyboard has three different scan code sets. Set 1 is used by XT/PC and PS/2-30 compatible computers, while set 2 is used by AT computers and all other PS/2-compatible computers. Set 3 supports workstations and terminal emulation on the PC. Country-specific keyboard drivers in the operating system translate each key press into the desired character.

When a key is pressed, the keyboard produces a Make code. This code corresponds to the scan code for that key. The repeat function causes the Make code to be continuously repeated if the key is held down long enough. The delay time and repetition rate of the repeat function are programmable.

When a key is released, the keyboard produces a Break code. However, if scan code set 3 is selected, no Break code is generated and the repeat function is disabled. After a reset, the keyboard selects scan code set 2 as a default.

You should bear in mind that a PC/XT keyboard cannot be programmed, since its internal controller cannot accept data. Only with the introduction of the AT computer did the keyboard become more user-friendly, since the behaviour of an AT keyboard can be adapted to the needs of the user via software. The following information relates to a keyboard that is set to operate in the AT mode.

Sending and receiving

Figure 1 shows the pin assignments of the two commonly-used keyboard plugs (the 5-pin DIN plug and the 6-pin PS/2 plug). The keyboard is powered with 5 V from the PC. Its maximum current consumption is around 200 mA.

In general, the clock rate is set by the keyboard. It lies in the range of 10 to 16.7 kHz. Data are sent using a start bit (always 0), eight data bits with bit 0 first, an odd-parity bit and a stop bit (always 1). **Figure 2** shows the data transfer timing diagram.

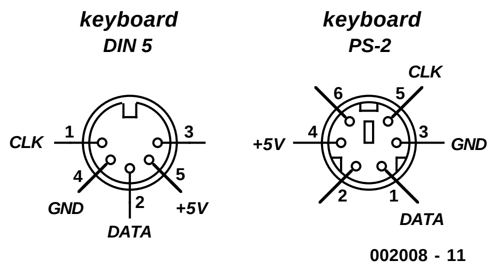


Figure 1. Pin assignments of the standard PC keyboard connectors (viewed from the front).

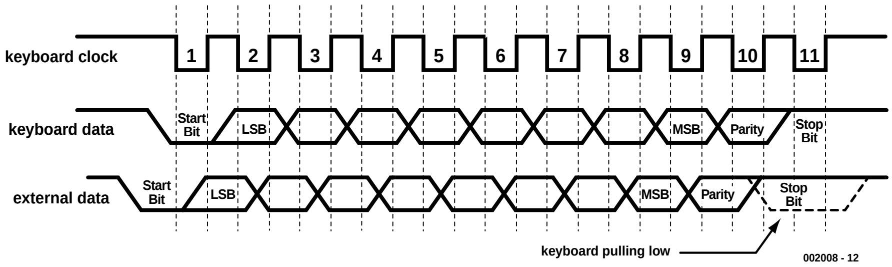


Figure 2. Timing diagram for serial data transfers between the keyboard and a PC.

If an external device or system (which is normally a PC) wants to send data to the keyboard, the keyboard recognizes this by the fact that the data line is pulled to earth by the external device (the PC). The keyboard responds by sending the clock signal, and it expects the data to be sent synchronized to the clock signal. After the data transfer, the data line must exhibit a High level, which acts as the stop bit. The keyboard will continue to send the clock until this condition is satisfied. Following this, the command FE_H is sent to request a new data packet. Data are accepted on the rising edge of the clock signal. After the stop bit has been detected, the keyboard controller holds the data line Low for the duration of one bit period. The keyboard answers every command that it receives, within at most 20 ms, by sending the byte FA_H (ACK), except for the ECHO and RESET commands.

The keyboard sends data in the AT format to an external device using the following process.

Before sending data, the keyboard controller first tests whether the clock or data line is at earth level. Communications can be blocked by holding the clock line Low. In this case, the keyboard holds the data to be sent in an internal buffer. The keyboard can send data only if both the clock and data lines are at a High level. It then sets the data line Low (for the start bit) and generates the clock signal. The data are valid on the falling edge of the clock signal and change after the rising edge.

In order to implement a MF2 keyboard connection in a microcomputer system, you will also need to know certain information regarding the most important commands and return codes for an AT keyboard. These are described in the following section. The key codes, which are the codes that the keyboard produces according to the selected scan-code set when keys are pressed, are listed in **Table 1**.

The most important commands

SET/RESET MODE INDICATORS → code ED_H

This two-byte command controls the behaviour of the LEDs.

Command: ED_H

Command: 0000 0xxx

Bit 0: Scroll lock

Bit 1: Num lock

Bit 2: Caps lock

1 = LED on, 0 = LED off

ECHO → code EE_H

The keyboard answers this command with EE_H. It can be used to confirm the presence of a keyboard.

SCAN CODES SELECT → code $F0_H$

This two-byte command selects the scan code set. Scan code set 2 is selected by default after a reset. However, scan code set 3 recommends itself for microcontroller applications, due to its simplicity. With scan code set 3, no Break code is sent for almost all keys and the repeat function is disabled.
Command: FO_H
Command: 0000 00xx
01 = scan code set 1
10 = scan code set 2
11 = scan code set 3

READING ID CODE → code $F2_H$

In response to this command, the keyboard sends three bytes which contain a manufacturer-specific code.
1st byte = FA_H (ACK)
2nd byte = xxxx xxxx
3rd byte = xxxx xxxx

SET TYPEMATIC RATE/DELAY → code $F3_H$

This two-byte command controls the key repeat rate and the delay for starting key repetition.
Command: $F3_H$
Command: 0xxx xxxx
Bits 5 and 6 control the delay, which ranges from 150 ms to 1 s.

Bit 6	Bit 5	Delay (± 20%)
0	0	150 ms
0	1	500 ms
1	0	750 ms
1	1	1 s

Bits 0 through 4 control the repetition rate, which ranges from 2 to 30 Hz. In the following table, only three values are shown as examples.

Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Frequency (± 20%)
0	0	0	0	0	30 Hz
0	1	1	1	1	8 Hz
1	1	1	1	1	2 Hz

SET ALL KEYS → codes $F7_H$, $F8_H$, $F9_H$, FA_H

These commands assign attributes to the keys, as follows:

- $F7_H$: all keys have the repeat function
- $F8_H$: all keys produce Make and Break codes
- $F9_H$: all keys produce only a Make code
- FA_H: all keys have the repeat function and produce Make and Break codes

RESET → code FF_H

This command restores all keyboard settings to their default values.

The most important return codes

BAT COMPLETION → code AA_H

This byte is sent to the external system after the supply voltage has been applied or a reset command (FF_H) has been recognized. In indicates correct execution of the keyboard self-test.

RESEND NAK → code FE_H

This byte is sent in response to a data transfer error.

ACK → code FA_H

This byte is sent to the external device in response to each received command.

OVERRUN → code $00_H/FF_H$

All key presses are stored internally in the keyboard until their codes can be serially transferred to the external device. If the storage buffer overflows, the byte 00_H is sent for scan code sets 2 and 3, while the byte FF_H is sent for scan code set 1.

BREAK CODE PREFIX → code $F0_H$

With scan code set 2, the byte $F0_H$ is sent before the Break code.

Programming example

In conclusion, the manner in which a (microcontroller) system should address an AT keyboard can be illustrated with a simple programming example.
When the 5-V supply voltage is switched on, +5 V is applied to the keyboard. The keyboard controller in the keyboard then executes a self-test. If this is completed successfully, the keyboard sends the byte AA_H.

Next comes the selection of the scan code set. In this example, scan code set 3 is selected using the SCAN CODES SELECT command, as follows:

1 Pull the data line to earth.
2 Send the command code $F0_H$, synchronous to the clock.
3 The keyboard sends the code FA_H (ACK) as confirmation.
4 Pull the data line to earth.
5 Send the command code 03_H, synchronous to the clock.
6 The keyboard sends the code FA_H (ACK) as confirmation.

Now a key can be pressed, and the key code from the scan code 3 set (see Table 1) will be received:

7 Press ‘G’ on the keyboard.
8 The keyboard sends the code 34_H.

A practical application example of the use of a PC keyboard with a microcontroller system is “Text Running Line Display” elsewhere in this Supplement. In this example, a common or variety PC keyboard is used for entering text to be displayed on an LED running-line display. Keyboard decoding is handled by a COP-8 microcontroller, and the data transfer to the running-line display uses an infrared link. The photo at the head of this article shows a small circuit board holding the keyboard decoder and IR transmitter.

(002008-1)

Table 1.
The key codes produced when keys are pressed, for each of the three scan code sets.

Symbol	Scan code set 1		Scan code set 2		Scan code set 3	
	Make	Break	Make	Break	Code	Type
^	29	A9	0E	F0-0E	0E	T
1	02	82	16	F0-16	16	T
2	03	83	1E	F0-1E	1E	T
3	04	84	26	F0-26	26	T
4	05	85	25	F0-25	25	T
5	06	86	2E	F0-2E	2E	T
6	07	87	36	F0-36	36	T
7	08	88	3D	F0-3D	3D	T
8	09	89	3E	F0-3E	3E	T
9	0A	8A	46	F0-46	46	T
0	0B	8B	45	F0-45	45	T
ß	0C	8C	4E	F0-4E	4E	T
‘	0D	8D	55	F0-55	55	T
← Back	0E	8E	66	F0-66	66	T
|← →| Tab	0F	8F	0D	F0-0D	0D	T
Q	10	90	15	F0-15	15	T
W	11	91	1D	F0-1D	1D	T
E	12	92	24	F0-24	24	T
R	13	93	2D	F0-2D	2D	T
T	14	94	2C	F0-2C	2C	T
Z	15	95	35	F0-35	35	T
U	16	96	3C	F0-3C	3C	T
I	17	97	43	F0-43	43	T
O	18	98	44	F0-44	44	T
P	19	99	4D	F0-4D	4D	T
Ü	1A	9A	54	F0-54	54	T
+	1B	9B	5B	F0-5B	5B	T
CapsLock	3A	BA	58	F0-58	14	M,B
A	1E	9E	1C	F0-1C	1C	T
S	1F	9F	1B	F0-1B	1B	T
D	20	A0	23	F0-23	23	T
F	21	A1	2B	F0-2B	2B	T
G	22	A2	34	F0-34	34	T
H	23	A3	33	F0-33	33	T
J	24	A4	3B	F0-3B	3B	T
K	25	A5	42	F0-42	42	T
L	26	A6	4B	F0-4B	4B	T
Ö	27	A7	4C	F0-4C	4C	T
Ä	28	A8	52	F0-52	52	T
#	2B	AB	5D	F0-5D	53	T
Return	1C	9C	5A	F0-5A	5A	T
Shift l.	2A	AA	12	F0-12	12	M,B
<	56	D6	61	F0-61	13	T
Y	2C	AC	1A	F0-1A	1A	T
X	2D	AD	22	F0-22	22	T
C	2E	AE	21	F0-21	21	T
V	2F	AF	2A	F0-2A	2A	T
B	30	B0	32	F0-32	32	T
N	31	B1	31	F0-31	31	T
M	32	B2	3A	F0-3A	3A	T
,	33	B3	41	F0-41	41	T
.	34	B4	49	F0-49	49	T
-	35	B5	4A	F0-4A	4A	T
Shift r.	36	B6	59	F0-59	59	M,B
Ctrl l.	1D	9D	14	F0-14	11	M,B
Alt l.	38	B8	11	F0-11	19	M,B
Space	39	B9	29	F0-29	29	T
Num	45	C5	77	F0-77	76	M
7 Nb	47	C7	6C	F0-6C	6C	M
4 Nb	4B	CB	6B	F0-6B	6B	M
1 Nb	4F	CF	69	F0-69	69	M
/ Nb	E0-35	E0-B5	E0-4A	E0-F0-4A	77	M
8 Nb	48	C8	75	F0-75	75	M
5 Nb	4C	CC	73	F0-73	73	M
2 Nb	50	D0	72	F0-72	72	M
0 Nb	52	D2	70	F0-70	70	M
* Nb	37	B7	7C	F0-7C	7E	M
9 Nb	49	C9	7D	F0-7D	7D	M
6 Nb	4D	CD	74	F0-74	74	M
3 Nb	51	D1	7A	F0-7A	7A	M
Del Nb	53	D3	71	F0-71	71	M
- Nb	4A	CA	7B	F0-7B	84	M
+ Nb	4E	CE	79	F0-79	7C	M
Enter	E0-1C	E0-9C	E0-5A	E0-F0-5A	79	T
Esc	01	01	76	F0-76	08	M
F1	3B	BB	05	F0-05	07	M
F2	3C	BC	06	F0-06	0F	M
F3	3D	BD	04	F0-04	17	M
F4	3E	BE	0C	F0-0C	1F	M
F5	3F	BF	03	F0-03	27	M
F6	40	C0	0B	F0-0B	2F	M
F7	41	C1	83	F0-83	37	M
F8	42	C2	0A	F0-0A	3F	M
F9	43	C3	01	F0-01	47	M
F10	44	C4	09	F0-09	AF	M
F11	57	D7	78	F0-78	56	M
F12	58	D8	07	F0-07	5E	M
PrtSc	E0-2A-E0-37	E0-B7-E0-AA	E0-12-E0-7C	E0-F0-7C-E0-F0-12	57	M
Scroll Lock	46	C6	7E	F0-7E	5F	M
Pause	E1-1D-45-E1-9D-C5	no break code	E1-12-77-E1-F0-14-F0-77	no break code	62	M
Insert	E0-52	E0-D2	E0-70	E0-F0-70	67	M
Del	E0-53	E0-D3	E0-71	E0-F0-71	64	T
←	E0-4B	E0-CB	E0-6B	E0-F0-6B	61	T
Home	E0-47	E0-C7	E0-6C	E0-F0-6C	6E	M
End	E0-4F	E0-CF	E0-69	E0-F0-69	65	M
↑	E0-48	E0-C8	E0-75	E0-F0-75	63	T
↓	E0-50	E0-D0	E0-72	E0-F0-72	60	T
PgUp	E0-49	E0-C9	E0-7D	E0-F0-7D	6F	M
PgDn	E0-51	E0-D1	E0-7A	E0-F0-7A	6D	M
→	E0-4D	E0-CD	E0-74	E0-F0-74	6A	T

Nb = numeric block
M = Make code when key pressed
B = Break code when key released
T = Typematic repeat function with delay & make

P.O. Box 190

We can only answer questions or remarks of general interest to our readers, concerning projects not older than two years and published in *Elektor Electronics*. In view of the amount of post received, it is not possible to answer all letters, and we are unable to respond to individual wishes and requests for modifications to, or additional information about, *Elektor Electronics* projects.

When Electronics was Young (9)

Dear Editor — in the November 1999 issue of Elektor the above interesting series has a brief note on Guglielmo Marconi that is misleading.

Marconi's early experiments were carried out in Italy not England. He established that communication using electromagnetic waves was possible at the age of 20 experimenting at his parent's estate near Bologna. He moved to England as the Italian government showed no interest in his invention and his English mother felt he would be more likely to achieve success in her mother country. At the time, this country was a leading maritime nation and there was no means of communicating with ships once they were out of sight of land.

It was in England that Marconi filed his patent applications, supported by the chief engineer of the Post office, Sir William Preece.

To call Marconi a 'physicist' is a gross insult. If he had been a physicist, he would be unlikely to have achieved transatlantic communication. The physicists of the day held the view that electromagnetic waves travelled in straight lines and were absorbed by land and water. The curvature of the earth created a 50 mile high mountain of water across the Atlantic Ocean so no waves coild possibly reach the New World. Marconi with no formal education ignored the 'experts' and was successful. Neither the physicists nor Marconi knew about the existence of the ionosphere which conveniently reflected his waves from Poldhu to St. Johns.

Beware of experts!

Guy Selby-Lowndes

Our contributor replies as follows.

"Interpretation of certain aspects of history can be contentious. In the Encyclopaedia Britannica, Marconi's entry reads 'Italian physicist and inventor of successful system of radio telegraphy. Received Nobel Prize for Physics in 1909'. Similarly, Chambers Dictionary of Scientists lists Marconi as 'Marconi, (Marquis), Guglielmo, 1874-1937, Italian physicist and engineer, pioneer of radiotelegraphy'. In line with these two renowned publications, I, too, have called Marconi a physicist."

"Owing to the shortness of the article, I could not include the fact that Marconi had started his experiments in Italy. However, his main experiments were carried out in England. While still a physics student at Leghorn technical school, Marconi in 1894 demonstrated the possibility of sending wireless signals over a distance of a distance of some 150 metres at his father's estate near Bologna. However, since he found nobody in Italy interested in his experiments, he and his Irish mother travelled to England a year later. There, in 1896, he demonstrated a transmission of wireless signals over a distance of three miles between Flatholm Island in the Bristol Channel and Penarth. He filed his first patent application for wireless telegraphy at the London Patent Office on 2nd June 1896. In 1897, he transmitted wireless signals over a distance of eight miles across the Bristol Channel. This experiment drew the attention of Sir William Preece, the chief engineer of the post office, as well as of the Press. In 1899, he transmitted Morse code across the English Channel, which attracted attention from the Admiralty, resulting in the installation of radio wireless equipment on Royal Naval ships. In 1901, he transmitted across the Atlantic from Cornwall to Newfoundland. His pioneering work was rewarded by the award (shared) of the Nobel Prize for Physics in 1909."

"It is a debatable point whether or not Marconi or his fellow physicists and engineers knew about the ionosphere. Michael Faraday's experiments in the 1840s and 1850s demonstrated a clear connection between magnetism and electricity. Faraday's friend and colleague, James Clark Maxwell, confirmed and explained Faraday's experimental results by a mathematical theory. Maxwell's calculations showed that magnetic and electric fields affect each other, and that light is a form of electromagnetic waves. Some traditional physicists adhered to the belief that light was a mechanical phenomenon."

"The conflict caused by Maxwell's hypothesis that electromagnetic waves are propagated at the speed of light lasted for many years, until in 1888 Heinrich Hertz, Professor at the Technical College in Karlsruhe, finally closed the debate on the wave character of light. He demonstrated that the electric sparks produced by him caused electrical vibrations which were propagated into space at the speed of light."

"During a cross-Atlantic trip, Marconi had already noticed that telegraph messages could be received over distances of up to 700 miles by day, but at up to 2000 miles by night. This discovery prompted Oliver Heaviside, an English physicist, and Arthur Edwin Kennelly, an American electrical engineer, independently and simultaneously to publish their prediction of the existence of an electrically conductive layer in the upper atmosphere that allows radio waves to follow the earth's curvature instead of travelling in a straight line as predicted by a number of mathematicians. The predictions of Heaviside and Kennelly were demonstrated in 1925 by Sir Edward Victor Appleton, an English physicist who, in 1947, was awarded the Nobel Prize for Physics for his contribution in the exploration of the ionosphere".

"Today, we know that the ionosphere is a region of the earth's atmosphere at a height of 30-300 miles where short-wave radiation from the sun partly ionizes gas molecules and atoms, leaving them positively charged. The ionized layers reflect short-wavelength radiowaves, which makes long-distance radio communication possible. The ionosphere is layered according to the concentration of free electrons, called D and E layers (also called the Heaviside or Kennelly layers, depending on which side of the Atlantic the term is used), which result from molecular ionization, and the upper layer, termed F layer or Appleton layer, which results from atomic ionization. The thicknesses of the layers vary with latitude, season, time of day, and solar activity".

"Since much of the research into propagation was going on and published while Marconi, Braun, Slaby, and Arco, were conducting their experiments in England and Germany, it is extremely likely that they read about it."

2-metre band converter

capture 144 MHz DX signals

If you have a general coverage shortwave receiver available and would like to extend its frequency range with the two-metre amateur radio band, the present design is for you. Easy to build from low-cost parts, the converter should make an excellent entry-level project for budding radio enthusiasts. The converter is also a prefect companion to the general-coverage multi-mode SW receiver described last year in this magazine.

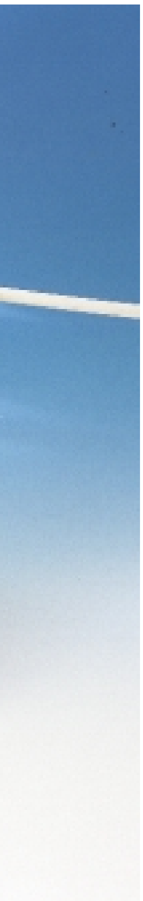
Design by G. Baars

The 2-metre radio amateur band is still the most popular band worldwide. It extends from 144 to 146 MHz in most European countries, and from 144 to 148 MHz is some other countries like the U.S.A. and Australia. Traditionally, the band is associated with short-range communication over distances of up to 50 miles or so using narrow-band frequency modulation (NBFM) and power levels up to about 50 watts. This is also called ‘local traffic’ by some radio amateurs. Thanks to the relatively short antenna lengths and general profusion of cheap Japanese high-tech rigs and converted PMR kit, the 2-m band is also the place to be for mobile and portable communication, witness the presence of amateur-built and operated repeater stations in many countries and areas.

The lower part of the 2-metre band is reserved for narrow-band modes like CW (Morse) and SSB (single-sideband). Mainly because of the smaller bandwidth and resultant better signal-to-noise ratio for weak signals, these modes offer far greater ranges than NBFM. The ‘sound’ of the band section between say 144.000 MHz and 144.400 MHz is therefore not unlike that of a shortwave band like 10 metres (28 MHz). Provided you use a good directional antenna (like a yagi) you should be able to pick up the CW idents of low-power beacons as well as CW and SSB signals from stations far beyond the range of NBFM.

Why a converter?

Many beginners to the radio hobby start will start out with a second-hand shortwave receiver. This will typically be a general-coverage type for CW/USB/LSB/AM/RTTY reception between 150 kHz and 30 MHz. The Yaesu FRG-7 is an excellent example of such a receiver, and although its design is 25 years old, it is still in popular demand in the radio amateur trade. The same beginners will also lack the funds (and a licence) to buy an all-mode VHF transceiver, so why not add 2-metre band reception to the available shortwave receiver? With some luck, this has an NBFM mode, too, so you can also listen to 'local' traffic and get to know the hams in your area.



It should be noted that reception of DX (long-distance) signals in the 2-metre band requires a good directional antenna with a gain of at least 10 dB and low-loss coax cable to the receiver (or converter) input. Whatever low-noise pre-amplifier you may have in mind, experience shows that it is beaten hands down by a good antenna in an elevated position.

Following a well-established tradition in ham radio, the present converter mixes the 2-metre band signals down to the 10-metre band (28.0-29.7 MHz).

How it works

The circuit diagram of the 2-metre band converter is given in **Figure** 1. As you can see it employs only five active components, and these are all of the common or garden variety.

The design consists of four sections, a local oscillator, a mixer, an input stage, and an output stage which will be discussed separately below.

Local oscillator (LO)

Transistor T1 and quartz crystal X1 are configured as an oscillator with an output frequency of 38.667 MHz. The crystal operates in third-overtone mode. Trimmer C1 is available to net the oscillator. The oscillator output signal is fed to frequency tripler T2 whose collector circuit is tuned to 116 MHz by L3 in combination with trimmer C7. The local oscillator signal has a level of about 100 mV peak to peak and is inductively coupled (via L4) to the mixer.

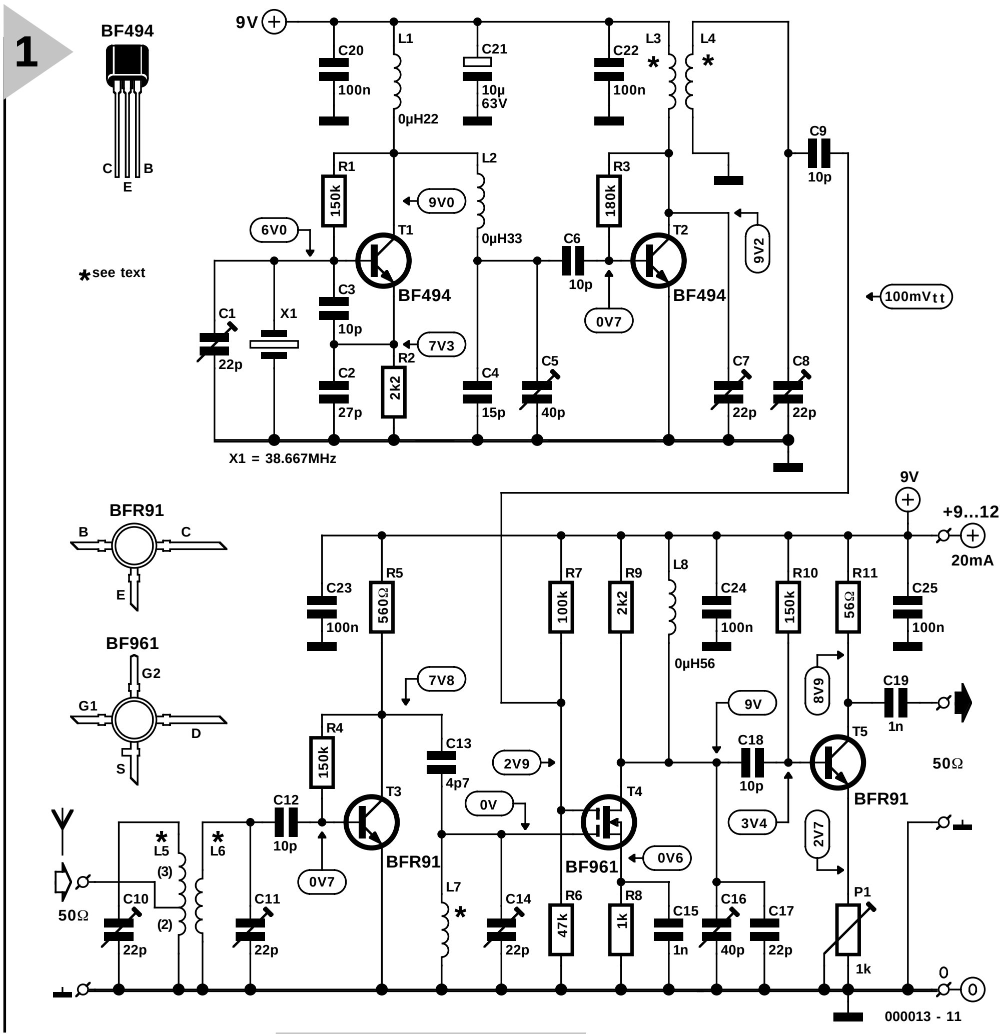


Figure 1. Circuit diagram of the 2-to-10-m converter.

Mixer

The mixer in the converter is a type BF961 dual-gate MOSFET, T4. The local oscillator signal is applied to gate 2 (G2) and the RF input signal, to gate 1 (G1). Note that G2 is held at a fixed potential of about 2.9 V by R6-R7, while G1 is dc-wise at ground potential. This is the traditional configuration, with the G2 resistors determining the conversion gain. The mixer products are available at the drain of the BF961. These products are, in principle: 144 + 116 = 260 MHz, 144–116 = 28 MHz and the LO signal residue at 116 MHz. The combination L8/C16/C17 is tuned to 28.8 MHz and serves to suppress the 116 MHz LO component — given the frequency difference between these components sufficient suppression is not hard to achieve.

Input stage

The signal from the 2-metre antenna is inductively coupled to the base of T3, a low-noise VHF/UHF transistor type BFR91. The input inductor pair L5-L6 is accurately tuned to 144 MHz by trimmers C10 and C11. The input bandfilter serves to suppress image frequencies at 116–28 = 88 MHz and at the same time match the transistor to the cable impedance of 50 Ω. The amplified signal is capacitively coupled to gate 1 of the mixer via C13.

Output stage

The main function of the driver stage around T5 is to provide a good match to the receiver input (50 Ω). The gain of this stage is made adjustable with preset P1 to ensure that no overdriving occurs with sensitive shortwave receivers.

The converter is powered by a regulated and well decoupled supply with an output of between 9 and 12 volts. Current consumption will be of the order of 20 mA.

Construction

The converter is built on a single-sided printed circuit board of which the design is shown in Figure 2. This board is available ready-made through the *Elektor Electronics* Readers Services.

Before you start soldering away we recommend you make inductors L3-L7. This is not at all difficult. Take a pen or a drill bit with a diameter of 4.5 mm

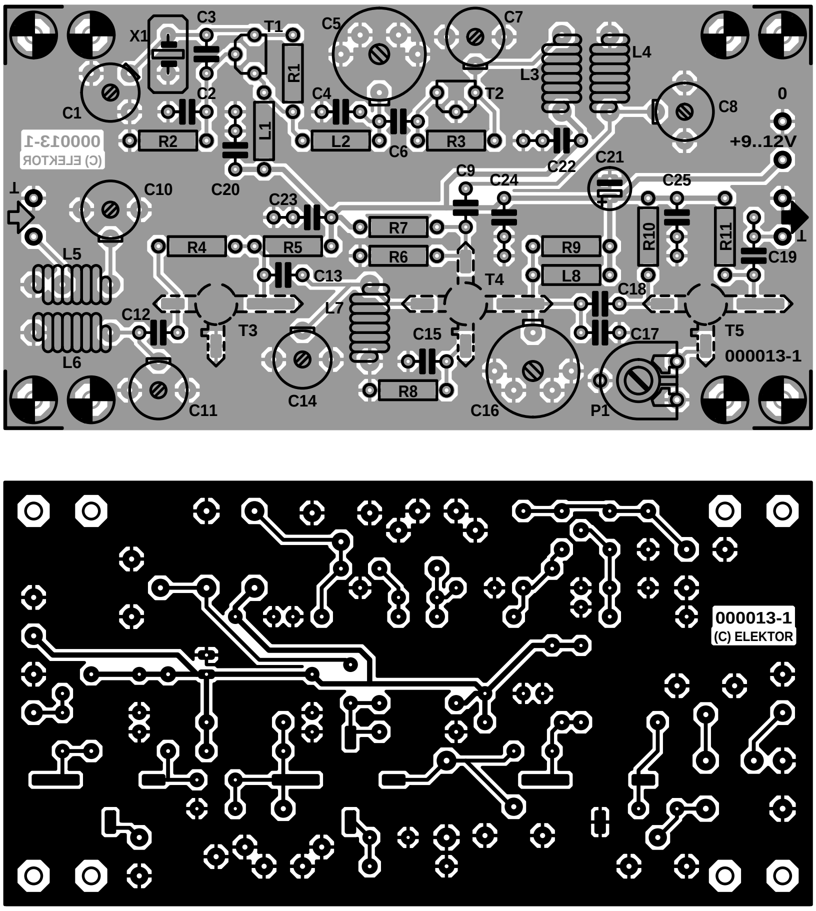


Figure 2. Copper track layout and component mounting plan of the single-sided PCB. Three transistors are mounted at the solder side!

COMPONENTS LIST

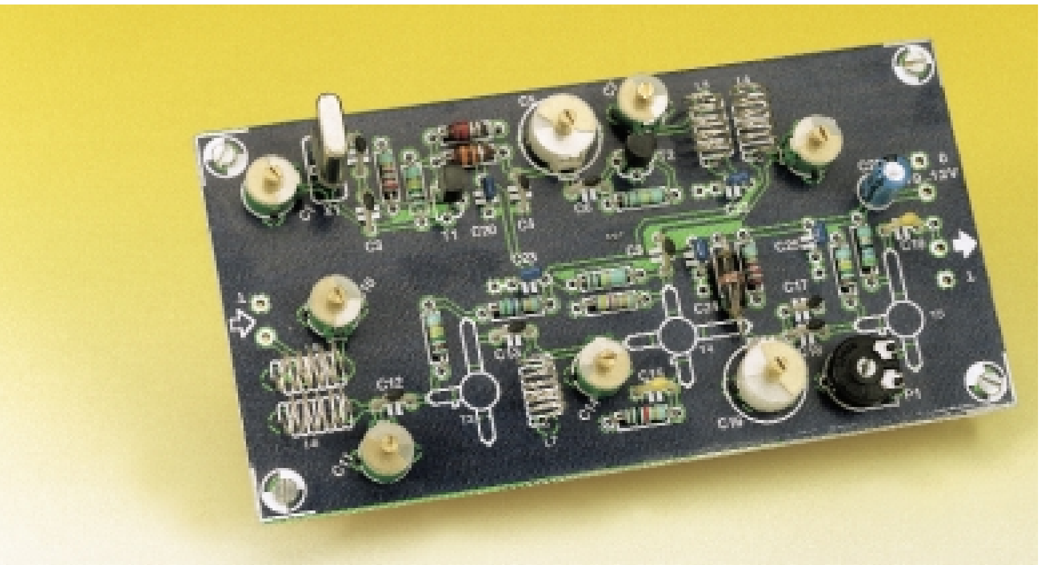
Resistors:
R1,R4,R10 = 150kΩ
R2,R9 = 2kΩ2
R3 = 180kΩ
R5 = 560Ω
R6 = 47kΩ
R7 = 100kΩ
R8 = 1kΩ
R11 = 56Ω
P1 = 1kΩ preset H

Capacitors:
C1,C7,C8,C10,C11,C14 = 22pF trimmer
C2 = 27pF
C3,C6,C9,C12,C18 = 10pF
C4 = 15pF
C5,C16 = 40pF trimmer
C13 = 4pF7
C15,C19 = 1nF, raster 5mm
C17 = 22pF
C20,C22-C25 = 100nF ceramic
C21 = 10µF 63V radial

Inductors:
L1 = 0.22µH miniature choke
L2 = 0.33µH miniature choke
L3-L7 = 5 turns silver-plated wire, dia. 0.8mm (SWG20), internal diameter 4.5mm, length 10mm
Distance between coupled inductors: 1mm, tap at 2 turns from ground side
L8 = 0.56µH miniature choke

Semiconductors:
T1,T2 = BF494
T3,T5 = BFR91
T4 = BF961

Miscellaneous:
X1 = quartz crystal 38.667MHz (3rd overtone) (Mainline, tel. 0870 2410810)
PCB, order code **000013-1** (see Readers Services page)
Case: e.g. Hammond 1590B 56×107×25 (inside dimensions)



and wind 5 turns of SWG20 (approx. 0.8 mm dia.) silver-plated wire around it. Then stretch the turns evenly until the coil has a length of about 10 mm. Only on L5 you ‘tap’ the inductor at 2 turns from the side you want to connect to ground (look at the component overlay). The tap is made by means of a small piece of bare wire. Make sure it does not short-circuit the adjacent turns! Coupled inductors L5-L6 and L3-L4 should be spaced 1 mm apart.

Next, fit all the parts on to the board, except transistors T3, T4 and T5. Remember, careful and accurate soldering work will be rewarded with a circuit that works spot-on.

To keep parasitic capacitance as small as possible, **the BFR91 and BF961 transistors are fitted at the solder side of the board. This is indicated by their dashed outlines on the component overlay**. Look very carefully at the orientation aids on these transistors to make sure they are mounted the right way around. On the BFR91, the collector is the longest pin; on the BF961, the source has a small tab and the drain is the longest pin.

The completed board has to be fitted in a metal case. For our prototype, we used a small diecast case from Hammond. The converter RF input and output may be BNC or SO239 style sockets, depending on what you have available. The connections between the sockets and the relevant PCB pins should be made in coax cable, for example, RG174 or RG58.

AN ADJUSTMENT TOOL

We are sure that the simple RF probe shown in **Figure 3** will pay dividends in adjusting RF circuits. Build it and you will wonder how you ever did without it.

The probe consists of an aluminium pen case (a felt pen, cleaned out, of

3

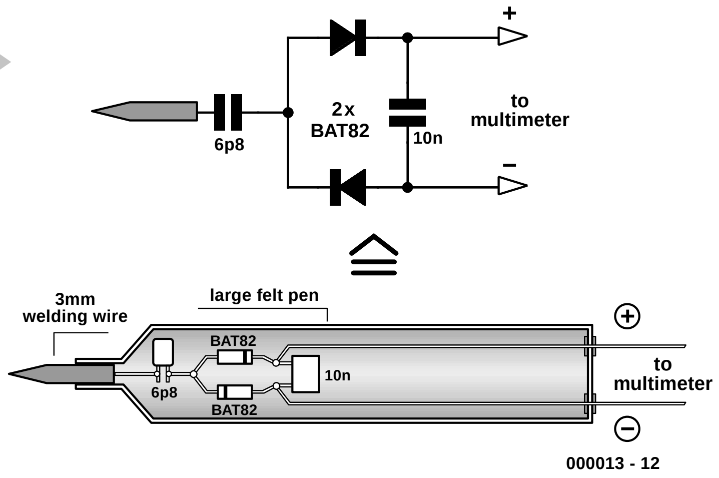


Figure 3. Build this simple RF probe and adjusting the converter will be a breeze.

course) in which a small diode detector is housed. The end of the copper or welding wire is carefully filed down to give a sharp tip. The choice of diode is not critical. While SHF diodes like the 1S99 will enable measurements well into the GHz range, the run-of-the-mill BAT82 will be fine for VHF circuits like the present converter.

The probe is only intended to give relative indications, providing an easy means to 'peak' inductors on their resonance frequency. It will only lightly load the tuned circuit and does not require a ground connection. The output voltage is fed to the inputs of a voltmeter — preferably analogue so you can see the 'trend'. In this case, needle movement on an antique moving-coil meter is rather more useful than rolling digits on a DVM.

Adjustment

Start by setting all trimmers on the board to full mesh, except C1, which is set to mid-travel. Connect the converter to the receiver and switch on the power supply. Set P1 to the centre of its travel. Measure the current consumption. If it as expected, proceed with the adjustment procedure described below. "Hot" means carrying RF, "Cold" means not carrying RF, i.e., ground or positive supply. "Peak" means adjust for maximum reading on the voltmeter connected to the probe, or for maximum S-meter reading on the receiver. In case of the probe, the absolute value you measure is irrelevant, it's the peak you should be looking for.

1. Put the probe tip on the hot side of C5 and adjust this trimmer for maximum reading on the voltmeter.
2. Connect the probe to about 1 turn from the cold side of L3 and peak C7. You want the first peak starting from fully meshed. If not, you tune to f_{osc} × 4 instead of f_{osc} × 3.
3. Connect the probe to 1 turn from the cold side of L4 and peak C8. Use the first peak starting from fully meshed. If not, you tune to f_{osc} × 4 instead of f_{osc} × 3.
4. Set C10, C11 and C14 to half mesh.
5. Tune the receiver to 28.800 MHz and adjust C16 for maximum noise.
6. Ensure that a relatively strong input signal is available on a frequency between 144.800 and 145.000 MHz (RF generator or ask a local radio amateur). Peak C10, C11 and C14 for best reception. Reduce the input signal as required to ensure you can always find a peak.
7. Adjust C1 so that the frequency readout on the receiver matches the signal frequency, for example, 144.800 MHz = 28.800 MHz.
8. Remove the input signal and adjust P1 so that the S meter on the receiver just starts to deflect.
9. Tune to a weak signal in the 2-m band and carefully adjust C10, C11 and C14 for highest S-meter indication.

That concludes the adjustment of the converter.

Weather-satellite band

By changing the LO injection frequency to 109 MHz, it should be possible to use the converter for reception of low-orbiting weather satellites in the 137 MHz band. A quartz crystal of 36.333 MHz (again, 3rd overtone) is then required, as well as readjustment of all trimmers for the slightly lower frequencies.

(000013-1)

Band plan for 144–146 MHz
(IARU recommendation)

144.000 – 144.500 MHz

Reserved for DX traffic. Some important sub-bands:

144.000 – 144.025:	*EME (earth-moon-earth or 'moonbounce')*
144.050:	*CW calling*
144.100:	*Meteor scatter in CW*
144.150:	*CW DX*
144.300:	*SSB calling*
144.400 – 144.490:	*Beacons*
144.490 – 144.500:	*Beacon guard band, no transmissions*

144.500 – 144.800 MHz

All modes, including

144.500	*SSTV calling*
144.600	*RTTY calling*
144.700:	*FAX calling*
144.750:	*ATV calling*

144.800 – 144.990 MHz

Digital modes (Packet Radio)

145.0000 – 145.1875 MHz

Repeater input frequencies (12.5 kHz raster, shift 600 kHz)

145.2000 – 145.5875 MHz

Simplex channels, FM, 12.5kHz raster.

145.6000 – 145.7875 MHz

Repeater output frequencies (12.5 kHz, shift 600 kHz)

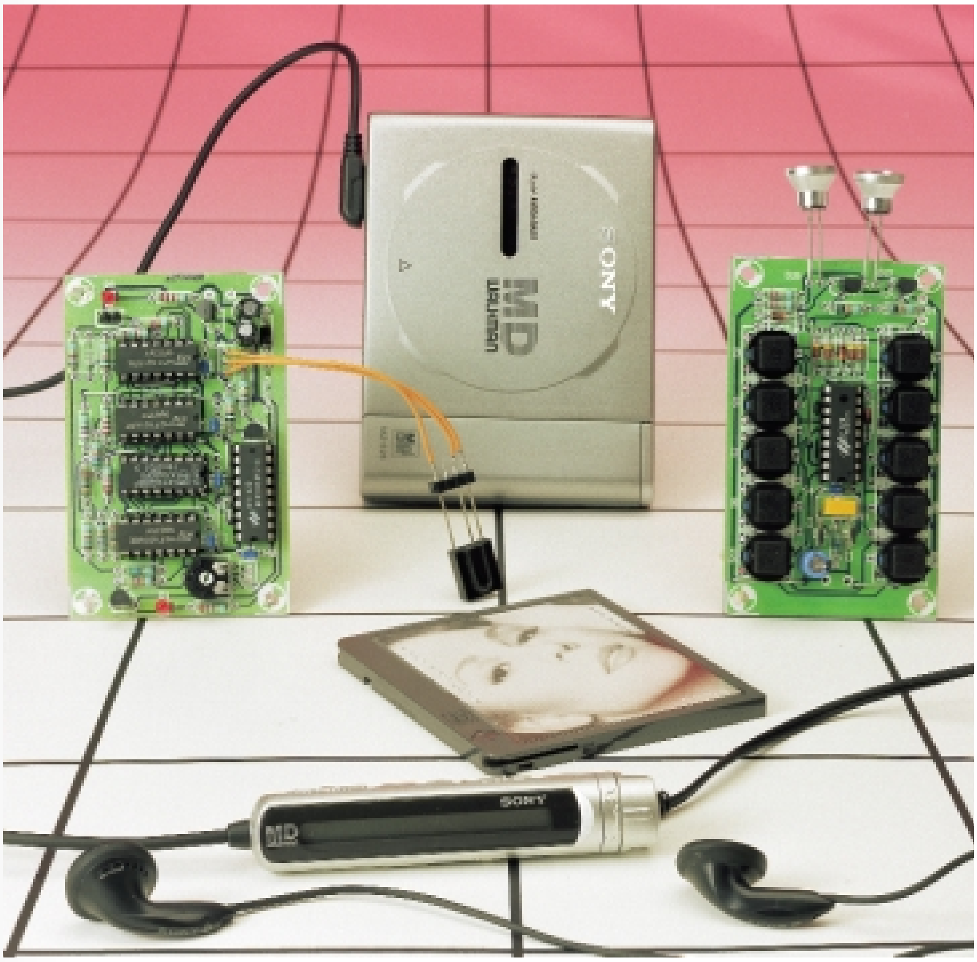
145.8000 – 146.0000 MHz

Satellite services

infrared remote control

for Sony MZ-R30 MiniDisc Walkman

a useful add-on for a popular player



The MZ-R30 MD recorder comes with its own cable-operated remote control, which is primarily designed for Walkman-ish operation. However, Brian Houghton's own application for recording choral rehearsals needed to have a means of controlling the stop/start function from a location that made cable control impossible. Here's how Brian solved the problem — elegantly and without breaking the bank.

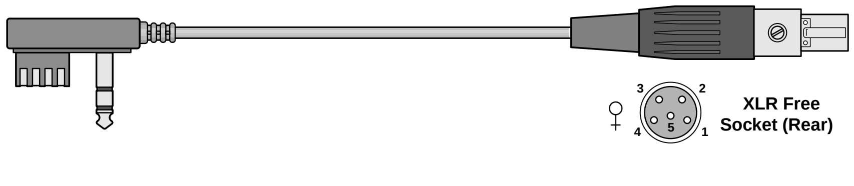
Design by Brian Houghton G4BCO

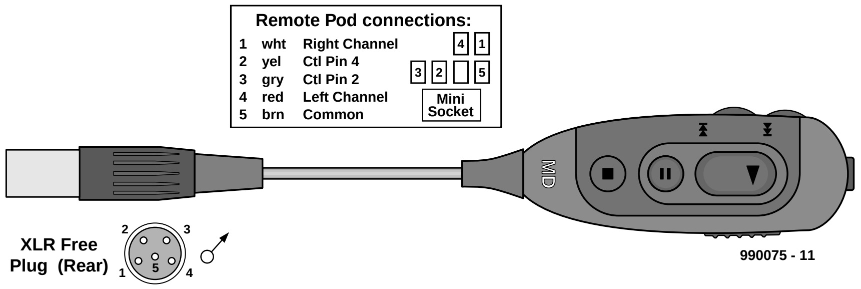
Rather than starting straightaway with the technical description of the project it may be interesting to tell you some of the design history.

I found the website "The Minidisk Community Pages" on *http://www.amulation.com/minidisc* very helpful with information on how the Sony MZ-R30 remote functions are controlled by different resistance values across a pair of wires.

Initially a 173-MHz licence-free "HomeCall "type transmitter and a suitable receiver was obtained, and the receiver mounted in a control box with some CMOS logic. This RF system

Figure 1. Modification to the remote control cable supplied with the Sony MD MiniDisc walkman.





Remote Pod connections:

1	wht	Right Channel
2	yel	Ctl Pin 4
3	gry	Ctl Pin 2
4	red	Left Channel
5	brn	Common

Mini Socket: 4 1 / 3 2 5

Figure 2. The remote pod can still be used if you give it a connector again.

works well up to 100ft, but is slow to respond (5 seconds) and only allows for the one function, in this case “pause”. The details of this remote control can be found on : *http://www.amulation.com/minidisc/mzr30_remote_radio/index.html.*

Several emails from people regarding that design, were received and one in particular from someone who wanted some help with a college project to build a full function IR remote control within a budget of £50. This spurred the author to have a go.

Design

This design logically splits into two parts:

- The **hand held transmitter** containing the function select push button switches, encoder and infrared transmitter.

- The **receiver/decoder** containing the infrared receiver, decoder and resistance ladder selector analogue switches.

The remote control connector on the Sony MD Walkman is unique and unfortunately is not obtainable as a spare item, and since the whole remote cable is £50 to purchase separately, it was decided to perform minor surgery on the existing cable.

Remote cable modifications

The remote control pod was opened and the existing cable disconnected. A mini XLR 5-way socket was fitted to the free end of the old cable. This cable is then suitable to connect the IR Control Unit to the Sony Walkman. The modification is illustrated in **Figure 1**.

A new cable was made up using 5 fairly thin wires, stripped from some multicore cable, and a length of 3 mm heatshrink sleeving, with a mini XLR 5 pin plug at one end and the old remote control pod on the other. This enables normal cable remote functions to be made via this ‘adapter cable’. The construction is illustrated in **Figure 2**.

Hand held transmitter

The circuit diagram of the hand held control box is given in **Figure 3**. The key component is a Holtek HT12A remote control encoder chip. The (condensed) datasheets of this interesting and versatile chip appear elsewhere in this magazine.

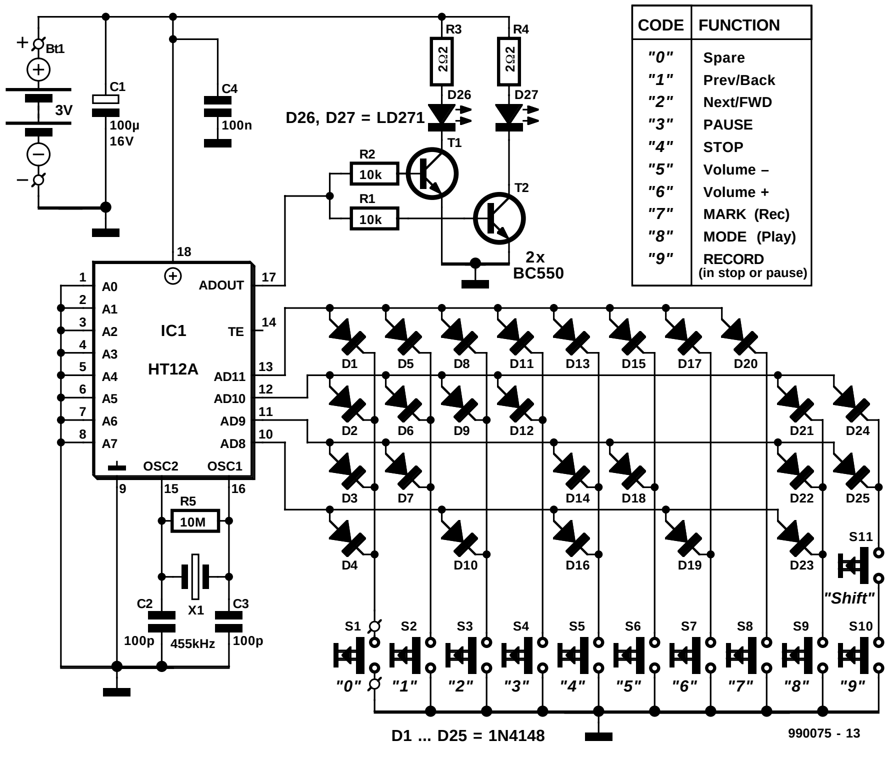
Any button press will place 0 volts on the selected diode(s) to pull down the data inputs of the encoder IC1. Any data input going low will ‘wake up’ the encoder chip, start the 455 kHz oscillator and the encoded data stream will be output from pin 17 (Dout) driving the two transistors and subsequently the two infrared sender diodes. A low value series resistor (2.2 Ω) enables the IR diodes to be driven with high current pulses, although the average current is only 10 mA per transistor. The large electrolytic capacitor C1 is essential to overcome the relatively high internal resistance of the two AA or AAA batteries.

The relation between pushbutton number, transmitted code and the associated MD recorder function is shown in tabular form in the circuit diagram. Note that pushbuttons S10 and S11 have to be pressed simultaneously to transmit a RECORD command. This is done to prevent inadvertent selection of the RECORD mode.

Receiver/decoder unit

The Sony MZ-R30 requires a number of specific resistance values to recognize certain functions selected via its remote input socket: The resistance values and associated functions are shown in **Table 1**. The code in the third row is the decimal value of the 4-bit data used by the IR encoder/decoders.

The use of type 4016 analogue switches (IC3, IC5,



CODE	FUNCTION
"0"	Spare
"1"	Prev/Back
"2"	Next/FWD
"3"	PAUSE
"4"	STOP
"5"	Volume –
"6"	Volume +
"7"	MARK (Rec)
"8"	MODE (Play)
"9"	RECORD (in stop or pause)

Figure 3. Circuit diagram of the hand held control.

4

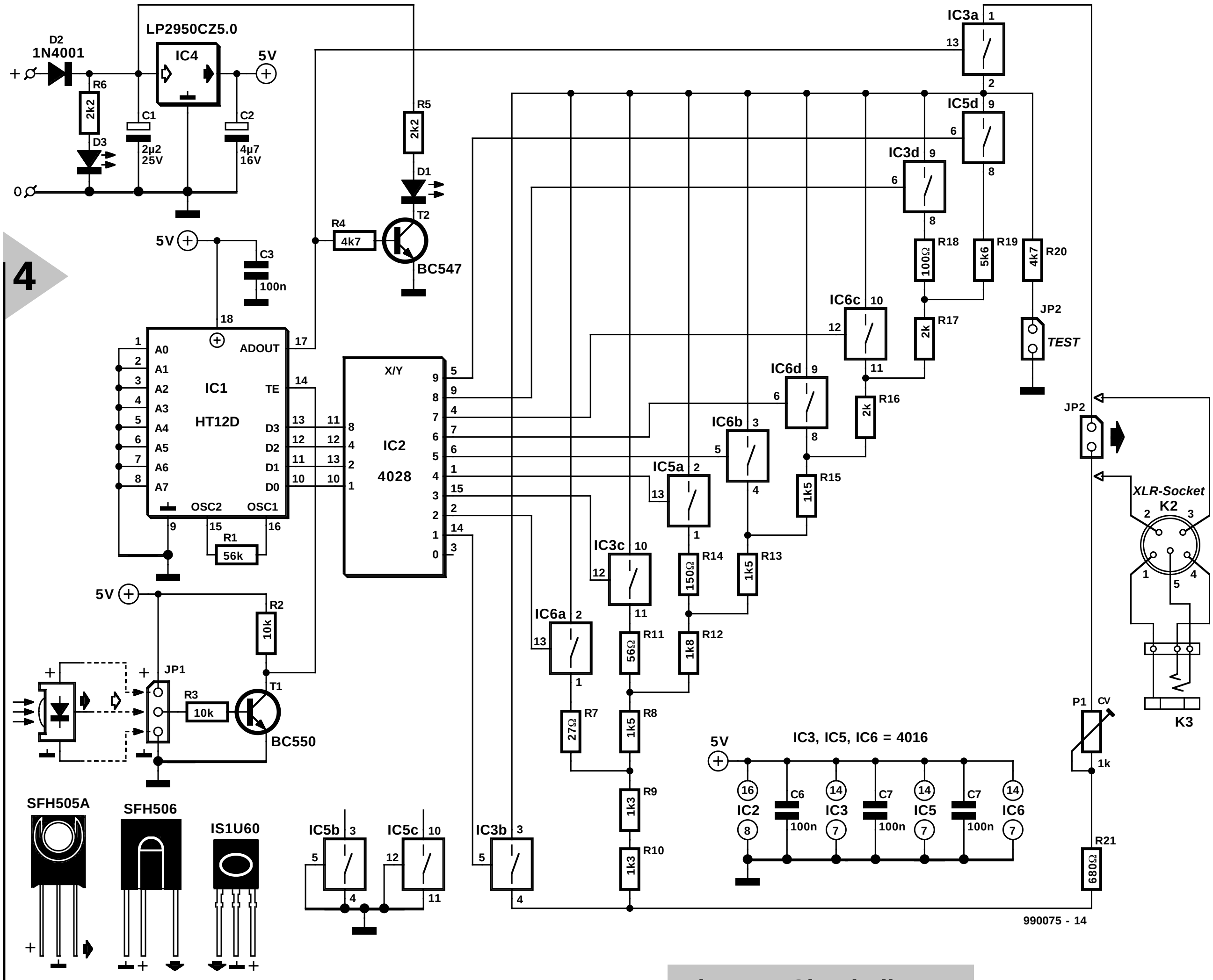
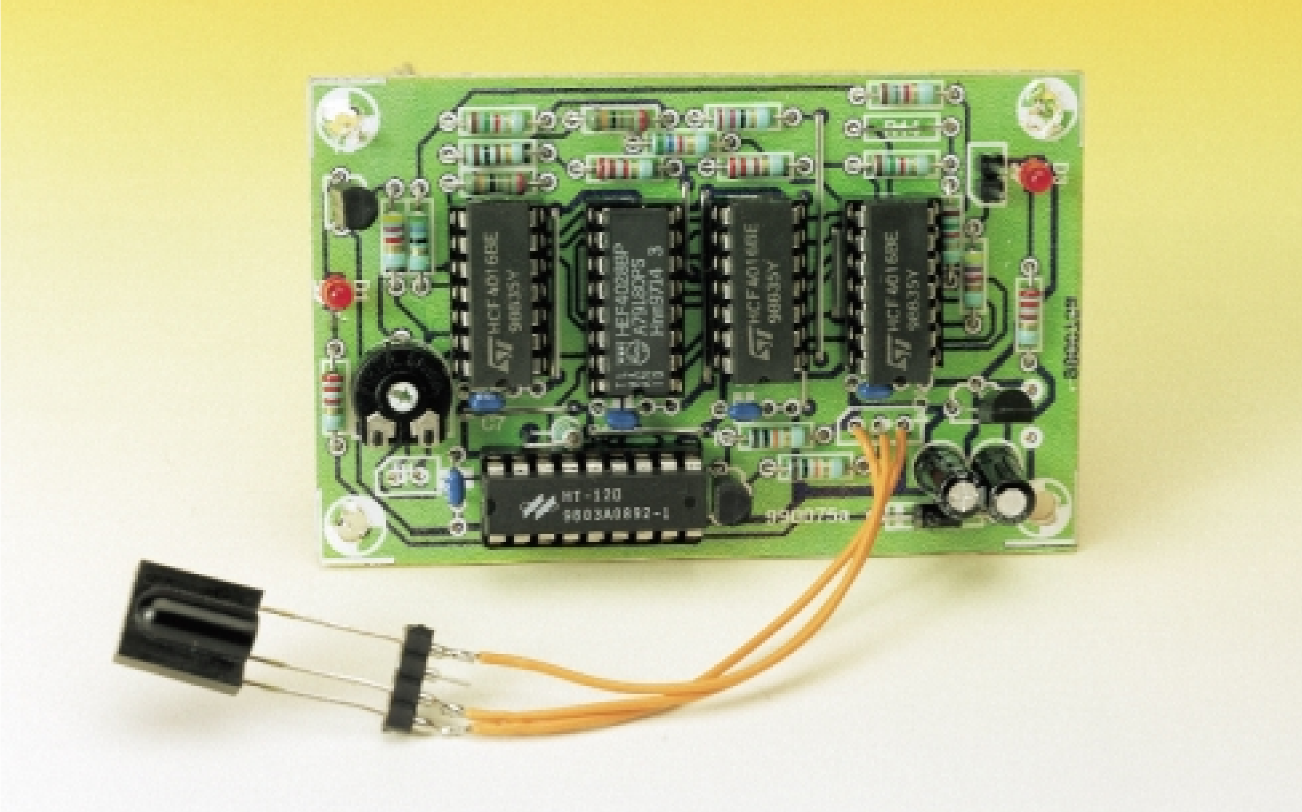


Figure 4. Circuit diagram of the remote control receiver/decoder.



IC6) in conjunction with a resistor network (R7-R20) offers a simple and cheap method of selecting the required resistance value.

A Sharp IS1U60 IR receiver IC detects the IR carrier and its output is inverted by T1 and fed to the HT12D decoder IC which latches the original 4 bits of data. This data is then presented to a BCD/Decimal decoder (IC2, 4028) which selects one of nine analogue switches. These in turn select the appropriate point on a resistor ladder to be used by the MZ-R30 for the selected function. The condensed datasheet of the HT12D IR decoder

COMPONENTS LIST

Receiver/decoder
(Board section 990075a)

Resistors:
R1 = 56kΩ
R2,R3 = 10kΩ
R4,R20 = 4kΩ7
R5,R6 = 2kΩ2
R7 = 27Ω
R8,R13,R15 = 1kΩ5
R9,R10 = 1kΩ3
R11 = 56Ω
R12 = 1kΩ8
R14 = 150Ω
R16,R17 = 2kΩ
R18 = 100Ω
R19 = 5kΩ6
R21 = 680Ω
P1 = 1kΩ preset H

Capacitors:
C1 = 2µF2 25V radial
C2 = 4µF7 16V radial
C3-C7 = 100nF

Semiconductors:
D1,D3 = low-current LED
D2 = 1N4001
T1,T2 = BC550
IC1 = HT12D (Holtek) (Maplin)
IC2 = 4028
IC3,IC5,IC6 = 4016
IC4 = LP2950-CZ5.0

Miscellaneous:
JP1 = SFH506-36

Transmitter
(Board section 990075b)

Resistors:
R1,R2 = 10kΩ
R3,R4 = 2Ω2
R5 = 10MΩ

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

Semiconductors:
D1-D25 = 1N4148
D26,D27 = LD271 or similar IR LED
T1,T2 = BC550
IC1 = HT12A (Holtek) (Maplin, Farnell)

Miscellaneous:
S1-S11 = pushbutton, PCB mount, Multimec or D6-R-RD
BT1 = 2 off AA or AAA penlight battery
X1 = ceramic resonator, 455kHz (e.g., SB455E) (Mainline, 0870 241 0810)

5

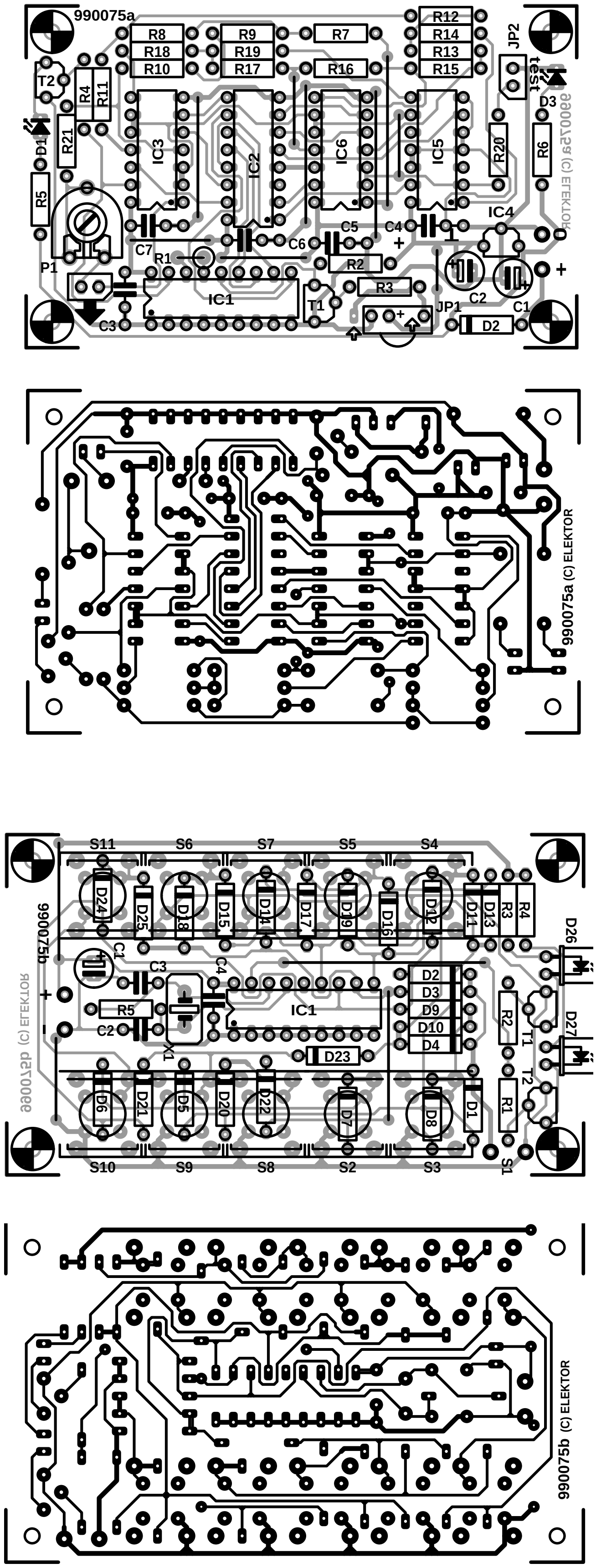
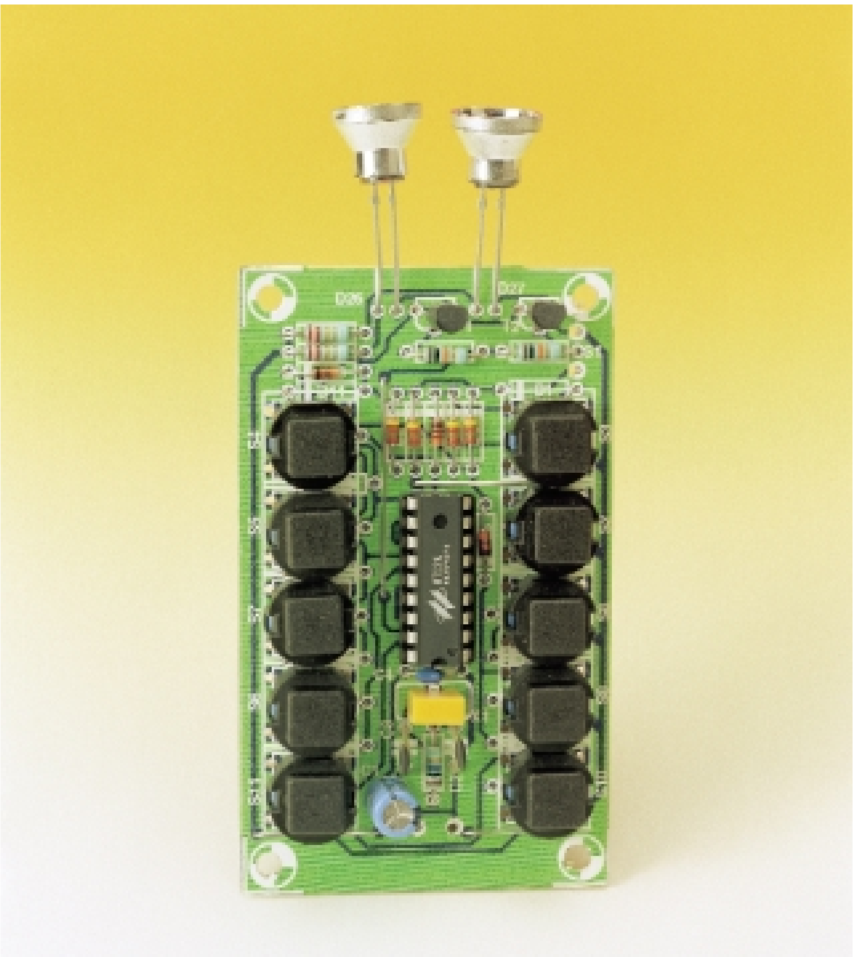


Figure 5. PCB copper track layouts and component overlays. Cut the board to separate the transmitter (below) and receiver (top) section.



appears elsewhere in this issue.

A common analogue switch (IC3a) controlled from the IR decoder (VT) pin is used to gate the output, since the 4-bit data is latched by the decoder and always selects the last switch that was used. The minimum selectable resistance (function “Preview/Back”) is 1000 Ω and to achieve this, a 1kΩ preset potentiometer, P1, is used to add approximately 700Ω in series with the 300Ω total resistance of the two analogue switches. A bright LED, D1, is also switched on at the same time to indicate correct operation.

The resistance value associated with a control code appears across JP2, the output of the circuit. A suggested connection to a mini-DIN socket is shown in the circuit diagram. Alternatively, you may want to use a mini XLR chassis plug. The pinout is then as follows:

Pin 1:	audio left
Pin 2:	pin 1 of IC3a
Pin 3:	preset P1
Pin 4:	audio right
Pin 5:	audio common

Alternatives to the IS1U60 include the Siemens SFH505A and SFH506. Their pin functions being different from the IS1U60, you have to pay attention to the way they are connected to the board.

The output of the remote control receiver is wired to a suitable miniature socket which is then connected to the remote control input of the MD30-RZ via a short cable.

The function of the test jumper, JP2, is discussed further on.

The receiver/decoder may be powered by just about any wall adaptor capable of supplying 8-12 V DC at about 100 mA.

Construction

The printed circuit board you will need to build this project is shown in **Figure 5**. The first thing to do is separate the receiver and transmitter sections with a jigsaw.

To keep cost as low as possible, these are single-sided circuit boards. They contain a few wire links which should be fitted before anything else.

The PCBs are easily stuffed using the parts list and the component overlay. Be sure to fit all polarized components the right way around, that is, diodes, LEDs, transistors, electrolytic capacitors and ICs. Although they are neither expensive nor hard to get (Maplin), the HT12 ICs are best fitted in IC sockets.

The transmitter board has a number of diodes fitted **at the solder side of the board**. The IR sender diodes may be fitted with reflector caps to boost their directivity.

The author fitted his version of the handheld control in a type HH1 plain box from Maplin. This had enough space to incorporate a holder for two alkaline rechargeable AAA cells.

Setting up

The only setting up required is to adjust the 1 kΩ preset to compensate for the resistance of the analogue switches. This is done as follows:

a) Install jumper JP2. This links R20, the 4.7 kΩ resistor from the common rail of the network, to 0 V.
b) Connect a DMM on a suitable resistance range to be able to measure 7,050 Ω, across the output pins 2 and 3 on the XLR connector.
c) Power up the receiver/decoder and select code 4 “Stop” on the remote hand held control.
d) Adjust preset P1 to give 7,050 Ω on the DVM.
e) Remove power, pull JP2 and disconnect the DVM.
f) Connect the receiver/decoder to the MZ-R30 and test all functions.

(990050-1)

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

Table 1. MiniDisc Walkman remote control codes

Function:	*Resistance:*	*Code:*
Preview/Back	*1,000 Ω*	*1*
Next/Forward	*3,627 Ω*	*2*
Pause	*5,156 Ω*	*3*
Stop	*7,050 Ω*	*4*
Volume (–)	*8,400 Ω*	*5*
Volume (+)	*9,900 Ω*	*6*
Mark	*11,900 Ω*	*7*
Mode	*14,000 Ω*	*8*
Record	*19,500 Ω*	*9*

parameter box for MIDI software

convenient software synthesizer operation

Using a keyboard and mouse to operate the innumerable knobs and sliders of a PC mixing board can turn playing around with a sound synthesizer into an agonizing experience. With the MIDI parameter box it's a different story!

Design by T. Klose

Almost every PC these days has a sound card. However, most PC users employ it only to reproduce operating system sounds, music from audio CDs and sound effects for games and presentations. They thus use only the standard features of the software for the sound card, even though there are outstanding programs available that allow even non-musicians to exploit the manifold features of modern sound cards. These are sound synthesizer programs.

Such synthesizers are sometimes even included in the software packages that come with the better quality sound cards. However, there are also separate programs, such as *Generator* from Native Instruments or *Rebirth* from Propellerhead. These programs simulate the functions and operations of a real synthesizer using a screen full of sliders and knobs that are controlled by the mouse. They represent a sort of Gordian knot for anyone who wants to do more than adjust a single slider.

The circuit described in this article makes working with a software synthesizer considerably easier. It utilises the ability of a software synthesizer to receive MIDI codes and to use these codes to drive certain controllers. The hardware of the MIDI parameter box can thus be kept very simple. A microcontroller reads the positions of eight standard potentiometers in turn, via an 8-channel A/D converter. If one or more of the potentiometer positions is changed, the microcontroller sends this information in MIDI format to the MIDI input of the sound card. The software synthesizer translates the MIDI codes into new settings for the controller in question.

Layers and super-layers

The somewhat nebulous term 'layer' often crops up in connection with MIDI. Layers are actually nothing more than groups of eight synthesizer functions, which correspond to the eight potentiometers of the MIDI parameter box using a sort of multiple allocation.

1

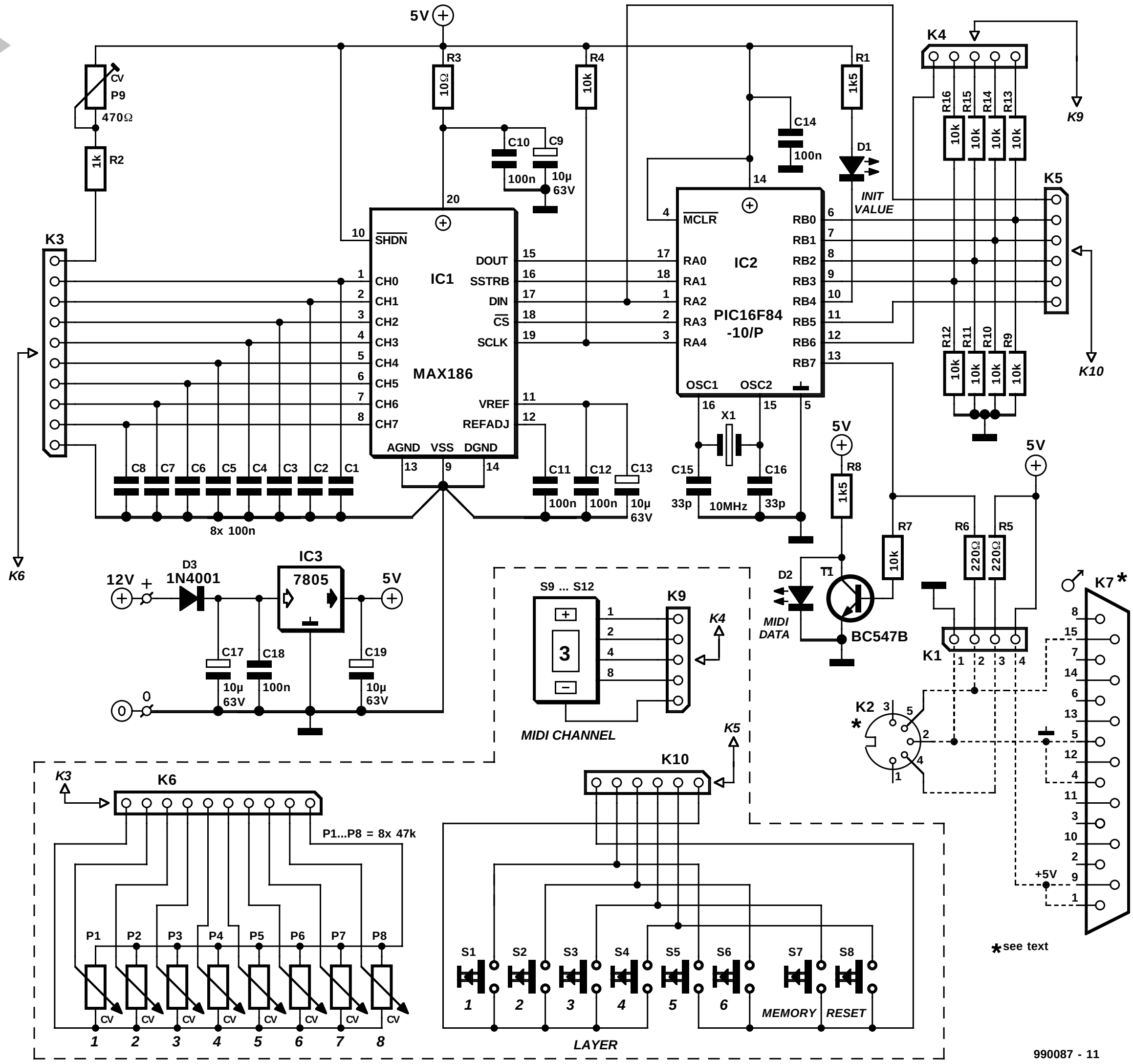


Figure 1. The microcontroller reads the status of the switches and of each of the eight potentiometers in turn.

The parameter box has pushbutton switches that can select one of six layers. The MIDI codes sent by each potentiometer (or the controller) thus vary according to the layer that is selected.

The assignment of layers to specific functions is not the same for all controllers. Instead, controllers can be divided into different groups. The various types of assignments are referred to as super-layers. The MIDI parameter box knows the three most commonly used groups and adapts the MIDI codes to their specific needs. **Table 1** summarises the contents of the six layers within the three super-layers.

Hardware

The most important elements of the hardware, which have already been mentioned, can easily be recognised in the schematic diagram shown in **Figure 1**. The eight potentiometers are connected to channels 0 through 7 of the A/D converter IC1. The MAX186 IC, which has already been used in a number of Elektor projects, has an interface to the microcontroller (IC2). This interface carries the output data from the D/A converter (DOUT) and the clock (SCLK), as well as the settings for the multiplexer (DIN). The converter is controlled via the SSRB and $\overline{\text{CS}}$ leads, synchronous to SCLK.

The microcontroller is a type PIC16F84 IC that is clocked at 10 MHz. In addition to the potentiometer positions, it also reads the layer selection switches (S1–S6), the MIDI channel switches ((S9–S12) and two other pushbutton switches, MEMO and RESET. **Table 2** describes the meanings and uses of all of the switches.

MIDI communication with the sound card takes place via port lead RB7. The MIDI signal can be visually checked via the (blinking) LED D2. There is a good reason why two connection options are shown in the drawing. Actually, the MIDI parameter box should only be connected to the sound card via a true, optically isolated MIDI interface. Such an interface is part of the AWE-64 package, for instance, but it cannot be ordered separately. You can either buy one for around £15 in a computer shop or copy one of the numerous Elektor designs (such as the MIDI interface in the 1995 Summer Circuits issue). A less elegant option, but one that can conceivably be used if no other MIDI devices are connected, is to connect the MIDI parameter box directly to the 15-pin joystick interface, which also has a MIDI input. The 220Ω resistor in the data line protects against short circuits if this alternative is used. This type of connection has one advantage, which is that the operating power can be drawn from the joystick port, so that D3, C17, C18 and IC3 are not needed.

This brings us to the power supply. An external power supply is obligatory with ‘real’ MIDI interfaces, in order to ensure the electrical isolation of the PC and the MIDI equipment. Only capacitors C17 through C19 and the voltage regulator IC3 are needed to provide a sufficiently stable +5 V. Power can be

Table 1. Contents of the six layers within the three super-layers.								
Layer	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**
super-layer 1 *(AWE, EMU8000 and EMU10k-1 synthesizers)*								
layer 1	*Controller 10 – 17*							
layer 2	*Controller 18 – 1F*							
layer 3	*volume*	*pan*	*expression*	*modulation*	*LP cutoff*	*LP reson.*	*chorus*	*reverb*
layer 4	*LFO1 delay*	*LFO1 freq.*	*LFO1 pitch*	*LFO1 vol.*	*LFO2 delay*	*LFO2 freq.*	*LFO2 pitch*	*LFO2 vol.*
layer 5	*env1 delay*	*env1 attack*	*env1 hold*	*env1 decay*	*env1 sustain*	*env1 release*	*env1 pitch*	*env1 cutoff*
layer 6	*env2 delay*	*env2 attack*	*env2 hold*	*env2 decay*	*env2 sustain*	*env2 release*	*-, -*	
super-layer 2 *(XG synthesizer)*								
layer 1	*Controller 10 – 17*							
layer 2	*Controller 18 – 1F*							
layer 3	*volume*	*pan*	*expression*	*modulation*	*portamento*	*reverb*	*chorus*	*variation*
layer 4	*attack*	*decay*	*release*	*vib. delay*	*vib. rate*	*vib. depth*	*cutoff*	*resonance*
layer 5	*pitch init*	*pitch attack*	*pitch rel.*	*p.r. time*	*vel. lim. L*	*vel. lim. H*	*-, -*	
layer 6	*pitch ben.*	*filter ben.*	*amplitude ben.*	*LFO PMOD ben.*	*FMOD D ben.*	*AMOD*		
super-layer 3 *(software synthesizer)*								
layer 1	*Controller 00 – 07*							
layer 2	*Controller 08 – 0F*							
layer 3	*Controller 10 – 17*							
layer 4	*Controller 18 – 1F*							
layer 5	*Controller 20 – 27*							
layer 6	*Controller 28 – 2F*							

provided by a simple 12 V mains adapter. Diode D2 provides protection against a reverse-polarity connection.

AS SMALL AS POSSIBLE

To make the construction of the MIDI parameter box as convenient as possible, we have designed a printed circuit board that is the size of a pack of cigarettes, as shown in **Figure 2**. It is available from Readers Services (order number **990087-1**). You shouldn't experience any problems mounting all the components, which takes around half an hour. Pay attention to the orientation of the electrolytic capacitors, the LEDs and the ICs, and don't forget the wire bridge next to K5. After this you can prepare the control panel. The potentiometers and switches can be glued to the front panel in a reasonable arrangement, such as that shown in **Figure 3**, and wired using flatcable. You can also mount these components

Table 2. Meanings and uses of all switches.		
Component	**Designation**	**Meaning**
LED D2	*MIDI Message*	*Message Blinks when a MIDI message is sent via the MIDO OUT port.* *Also blinks when a potentiometer is set exactly between two quantization levels.*
LED D1	*Init Value*	*Blinks after the power is switched on to indicate that one of the three super-layers must be selected using switches S1 – S3.* *Illuminated when the initial value is set for the potentiometer that was last rotated.*
Switches S1 – S4	*MIDI-Channel*	*These four binary-coded switches select the MIDI channel.*
Potentiometers P1 – P8	*Fader*	*These potentiometers are used to set the MIDI values.*
Pushbuttons S1 – S6	*Layer*	*These switches select layers 1 through 6.*
Pushbutton S7	*Memo*	*Save the last modified value in the current layer.*
Pushbutton S8	*Reset*	*Overwrite the current value with the predefined initialization value.*

COMPONENTS LIST

Resistors:
R1,R8 = 1kΩ5
R2 = 1kΩ
R3 = 10Ω
R4,R7,R9-R16 = 10kΩ
R5,R6 = 220Ω
P1-P8 = 47kΩ linear potentiometer
P9 = 470Ω preset Hi

Capacitors:
C1-C8,C10,C11,C12,C14,C18 = 100nF
C9,C13,C17,C19 = 10µF 63V radial
C15,C16 = 33pF

Semiconductors:
D1,D2 = LED, high efficiency
D3 = 1N4001
T1 = BC547B
IC1 = MAX186BEPP
IC2 = PIC16F84-10/P (programmed, order code **996521-1**)
IC3 = 7805

Miscellaneous:
K1 = 4-way SIL connector
K2 = 2 PCB solder pins
K3,K6 = 10-way SIL connector
K4,K9 = 5-way SIL connector
K5,K10 = 6-way SIL connector
K7 = 15-way Sub-D plug, chassis mount (see text)
K8 = 5-way DIN socket, chassis mount, 180° (see text)
K11 = mains adaptor socket
S1-S8 = pushbutton, 1 make contact
S9-S12 = on/off switch
X1 = 10MHz quartz crystal
Enclosure, e.g. Teko 363 (216x130x77mm)
PCB, order code **990087-1**

on a piece of prototyping board and wire them point-to-point. Of course, you can also design a ‘real’ circuit board. You should dress the flat cables such that the unit can later be built into an enclosure.

TESTING

In order to thoroughly test the MIDI parameter box, you absolutely need a MIDI monitor with a MIDI-through option for the PC, so that you can observe the transmitted MIDI data on the monitor and properly calibrate the potentiometers. The text box contain more information about suitable MIDI monitors. After a visual inspection of the soldering, connect the parts together and cable the unit to the PC. Then switch everything on and start the MIDI monitor. All the stored values in a virgin PIC are set to FFh, so they must be set to valid MIDI protocol values by pressing the Reset button. Next select super-layer 1 and layer 1 (the default layer) by pressing S1 twice.

Now comes the moment of truth. When the potentiometers are rotated, LED D2 should flash and the MIDI monitor should display control codes. These will have values ranging from 0 to 127.

2

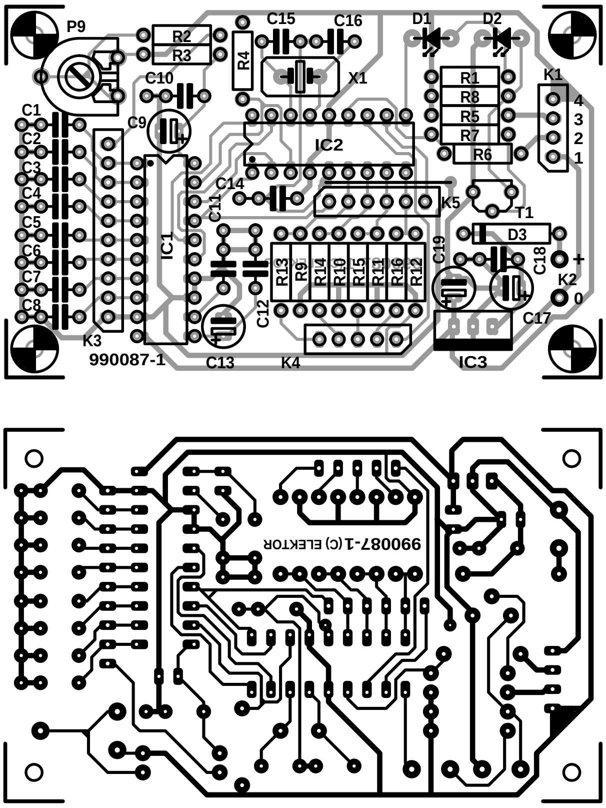
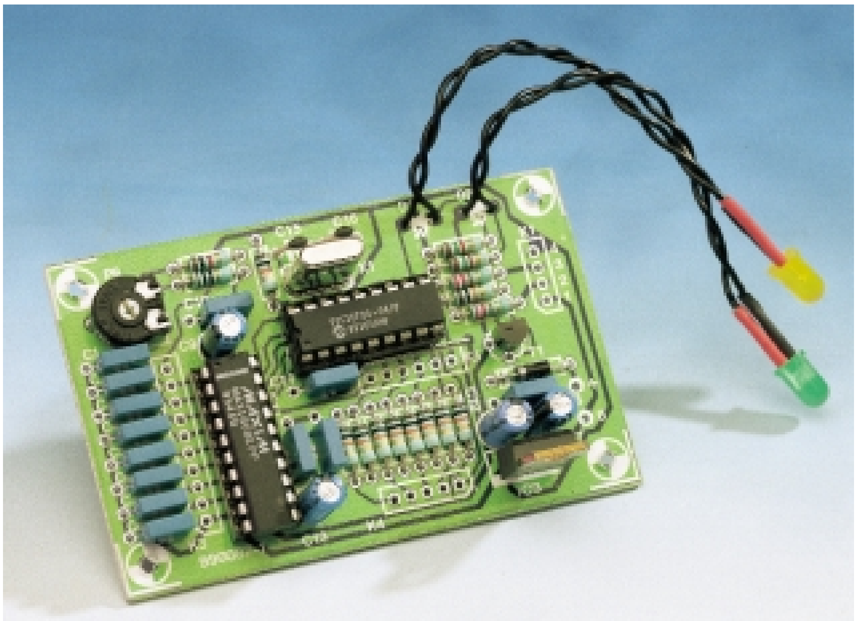


Figure 2. The MIDI parameter box can be build using this small printed circuit board.

If this does not happen, thoroughly check the circuit construction, the cabling and the settings of the MIDI monitor. If this doesn’t help, you can curse Windows or the sound card.

However, if the MIDI monitor displays the first MIDI events, then everything is in order. Trimpot P9, by the way, can also be used for calibration to adjust actual range of the MIDI values to 0 through 127.

Verify that the MIDI channel is changed by S9–S12 (binary), and that changing the layer works properly. When the layer is changed, the last stored values for the potentiometers are always output via the MIDI interface. The advantage of this is that the parameters of the synthesizer or the sound card are reset to their last stored values. If for example you change the volume in layer 3, change to a different layer and some time later return to layer 3, the volume will be restored to its original level. If you want to avoid this, all you have to do is to press the Memo button before



3

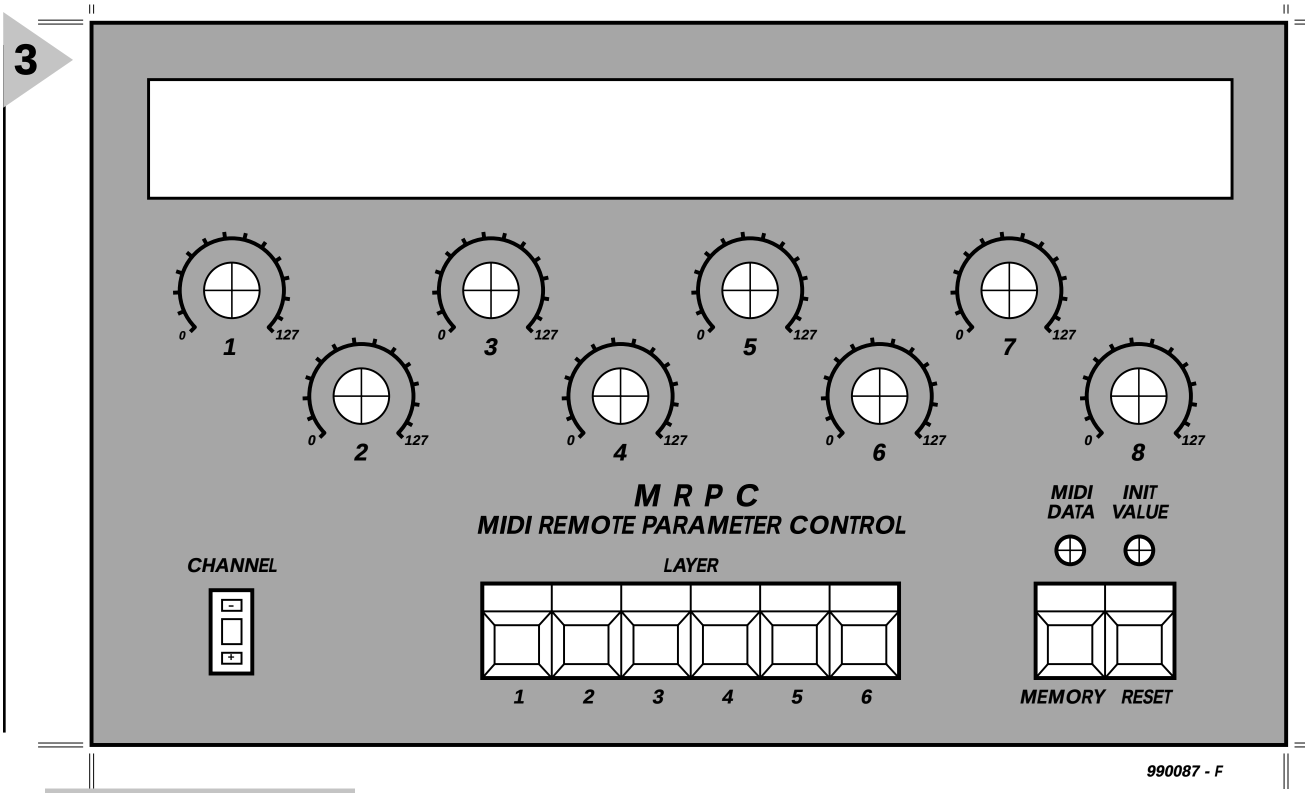


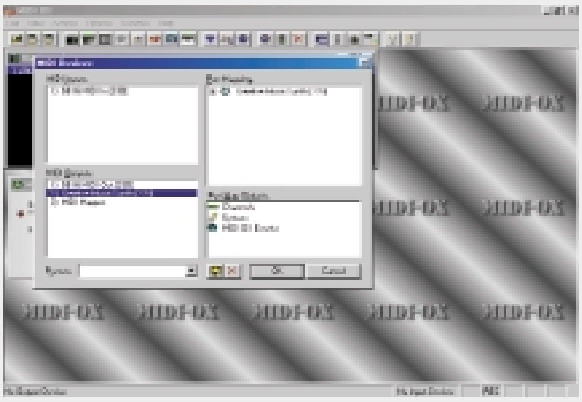
Figure 3. A reasonable arrangement for the control elements on the front panel.

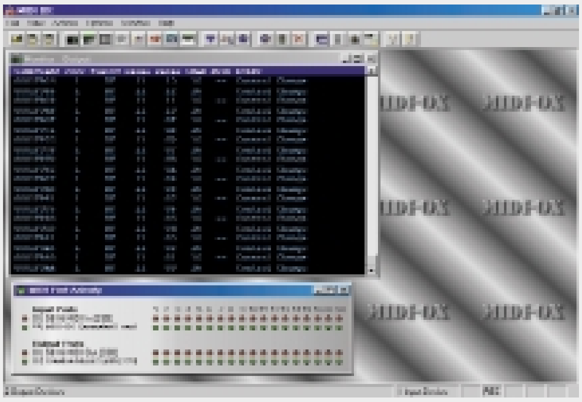
changing the layer. In this case only the values that have changed since the last layer change are stored.

(990087-1)

Text editing
(German original): R. Gerstendorf
Design editing: K. Walraven

MIDI monitors





There is a whole series of MIDI monitors that can be used with a PC. An outstandingly suitable program is MIDI-OX, for which a beta version is available for free on the Internet at www.members.xoom.com/_XOOM/MIDIOX/moxbeta.htm. After installing and starting the program, you must first select the MIDI devices, either via the menu Options/MIDI Devices or by clicking on the dark blue button with the five-pin DIN connector (see ***Figure A****). On this PC, the MIDI input and output of the SoundBlaster SB16 are active.*

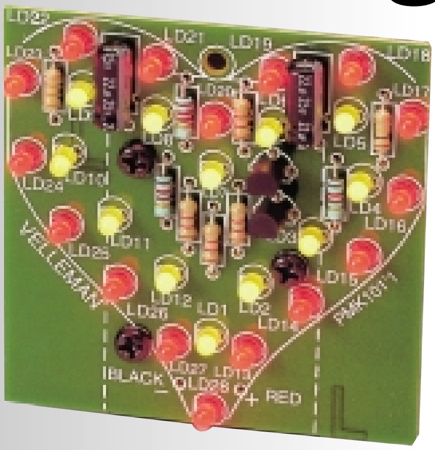
The MIDI Port Activity window shown in ***Figure B*** *appears if you press the bright green DIN icon in the second group of buttons. Each MIDI input and output gets its own row of 'LEDs', so that it is clear which channel is active.*

The content of the transmitted data appears in the Monitors Output window. The first column shows the time when the MIDI message occurred (as noted by MIDI-OX), and the second column indicates the MIDI port via which the message arrived (in this case, Port 1 via SB16 MIDI). The following byte, 0BFh, consists of two parts: a Control Change (indicated by the 'B') and the MIDI channel number (indicated by the 'F', which corresponds to MIDI channel 16). DATA1 shows the controller number (0 – 127 in this case) and DATA2 shows the assigned value. Just as with DATA1, only the lower seven bits are used, so that the values range from 0 to 127. CHAN shows the MIDI channel once again. Note that MIDI officially uses channel numbers 1 through 16, but many programs display 0 through 15. When a different super-layer is active or the layer is changed, the messages displayed on the monitor also change.

If you want to learn more background information regarding MIDI, you can find an adequate amount of literature on the Internet. One example is Eddies Home – MIDI-RPN and NRPN (http://members.delosnet.com.tlc/nrpn.htm).

MIDI-OX is especially well suited to checking equipment functions. If you want to go deeper into the matter, you can try easy-to-use and powerful programs such as Generator from Native Instruments or Rebirth from Propellerhead. Free demo versions of both programs are available. These can be used for only a very short time and have no save functions, but they are an outstanding choice for just playing around with the MIDI parameter box.

flashing LED sweetheart



an original Valentine present

A small effort and an even smaller outlay is required to throw an electronic gadget together that's sure to make a great gift for Valentine's Day. Belgium-based Velleman are suppliers of kits that enable this type of circuit to be built by the masses. We decided to try out one of their kits.

Most of you, we are convinced, would avow to being pretty seriously involved in electronics, be it as a hobby or professionally. Sometimes, too, you may get the feeling that it's all getting a bit too serious. Typically, our readers are busy working on practical applications of published circuits, or tweaking the specs. They will rave on about distortion, signal/noise ratios, or memory capacity, painstakingly seeking ways to achieve improvements no better than tenths of a decibel or a few parts per million.

Riveting stuff, of course, but it makes you wonder sometimes if all this activity captures any of the sheer fun that can be had from the noble art of soldering. That is why we can not resist voicing a clear "start having fun again" note to those of you with a tendency of taking a high-brow look at the hobby. Electronics, we feel, need not always be useful, in fact there's no reason why it should not be amusing, playful and without pretension. With simple means, dozens of interesting projects can be built. So, why not build an original doorbell, a running lights unit or a flashing brooch? Just for the fun of it.

Quite possibly, people around you may value simple gadgets more than the latest high-spec complex devices, probably because to them the latter will forever remain big electronic mysteries.

A lovely flasher

Most Summer Circuits and December issues of *Elektor Electronics* contain at least a few 'playful' electronic circuits. Some kit suppliers go one step further, having discovered a market for such products.

In the recent Maplin catalogue, we came across a **Flashing LED Sweetheart** from kit supplier Velleman. This is presented as a lively little ornament consisting of red LEDs arranged in the shape of a heart. Flashing all the time, the LEDs beg for attention. Very decorative, we'd say, and highly suitable as a gift to someone close to your real heart.

Do not expect the latest design technology from this kit. As indicated by the circuit diagram, the circuit consists of little more than a bistable multivibrator built around two common or garden transistors, with seven rows of four LEDs each in their collector lines. LEDs LD1-LD12 form the inner heart, and LD13-LD25, the outer heart. Both hearts flash in alternate fashion to mimic the well-known pump action.

The high-efficiency LEDs operate 'sparsely' using series resistors R1-R7. The result is a current consumption of just 8 mA, enabling a 9-V PP3 battery to last for about 24 hours.

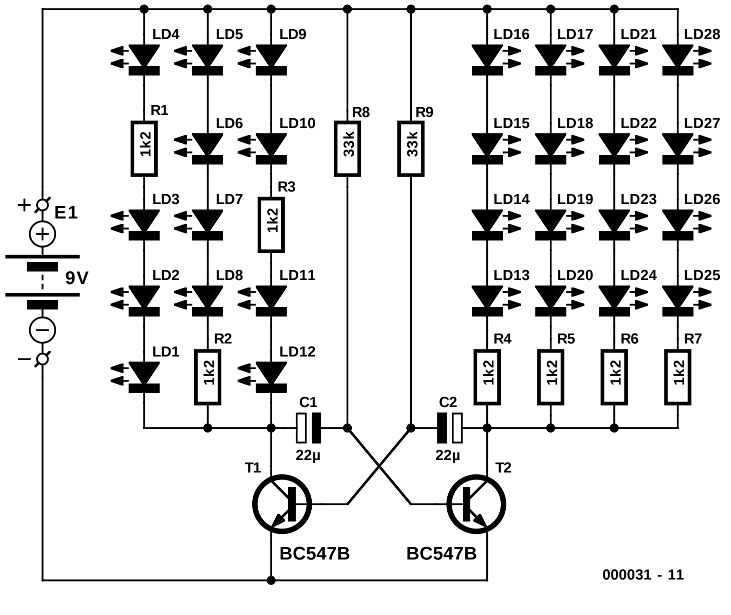
If you want to personalise the circuit, you may decide to use different colour LEDs for one of the hearts. Yellow LEDs, for example, do a fine job. However, in that case the series resistors may have to be decreased to about 820 Ω to compensate the higher voltage drop of yellow LEDs. If you do not change the resistors, the light intensity from yellow LEDs will be too low.

We do not know if Velleman offers any kind of warranty on this circuit. Reproducibility will not be a problem, we reckon, but a guarantee that the flashing sweetheart will succeed in actually conquering a heart will be impossible to obtain! However, at a price of just £4.99 it's well worth the effort.

The Maplin order code for this kit is VX75S.

(000031-1)

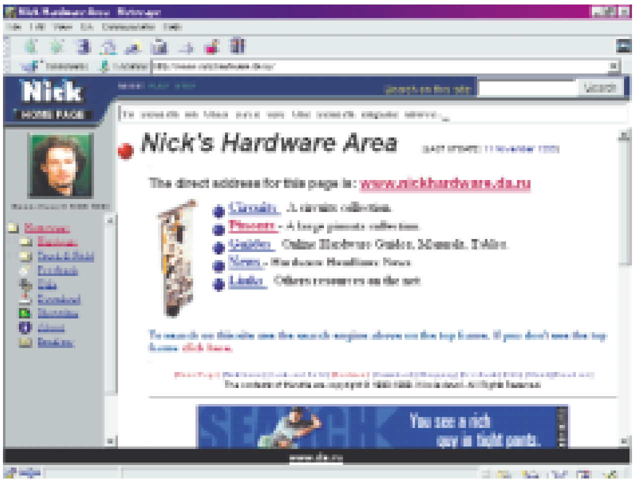
Text (Dutch original): S. van Rooij.



Nick's hardware area

InfoDesk on electronics

Electronics enthusiasts spend a lot of time just looking for all sorts of data: resistor colour codes, schematics for various applications, connector pinouts or perhaps a description of DIY circuit board etching. Well, Nick's website has it all nicely bundled. Practical and useful!

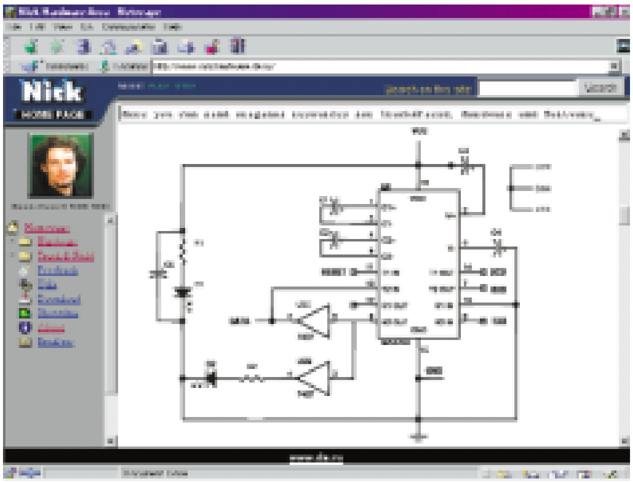


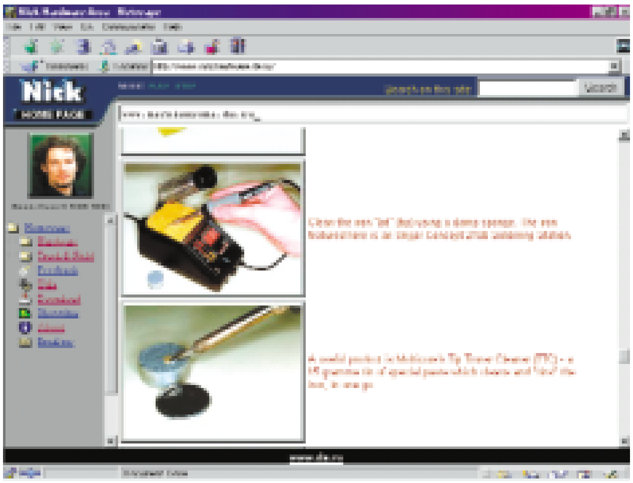
Nick's Hardware Area

The direct address for this page is: www.nickhardware.da.ru

- Circuits - A circuits collection.
- Pinouts - A large pinouts collection.
- Guides - Online Hardware Guides, Manuals, Tables.
- News - Hardware Headlines News
- Links - Others resources on the net

To search on this site use the search engine above on the top frame. If you don't see the top frame click here.





Nick's real name (we are informed) is Nicola Asuni and he lives on Sardinia. The website he built contains data that every electronics enthusiast or professional may need for everyday work. Starting from the homepage: (*http://www.nickhardware.da.ru*) you may choose between various departments (some of which having their own address):

- **Circuits** (*www.circuits.da.ru*)

This is a collection of small circuit diagrams for miscellaneous applications, subdivided into these subjects: infrared, telephone, sound, Smartcard and filters. The last subject is covered in great depth.

- **Pinouts** (*www.pinouts.da.ru*)

We reckon Nick has one of the largest overviews of connector pinout (i.e., connection data) on the entire Internet. Even if there are more massive sites, Nick's list is pretty impressive to say the least. A true galaxy of extremely common but also oddball connectors, cable links and adaptors may be found here.

- **Guides**

This department contains a number of manuals covering several aspects of home-brew electronics such as DIY circuit board etching, a course in soldering, SI and derived units, tables, passive component colour codes and a piece on electrical safety.

- **News**

A section covering the latest in electronics, but heavily computer-inclined. This page is frequently updated.

- **Links**

This section presents a list of other websites with useful information. The list is divided in computer/electronics links and links to various design standards.

On Nick's website you may also find lots of software downloads including C++ programs and program manuals written by Nick himself.

A search engine is available on the on this website if you are after information on a certain subject.

As an aside, Nick is an active athlete and he is even a member of the Italian sprinters' team. Thic subject, too, is covered in some depth on his website.

Nick's hardware area is a valuable asset for everyone actively involved in electronics, both professionally and/or as a hobbyist. This address certainly deserves a prominent place in your Favourites folder!

(005013-1)

Text (Dutch original): H. Baggen

BASIC Stamp programming course (6)

Part 6: introducing the FSM concept

By Dennis Clark

45

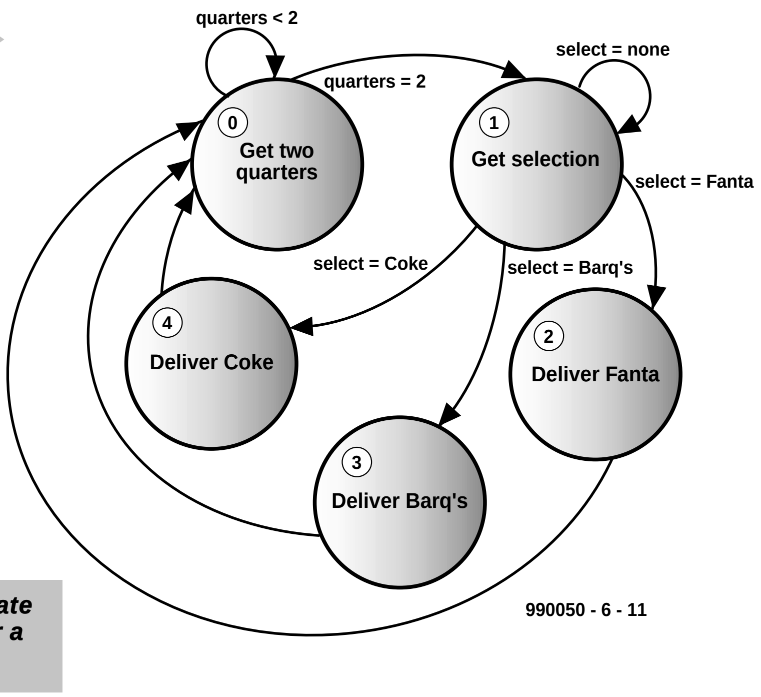


Figure 21. Finite State Machine design for a soda machine.

The Parallax Boe-Bot, or Board of Education Robot is a simple robot made from a Parallax Board of Education (BoE) mounted on an aluminium frame to which two Futaba hobby servos are mounted. These servos have been modified to provide continuous motion and have several speeds at which they will turn.

The BoE has a small prototyping area on which several experiments can be done. In order to realize the fascinating potential the BoE-Bot has to demonstrate robotic behaviours, you will need to know how to program it! In this and the following instalments I will explain how to make the robot move randomly, or purposely, how to seek out or avoid bright light and how to avoid objects. I will also explain how to program your BoE-Bot in such a manner that all of these functions seem to be operating at the same time.

Further, I will show you can get behaviours from your robot that you didn't even program in to it!

Linear execution vs. concurrent execution and the Finite State Machine

In a program, each statement is executed in sequential order, the next statement cannot start until the last one is done. When there is only one thread of logic running, this means that the program proceeds in a linear fashion, from start to finish. Concurrent execution is when there is more than one thread of logic running at what appears to be the same time. "How do I do that?", you may ask. In the Parallax Basic Stamp II processor you can do this by using independent sections of code, called *modules* or in our case, *subroutines* that will be called many times before their function is complete. In order to do that, you will need to keep track of where you are in your subroutine so that you know where to start up again when you return. One very good way to do this is called the *Finite State Machine* or *FSM* for short. Because there are only a few different actions you want your subroutine to execute, and it does actually complete its job at some point, it is a *finite* list. There are several forms of FSMs, this type of FSM is a hybrid we will use specifically for robotic programming. This *behaviour FSM* is a type of a state machine that returns no outputs, it merely changes state based on input and the current state. Each activity that the FSM engages in, is a *stat*e, unique in operation and distinct from all other states by its definition.

If this is difficult to understand, lets use a *sort of* real world application to explain FSMs, the soda machine. In order to keep our soda machine simple, we will create a pretty stupid machine with these abilities:

- takes only quarters
- needs two quarters to get a soda
- will not give you your money back
- does not give change, nor return money that isn't a quarter
- has Coke, Barq's Root Beer and Fanta Orange soda
- has an infinite amount of soda and never runs out

As you can see, we have eliminated all of the *exceptions* or error conditions that a normal soda machine could see in order to simplify this explanation — it's artificial for a reason; we're not designing soda machines. However, do pay attention to exceptions and error conditions when you are designing your own FSMs! **Figure 21** shows a graphical rendering of our soda machine FSM. The underlined numbers in each of the circles are the state number for that state. A line with an arrow denotes a transition from one state to another (the

Listing 9. Linear vs FSM programming

```
Linear based servo controller subroutine            FSM based servo controller subroutine

act:                                                act:
    for I = 1 to 10                                     if aDur > 0 then aDec
        pulsout LEFT,750                                    aDur = 5
        Pulsout RIGHT,750                                   pulsout LEFT,750
        pause 20                                            pulsout RIGHT,750
    next                                                    goto aDone
return                                              aDec:
                                                            aDur = aDur - 1

                                                    aDone:
                                                    return
```

arrow points the direction). If a transition line is labelled, that label is the result of the transition function and defines the condition required for that change of state. An unlabelled line is a transition that will always occur as soon as the function of that state is completed. The lines that loop back upon a state show iteration, or that the FSM remains in this state doing something until a terminal condition is reached, at which time a defined transition that is labelled will occur. Here we see that our soda machine FSM will remain in state 0 until two quarters have been given, at which point our FSM will transition to state 1. Here we will wait, looking at buttons until a selection is made. When a selection is made, our FSM will then transition to state 2, 3 or 4 depending on the selection made. From these terminal states, our FSM will immediately transition back to state 0 after completing. This is the general process of definition and representation for the FSMs that we will be using to define our BoE-Bot behaviours.

Remembering where you are in your subroutine is called *saving state* and is essential if you are to pick up where you left off when last this subroutine ran. Each state in our behaviour FSM will be executed when its subroutine is called and will exit the subroutine when that state is completed. Subsequent calls of that subroutine will execute the next correct state that is defined. Why is this useful? Let's look at two code snippets in **Listing 9** that show why this can make your whole program run faster. Both of these pieces of code operate the hobby servos that make your BoE robot move, don't worry about understanding them exactly, what this code does will be fully explained in due course. The one thing you must know is that a hobby servo requires that a pulse of 1 ms (millisecond) to 2 ms must be sent to each servo every 20 to 30 ms or the servo will not perform correctly. If you send it too often (say every 7 ms) the servo will jitter, if you send it too rarely (say every 50 ms) then the servo will stop. These pulses need to be repeated continuously, and regularly in order to operate correctly, a single pulse is not very useful to a servo.

The code on the left looks very simple and fast, but looks can be deceiving. The *pulsout* instructions are used to output a pulse of the needed width to turn the servos. Remember, this pulse needs to be repeated every 20 to 30 milliseconds (ms) in order for the servo to respond properly. Also, it needs to have several repetitions of this pulse for the motor to turn and keep running. The *pause* instruction will cause the Stamp II to pause for 20 ms, each of the pulses sent will be 2 microseconds * 750, or 1.5 ms. So, each pass through this *for/next* loop will take 3 ms + 20 ms = 23 ms at least, 10 times through the loop will take 230 ms! That is almost 1/4 of a second when nothing else can be done!

Now let's look at the code on the right that implements a two-state FSM to move the servos. You can see that our subroutine on the right does one of two operations at any given time. The first operation is to output the pulses to the servos and set the *aDur* variable. The second operation is to simply decrement the *aDur* variable. In either case, after the operation has been accomplished we exit the subroutine. Each of these operations will be defined as a *state* for the *act* behaviour.

We will get into more details on how to describe and design state machines for our robotic behaviours using examples and programs that you will write for your BoE-Bot in later instalments.

Returning to our code samples, let's figure the time spent in the subroutine on the left now. Since the Stamp II executes about 4000 lines of code a second this means that each instruction will take about 250 μs to execute. The pulsout instructions will obviously take 1.5 ms each to execute because that is the length of the pulse that is being sent. In state 1 it will take 3 ms for the pulsout instructions + 750 μs for the other three instructions, which equals 3.75 ms. In state two our second subroutine will take about 750 μs of processor time each time it is executed.

Instead of the 230 ms of processor time taken by the left subroutine, we now will take 5*750 μs + 3.75 ms = 7.5 ms of processor time total (we are taking 5 turns through it after the initial pulse outputs remember?) to accomplish the same purpose. If we only count a single 23 ms loop for each pass through the first subroutine, we will have saved 15.5 ms of processor time, which, at 4000 instructions per second amounts to 62 instructions that can be executed elsewhere and give us the exact same activity on our servo motors. If we take into account the full 230 ms time for the left loop we save over 226 ms which is a whopping 904 instructions!

But why is this important? A robot does not just wander aimless around in its environment, it usually has some task to accomplish. Whether it is searching for a fire to put out, trash to pick up or for another robot to attack, it is doing something else more important than just running its motors. When we use the motor driver routine on the left above, the robot is doing absolutely nothing but concentrating on running the motors for 230 ms. During this time it cannot look at a sensor, pick up trash or put out a fire. If it runs into something, it will just keep running into it until it is finished with that loop and can then do something else. Each of the other behaviours that we implement in our robot will be some activity that the robot will need to perform in a timely manner. It does us no good to detect an object to avoid *after* we have already run into it! Let's assume that our robot is running the following behaviours, listed in lowest priority to highest priority, to achieve some objective:

- Go North until home is found (chooses a direction to travel)
- Avoid hitting anything by using IR proximity detection (if something is a danger, choose another direction)
- If I hit something, back up and turn left (chooses *yet another* direction to go)
- Stop and beep when I am home (choose no direction at all, just stop)
- Select the highest priority direction

to go and call *act* to implement it

Many of these behaviours will tell the motors to perform some action; backing up, turning left, whatever. Each of these behaviours will need to refer to sensors in order to perform their actions. Each of these behaviours (using the system I am suggesting) will be Finite State Machines implemented in subroutines that will be called from within some main code loop (you will see some of these behaviours defined later on). In the motor driver routine *act* shown as the right side code snippet there is a variable *aDur* defined. When the *act* FSM is first called *aDur* is set to 5. This means that the *pulsout* instructions will be executed once, then the next 5 times the *act* subroutine is called it will do nothing but decrement *aDur* and exit. This means that it will spend as little time in the subroutine as possible. Why is this useful? It is useful because *act* only needs to send those *pulsout* actions once every 20 to 30 ms. Our robot can be looking at sensors and selecting the next motor action while it is waiting to send that next series of pulses out. In this way, we *use* the time it takes to read sensors and make decisions in those other four subroutines as the delay we *must* take between pulses we send to the servo motors instead of *wasting* that time with a *pause* instruction! In effect, this makes it look like everything is happening at the same time instead of one thing after the other. If we were to use a linear programming model instead of Finite State Machines to implement all of our behaviours and actions then we would not be able to look at the compass or check for an obstructing object until all 230 ms in the code snippet on the left had completed. In that time our robot might miss seeing the chair in front of it and collide with it before it gets a chance to change direction. There is nothing special about the number 5 chosen for *aDur* either, I used that number as a suggested starting point. In reality this number is chosen by trial-and-error to achieve the smoothest timing. I started with this number in my own robot and as I added behaviours I reduced it. For example, with four behaviours active I have my *act* routine set *aDur* to only 2.

When we implement all of our behaviours as FSMs this has the effect of interleaving the code that needs to be executed in each subroutine so that no one behaviour needs to wait until the prior behaviour completes in order to do at least some of the work that needs to be done in its own routine. This improves our robot's response time to its environment. In the case of the *act* subroutine above, this results in a smoother motor response and quicker reaction to obstacles and objectives.

Before using the Finite State Machine method of behaviour implementation I would notice that my robot would appear to hesitate longer and longer as I added more and more complex behaviours. This FSM method will all but eliminate this hesitation. It is more complex to design and code than linear programming, but the results, I feel, merit the complexity. Try programming your robots both linearly and using FSMs, I think you will agree that using Finite State Machines improves your robot's abilities and allows us to get as much out of our Stamp II as we can! Eventually a set of behaviours may become so complex that even using FSMs will not prevent some hesitation, but we can do much more using FSMs as our programming model rather than linear programming before that happens.

(990050-6)

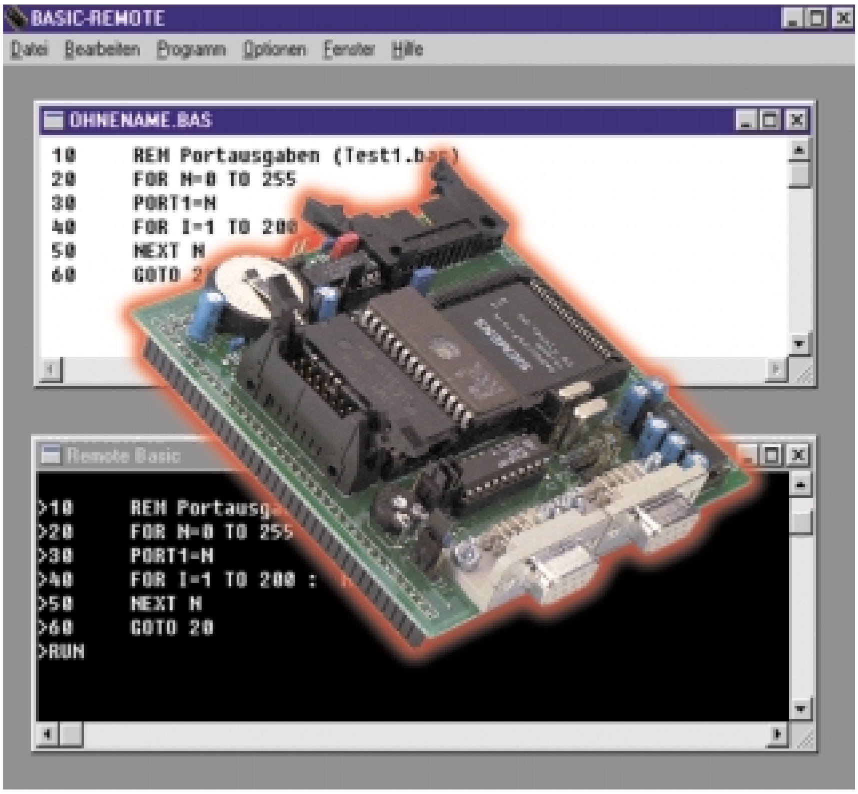
Next month we will continue with a method called Subsumptive Programming which will help us develop a step-by-step plan to implement robotic behaviour.

BASIC-537

a high-level language for the 80535/537

The most commonly used dialect of BASIC for microcontrollers, Intel’s 8052AH-BASIC, cannot be used as is with the new 8051 derivatives. This article presents a modified version of the interpreter that also supports the 80535/537 microcontrollers. These have been used in recent Elektor projects, such as the 80C537 Single-Board Computer and the 537 ‘Lite’ computer.

Design by B. Kainka



```
10      REM Portausgaben (Test1.ba
20      FOR N=0 TO 255
30      PORT1=N
40      FOR I=1 TO 20
50      NEXT N
60      GOTO 2
```

```
>10     REM Portausg
>20     FOR N=0 TO 255
>30     PORT1=N
>40     FOR I=1 TO 200 :
>50     NEXT N
>60     GOTO 20
>RUN
```

The Intel BASIC-52 interpreter was originally provided in the form of a mask-programmed microcontroller with the type designation 8052AH-BASIC. The interpreter automatically recognises the amount of available RAM and the baud rate of the connected terminal when the system starts up. There are also Autostart functions that allow a program to be run automatically on start-up. It is even possible to program an EPROM in the system, as long as you use the original Intel IC.

The 8052AH-BASIC IC is no longer produced. However, the program has been released for public use, so it can now be used legally in an EPROM. If you combine this with an 80C32 microprocessor, you have a low-power BASIC system. The only thing that you lose is the EPROM programming function of the original IC.

However, anyone who has tried to use a BASIC-52 interpreter in EPROM in an 80535 or 80537 system has experienced a bitter disappointment. The initialisation routine for the serial interface uses properties of the 8052 that have not been passed on to the 80535 in exactly the same form. The culprit is Timer 2, which is not compatible with the 8052.

The new processors can however be used if the reset routine is modified. The 8052-BASIC interpreter has provisions for a custom reset routine and for adding new user-developed instructions. We could thus write modifications that work with the new microcontrollers. In particular, the modified interpreter supports the Elektor 80537 Single Board Computer as well as the new 537 ‘Lite’ computer.

80C537 Single Board Computer

- *IC2 holds the BASIC-537 EPROM*
- *IC4 holds a 32-kB RAM*
- *IC5 and IC5 remain unused*
- *all 6 jumpers of J1A are connected to J1C (stand-alone mode)*

537-Light computer

- *J7:1—2 (12 MHz)*
- *J1A– J1C: 1—2 (stand-alone operation)*
- *switch S1: RUN*

Table 1. 80537 computer board settings

The modified interpreter, which we call BASIC-537, has the following features:

9600 baud data transfer rate
with a 12 MHz clock
32kB RAM
6 new instructions
direct A/D converter polling
LCD support (address FFCOh)
Autostart function

The BASIC-537 interpreter occupies the EPROM memory region from 0000h to 403Fh, and also requires 32 kB of RAM starting at address 0000h. A few jumper settings must be changed on the 80537 boards, as shown in **Table 1**, to ensure that the address regions are properly configured.

In principle, any available terminal emulator program running at 9600 baud can be used with the BASIC-537 interpreter. After the system is switched on, the interpreter announces itself with the following start-up message:

```
MCS-51(tm) BASIC V1.1 / ES537
READY
>
```

You can now enter lines of BASIC code in the usual manner. Each character is sent back to the terminal via the serial interface. Each time a Return character is received, the interpreter converts the complete line into its internal token format and stores the tokens in the RAM. The program can be started by typing *RUN*, and it can be interrupted at any time by pressing *Ctrl-C*. Typing *LIST* sends the complete program listing to the terminal, and typing *NEW* clears the memory.

Instead of a simple terminal emulator, you can also use the Windows program BASIC.EXE, which is found on the same diskette as the interpreter. This is a combined terminal emulation and editor program that is specially designed for the BASIC-535 interpreter. The diskette also holds an instruction summary file (BASIC52.HLP), an initialisation file (SBASIC.INI), the binary file of the EPROM contents (BASIC537.BIN) and sample programs with the extension BAS.

The program editor

When BASIC.EXE is first started, it is set up to use COM2. A different port can be selected via the *Options/RS232* menu. The selection is saved in the file SBASIC.INI when you quit the program.

When the microcontroller board is switched on, the BASIC-537 interpreter sends its start-up message. You can then enter the first program and run it. The sample program *Test1.bas* outputs data to port P1 and uses the original BASIC-52 instruction *PORT1*:

```
10 REM Port outputs
20 FOR N=0 TO 255
30 PORT1=N
40 FOR I=1 TO 200 : NEXT I
50 NEXT N
60 GOTO 20
RUN
```

BASIC.EXE has two text windows. A text editor with many editing functions runs in the upper window. The lower window displays all the characters received from the microcontroller. Every time that <Return> is pressed in the editor, the complete line is sent to the microcontroller.

The echoed characters appear in the terminal window. You can edit each line as much as you wish before it is transferred. This also applies to lines that appear earlier on in the text and that you want to modify after the fact. In the terminal window, you can see whether each line has been accepted or has caused an error message to be returned. Alternatively, you can activate the terminal window and use it for entering text, which will be transferred a character at a time.

The program's special functions can be called up using the *Program* menu. These are the *RUN, LIST, BREAK, CONTINUE* and *NEW* instructions for the interpreter in the microcontroller, and *CLS*, which clears the terminal window. You can use *LIST* to fetch a finished program and display it in the editor window, and then to save it to disk.

Naturally, instructions such as *NEW, LIST, RUN* and so on can be entered directly from the keyboard, the same as program lines. For testing, direct queries such as *PRINT PORT1* can be used. A special feature of the overall system is that code lines of a program that is already in the microcontroller are not overwritten by new lines unless the new lines have the same line numbers. This means that you can stock a collection of subroutines that can be downloaded as needed. In order to start from scratch, enter *NEW* before downloading a program.

You can load a program via the *File/Open* menu. First you should use *Program/Break* to stop any program that is already running, and then use *Program/New* to delete it. The text of the program will appear in a new editor window, and in the terminal window as well if the interpreter was ready. Several editor windows may be open simultaneously, which allows program sections to be copied back and forth.

Instruction summary

The most important instructions and functions of BASIC-537 are listed in **Table 2**. Numerous special functions have been adapted to the environment of the microcontroller. Some of the original BASIC-52 instructions cannot be used with the extended version and are not shown. Four new instructions have been specially added for the 80537.

The measurement range of the twelve possible analogue inputs is 0 to +5 V, but this can be modified within wide limits using the instruction *DAPR*, which writes a parameter byte to the processor register with the same name. A one-byte control parameter can be specified for each measurement. Its two 4-bit nibbles set the upper and lower limits of the range. Bits 0–3 set the lower limit in units of (5 V/16), and bits 4–7 set the upper limit in the same way. These two reference voltages must have a minimum separation of 1.25 V. For example, a parameter value of 84h sets the measurement range to 1.25–2.5 V. The value 00h is an exception; it sets the range to 0–5 V.

The instructions *SFR* and *WrSFR* allow all of the special hardware functions of the 80C537 to be used. They provide access to both serial ports, the timer, all supplementary ports and so on. Providing free access to all special-function registers presents a special challenge to an interpreter, since all 8051 derivative ICs allow only direct addressing (via an instruction such as `mov 90h, A`). The *SFR* address thus cannot be passed via another register. A compiler has no problem with this, since it hard codes the address, but an interpreter must be able to pass the address. This difficulty is probably the reason why the original BASIC-52 interpreter allowed free access to the internal RAM, the external RAM and even the program region, but not to the special function registers. Since a 32-kB EPROM has enough room for the necessary extensions, an indirect approach can be used. There are two small subroutines for reading and writing every address in the range from 128 to 255. When the interpreter encounters an *SFR* or *WrSFR* instruction, it calculates a jump address that leads to the proper subroutine. The extended interpreter is thus prepared for microcontrollers that haven't even been invented yet.

The text file *Sample Programs*, which is also located on the diskette, contains programs that illustrate the use of the specific instructions of the interpreter.

LCD control

BASIC-52 allows serial interface outputs to be redirected to a different output routine. This capability is used in the BASIC-537 interpreter to drive a liquid-crystal display. The display is largely addressed using normal instructions. Only the instruction for initialising the LCD (*LCDINIT*) and the instruction for setting the character location (*CURSOR*) have been added.

The new 537 'Lite' computer (January 2000) has an interface for an LCD module. This is located in the address space starting with address FFCOh. The display is initialised using the new instruction *LCDINIT*. After this, there are several ways to output characters

Table 2. Summary of BASIC-537 instructions

Instructions

RUN Ctrl-C CONT LIST NEW

Operators

*+ - / * = > >= < <= <> .AND. .OR. .XOR. ABS() NOT() INT() SGN() SQR() RND LOG() EXP() SIN() COS() TAN() ATN()*

Statements

CALL DATA READ RESTORE DIM DO-WHILE DO-UNTIL END FOR-TO-STEP NEXT GOSUB ON-GOTO ON-GOSUB IF-THEN-ELSE INPUT LET ONERR PRINT REM STOP

Extended BASIC-537 Statements

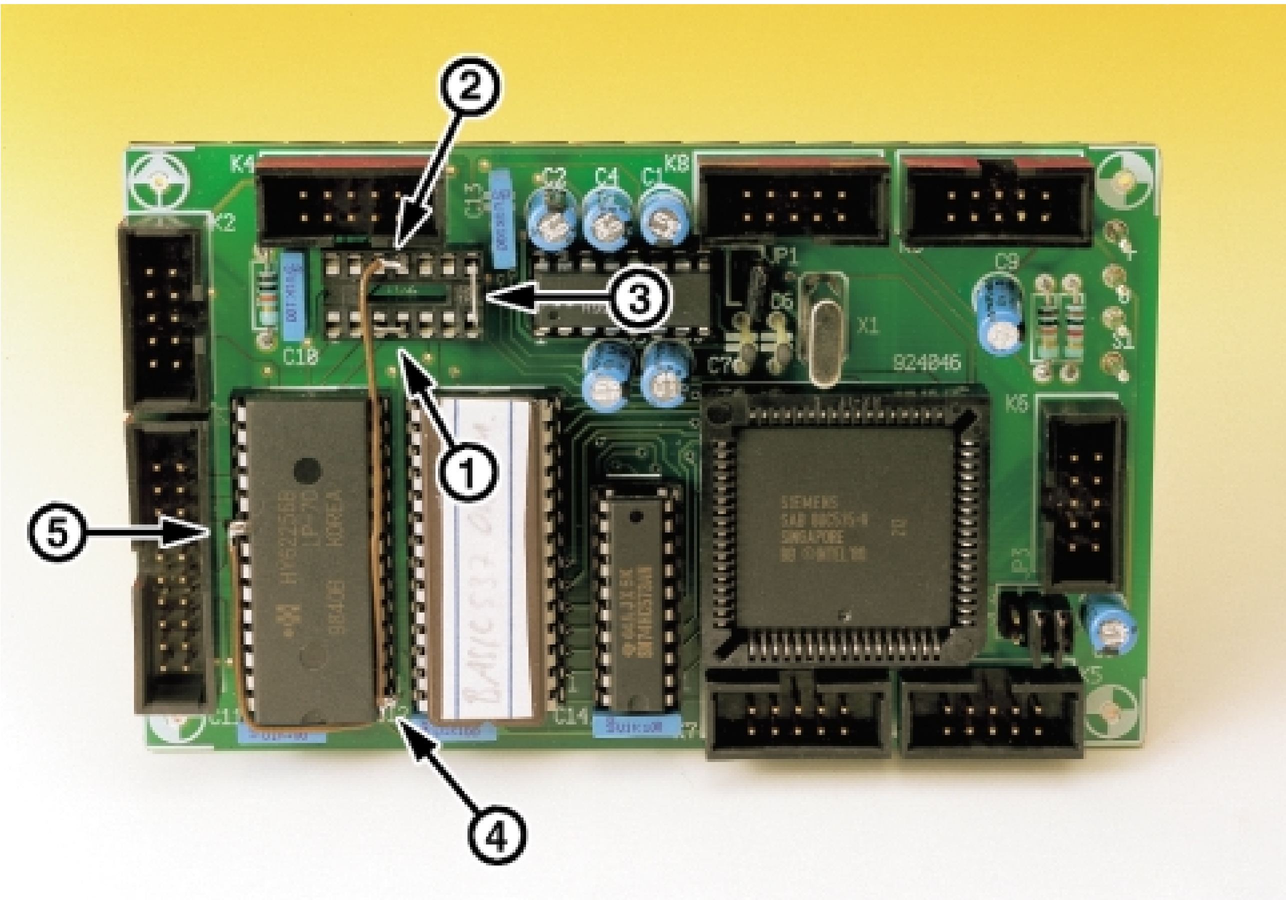
ONEX1	*subroutine call following Interrupt 1*
ONTIME	*timer interrupt call*
RETI	*end of an interrupt subroutine*
PH0., PH1.	*output a hexadecimal number with/without leading zeros*
PUSH, POP	*move data to/from the argument stack*
STRING	*reserve memory for a text string*
IDLE	*wait for interrupt*

Special function operators and system variables

CBY() DBY() XBY() GET IE IP PCON T2CON TCON TIME TIMER0 TIMER2 PI XTAL MTOP LEN FREE

80C537 I/O handling

AD (channel)	*[read-only]*	*C = AD (0)*	*analogue input (0 – 5V) at the AD0 input*
		C = AD (11)	*analogue input (0 – 5V) at the AD11 input*
DAPR	*[write-only]*	*DAPR = 128*	*switch the measurement range to 2.5 V*
SFR	*[read-only]*	*print SFR (90H)*	*reading special function registers*
WrSFR	*[write-only]*	*WrSFR 90H,255*	*writing special function registers*



to the display. The desired character position can be specified using the *CURSOR* instruction. The *PRINT@* instruction writes directly to the display. A sample program is available on the diskette or from the Elektor website.

A program can send its outputs to the terminal emulator using *PRINT*, or to the LCD using *PRINT@*, as desired. All output data can be formatted with the *USING* instruction. However, the line feed function has no effect on the LCD. This means that the CURSOR instruction should be used as much as possible to specify the desired location of the output data. Various types of LCDs can be used, with one, two or four lines. The sample program *LCD.BAS* shows the character position corresponding to the start of each line.

Another option is use the instruction *UO 1* to redirect the entire output to the LCD. The instruction *UO 0* switches the output back to the terminal

Autostart

The original BASIC-52 tests the entire RAM after start-up, clears it and then starts any program that it recognises in the external EPROM. Both of the Elektor 80C537 computers have battery-backed RAM, so that the program that was last run before the computer was switched off remains present in the RAM. The BASIC initialisation routine has therefore been modified to automatically start a program that is already in the RAM.

All that you have to do is to load a program into the computer's memory. After this you can disconnect the serial cable to the PC and let the computer work on its own. Each time the computer is started up, the program is run anew, at least as long as the lithium battery provides enough power to maintain the contents of the RAM. If at any time you don't want the system to automatically run a program when it starts up, all you have to do is to send it a *NEW* instruction.

We haven't forgotten the 80535!

Now we come to the widely used 80535 systems, such as the Elektor 80535 Single Board Computer. The new interpreter can also be used with this system, if you first carry out a few modifications.

The system must run at 12 MHz, in order to yield a data transfer rate of 9600 baud. Everything else works as well, as long as you take the available processor hardware into account. The A/D converter has only eight channels, but is otherwise compatible. Naturally, the 80535 has fewer timers, ports and so on than the 80537.

Intel's BASIC-52 interpreter expects to find data RAM starting at address 0000h. The new initialisation routine does not change this. However, the 80535 Single Board Computer divides its 32 kB of RAM between the address ranges 4000h–7FFFh and C000h–FFFFh. Special logic allows the RAM in these regions to be used to store program code as well as data.

The necessary changes are quite simple. Remove IC2 (a quad NAND IC, type 74HC00) from its socket and replace it with three small wire bridges, as follows:

pin 3 to pin 4	($\overline{OE}$ direct to $\overline{RD}$)
pin 11 to pin 12	($\overline{CE}$ direct to *A14*)
pin 7 to pin 8	($\overline{CS}$ direct to ground)

In this configuration there is 16 kB of RAM available to the system, starting at address 0000h. The normal BASIC-537 interpreter can work with this. Of course, you must tell the interpreter that only 16 kB of RAM is available, by entering:

```
MTOP = 16383
```

This line should also be placed at the beginning of every program, since the interpreter initialises itself to 32 kB of RAM after a restart. Since all variables are stored just below the upper RAM boundary, the interpreter will function correctly only if it has accurate information about the amount of available RAM.

If you would rather be able to use the full 32 kB of RAM, you must make two further modifications, as follows:

Bend pin 1 of the RAM IC (A14) so that it does not stick into the socket, and then connect it directly to address line *A14* (for example, at pin 12 of the socket for IC2).

Bend pin 20 of the RAM IC ($\overline{CE}$) so that it also does not stick into the socket, and then connect it directly to address line *A15*, which is on pin 1 of the same socket.

Now the system will see the full 32 kB of RAM in one continuous block, starting at address 0000h. With this modification, it is not necessary to specify the amount of available memory with *MTOP*.

The 80535 Single Board Computer does not have battery-backed RAM. However, it is possible to use a 'zero-power' RAM, which has a built-in backup battery. If such a RAM is used, this computer behaves the same way as the new 80537 computer. With a zero-power RAM you can have a program start automatically. Even without this luxury, though, the Autostart function of the extended BASIC interpreter makes its presence known. This can be seen if a reset is executed while power is still applied. In this case the program that is already present starts immediately after the reset. This is a quite useful feature during program development. Instead of entering *Break* and then *Run*, you can simply press the Reset button.

An additional difference can be seen in the fact that it is possible to use a liquid crystal display. The '535 computer does not have any decoding logic for the LCD, so you will have to either build some extra circuitry or else connect the LCD to a port and use the 4-bit mode. The display can be driven by a somewhat complicated BASIC routine.

All of the supplementary ports of the 80535 can be addressed not only via their SFR addresses but also directly. The instructions *P3*, *P4* and *P5* can be used for output only. An output value can be directly assigned to a port (e.g. *P4* = 255). Reading input data from the ports with the *InP3*, *InP4*, *InP5* and *InP6* instructions is unfortunately not as convenient, since this must be done via the result stack. Port states that have been read can be fetched and put into a variable using the POP instruction:

```
InP4 : POP A : PRINT A
```

The direct port instructions also work with the 80C537, but only up to port P6. BASIC-52 allows only a limited number of new keywords. It is thus not possible to give every new function its own name. This means that we have to continue programming the special hardware of the microcontroller via register addresses. With read operations in particular, the use of register addresses (e.g. *Print SFR(0E8h)*) is often more elegant than the use of direct port instructions.

It is relatively easy to write some reset routines and supplementary instructions yourself, since the BASIC-52 interpreter already has suitable interfaces. You can find more detailed information in the relevant literature.

(000018-1)

You can find additional information and sample programs for 80535 and 80537 microcontrollers, and a DOS terminal emulator for BASIC-52, at the author's homepage on the Internet: http://home.t-online.de/home/B.Kainka

Project Software

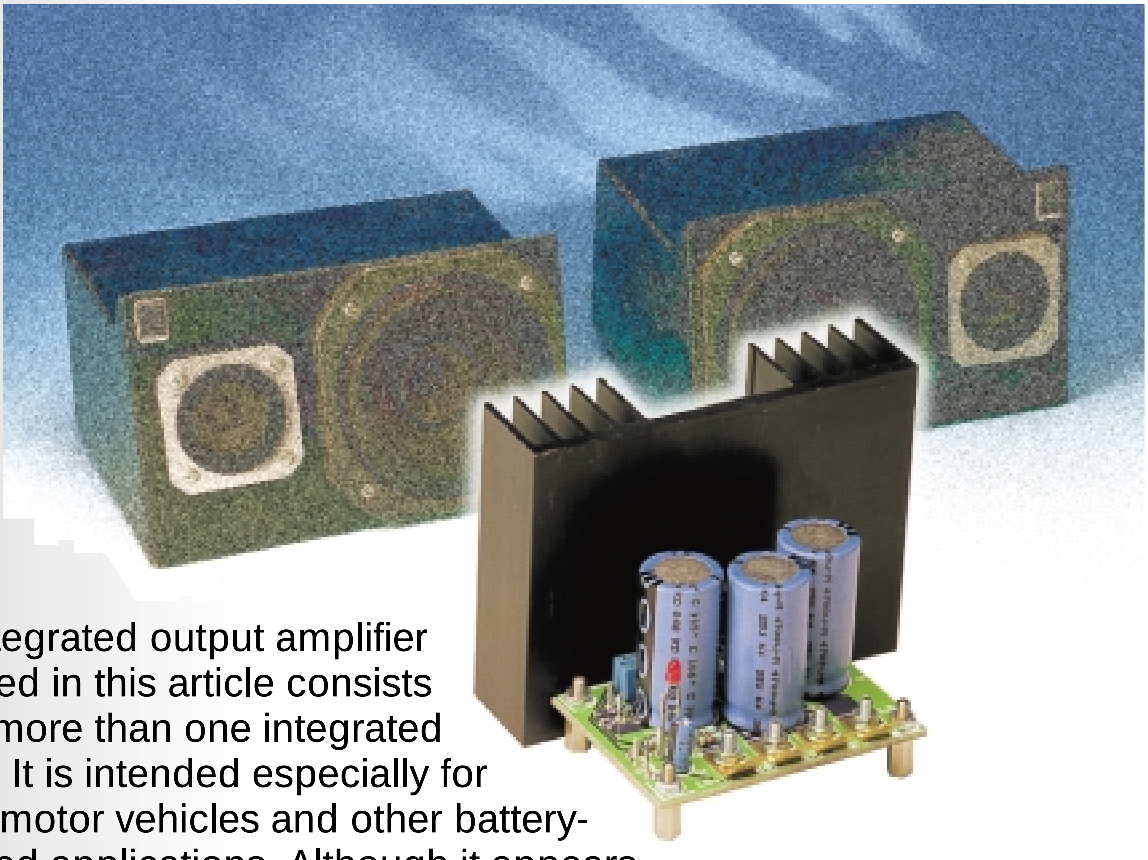
Diskette, order code **996029-1**.
(terminal emulator, help file and sample programs)

EPROM, order code **996532-1**.
(BASIC-537 interpreter)

designed for in-car use

one-IC audio power amplifier

50+ watts from a 12 V battery



The integrated output amplifier described in this article consists of little more than one integrated circuit. It is intended especially for use in motor vehicles and other battery-operated applications. Although it appears simple and hardly worth looking at, the amplifier can produce an appreciable audio power output.

Amplifiers come in all sorts. Most of the amplifier designs published in this magazine are intended for domestic or studio use. As such, they are usually powered by a 60–150 V supply, which may, moreover, be further divided into two balanced voltages. This makes these amplifiers unsuitable for use in motor vehicles.

Amplifiers that can operate from a 12 V supply are quite different units, designed as they are for low-voltage supplies. If, moreover, they are required to provide a substantial power output, their design is far from simple.

A simple calculation shows that a conventional amplifier operating from 12 V (or maximum 14.4 V when the battery is fully charged) cannot provide a power output of much more than 6 W. Use of a bridge arrangement may push this up to some 20 watts, but that is just about the maximum possible.

Nowadays, many motorists, and more particularly the younger ones, want considerably more power than 6–20 watts. Loudness is their god, but they unfortunately forget that this is an unforgiving god, leading them to early deafness.

Be that as it may, the most obvious way of increasing power output is to raise the supply voltage of 12 V to a higher level by means of a converter. Such a converter, however, is not exactly cheap and is, moreover, a notorious source of noise and interference. In view of the extensive electronic circuits in modern motor vehicles, this latter point is not to be underestimated.

Fortunately for the loudness-hungry, there is an alternative to a converter. Some years ago, Philips introduced a special integrated output amplifier chip, the TDA1560Q, that is able to provide 30 watts into 8 Ω from a 12 V power supply (without the use of a converter). This output power is

Design by T. Giesberts

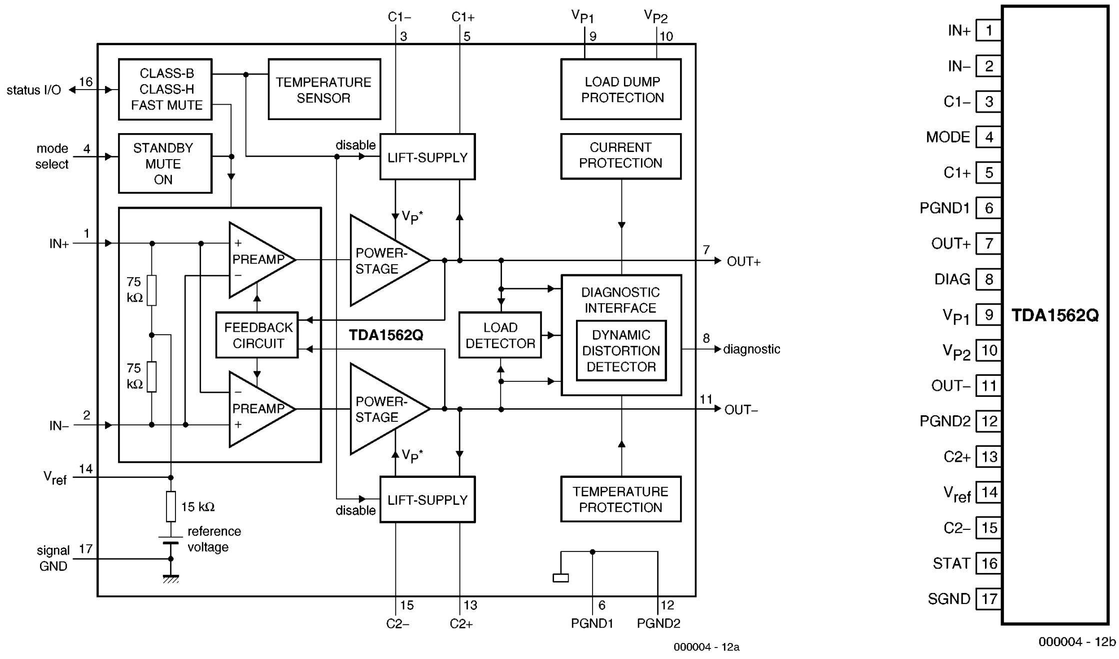


Figure 1.Block diagram of the innards of the TDA1562Q. A number of protection networks make the amplifier virtually foolproof.

obtained by operating the amplifier in Class H (see box elsewhere in this article).

An amplifier based on the Philips device was described in the February 1995 issue of this magazine.

Philips designers have further improved a number of properties of the IC, among which the power output. According to the Philips data sheet, the improved device, the TDA1562Q, can deliver 70 watts into 4 ohms, but that is at the cost of the distortion, which at 10 per cent is rather too high, even for in a car.

The prototype of the design described in this article provides 54 watts into 4 ohms at 1 per cent distortion. Since the number of requisite external components is smaller than in the case of the earlier device, the printed-circuit board is even more compact than that for the February 1995 amplifier.

The TDA1562TQ

The internal circuitry of the new device is very similar to that of its predecessor. Since, however, not everybody can be assumed to be *au fait* with the February 1995 article, the circuitry is described anew.

The block schematic of the device is shown in **Figure 1**. Particular attention is drawn to the boxes marked 'lift supply', which are necessary in a Class H system (see box).

The IC evaluates the input signal and estimates the consequent drive of the output transistors. When these transistors tend to go into saturation, the supply voltage is raised briefly by switching the capacitors connected to pins 3 and 5, and 13 and 15, in series with the supply voltage.

Apart from the input amplifier and Class H output amplifier, the device contains several protection circuits. One of these protects the IC against an excessive output current and against short-circuits.

The temperature protection circuit works in two steps. When the first threshold temperature is exceeded, the device switches from Class H to Class B operation. There is then no question of increased supply voltage. When the second threshold temperature is exceeded, the drive to the output transistors is reduced.

There is also a protection circuit against overvoltage and one against too low load impedances. When the load impedance drops below a predetermined critical level, operation is switched from Class H to Class B. A load impedance smaller than 0.5 Ω is seen as a short-circuit, which causes the device to be switched off altogether.

Brief technical data

Properties

High power output through Class-H operation
Low power dissipation during reproduction of music signals
Proof against short-circuits
Protection against excessive temperatures
Standby switch
No power-on or power-off clicks
Visible error indication

Measurement results *(at U_b=14.4 V)*

Supply voltage	*8–18 V*
Sensitivity	*760 mV r.m.s.*
Input impedance	*70 kΩ*
Power output	*54 W r.m.s. into 4 Ω (f=1 kHz; THD+N=1%)*
Harmonic distortion (THD+N)	*at 1 W into 4 Ω: 0.046% (1 kHz)*
	0.29% (20 kHz)
	at 35 W into 4 Ω: 0.12% (1 kHz)
	0.7% (20 kHz)
Signal-to-noise ratio (with 1 W into 4 Ω)	*88 dBA*
Power bandwidth	*7.5 Hz – 185 kHz (at 25 W into 4 Ω)*
Quiescent current	*about 135 mA ('on')*

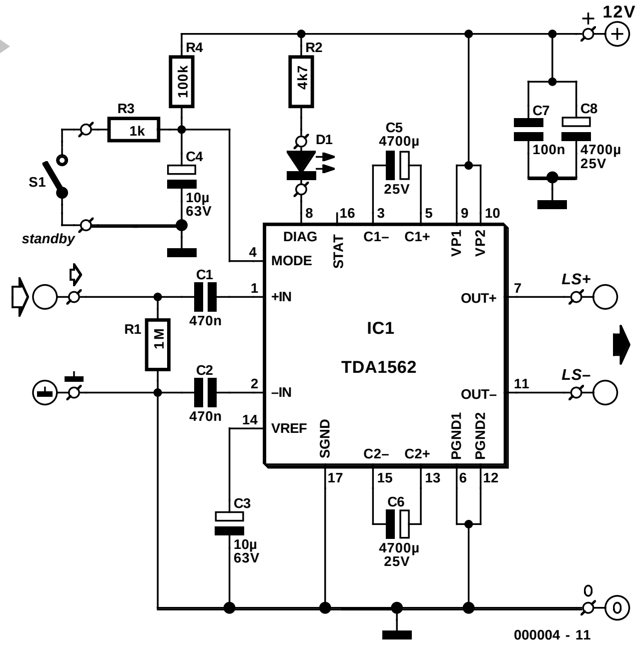


Figure 2. The circuit of the amplifier is conspicuous by its simplicity. Diode D1 is an error indicator.

Circuit description

The circuit diagram in **Figure 2** emphasizes how few external components are needed to construct a complete output amplifier (in fact, fewer than half the number in the February 1995 amplifier). For instance, the new device does not need compensation networks to enhance the stability. Also, because of the absence of switch-on phenomena, there is no need for a switch-on delay network.

There is, of course, still a need for supply line decoupling capacitors.

Capacitors C5 and C6 are required for Class-H operation, about which more in the box.

The value of input capacitors C1 and C2 is relatively low, thanks to the high input impedance of the IC.

Switched *RC* network R4-C4 at the 'mode select' input (pin 4) serves to switch the IC to 'mute' or 'standby'. When the supply voltage is switched on, the IC is first switched automatically to the 'mute' mode and to 'on' only after a short delay. The time constant R4-C4 is a few tenths of a second and this delay between the two states is sufficient to obviate disturbing (and annoying) switch-on phenomena.

Switch S1 enables the amplifier to be switched to 'standby' when the use of the amplifier is not needed for a period of time. When that time has elapsed, the amplifier is quickly reverted to normal operation. The current drain in the standby mode is virtually negligible at only 200 µA.

Resistor R3 prevents a short-circuit current ensuing when S1 is being closed at the instant C4 is being discharged.

Error indication

The diagnostic output (pin 8) of the TDA1562Q is one of the few facets of the device that is completely new. As shown in the diagram, it can now be used to drive an visible error indicator, D1, directly. During normal operation the diode should be out. Its lighting may be caused by one of four possible causes.

1. The amplifier is being overdriven. The internal circuit responsible for the indication is the 'Dynamic Distortion Detector' (see Figure 1). In practice, this will be the case when the distortion rises above 1.6 per cent at 1 kHz. The diode is, therefore, a kind of clipping indicator.
2. There is a short-circuit between the outputs or between one of the outputs and the supply line. In the first case, the outputs are disabled, whereupon the protection network ascertains at short intervals of time whether the short-circuit has been removed. The DIAG output is then disabled for 30 µs at 20 ms intervals. In the case of a short-circuit between one of the outputs to the supply line, the DIAG output remains active.
3. The internal sensor measures a temperature of 145 °C, whereupon the

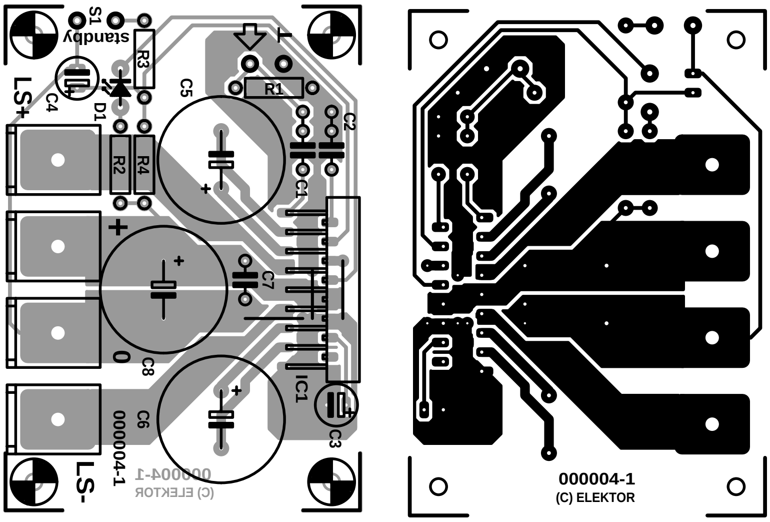


Figure 3. The compact printed-circuit board for the amplifier is available through the Readers' services (towards the end of this issue).

COMPONENTS LIST

Resistors:
R1 = 1MΩ
R2 = 4kΩ7
R3 = 1kΩ
R4 = 100kΩ

Capacitors:
C1,C2 = 470nF
C3,C4 = 10µF 63V radial
C5,C6,C8 = 4700µF 25V radial (18mm max. dia., raster 7.5 mm)
C7 = 100nF, raster 5 mm

Semiconductors:
D1 = high-efficiency-LED
IC1 = TDA1562Q (Philips)

Miscellaneous:
S1 = single-pole on/off switch
Four spade connectors, PCB mount
Heatsink for IC1 (R_{th}<2.5 K/W)
PCB, order code **000004-1** (see Readers Services pages)

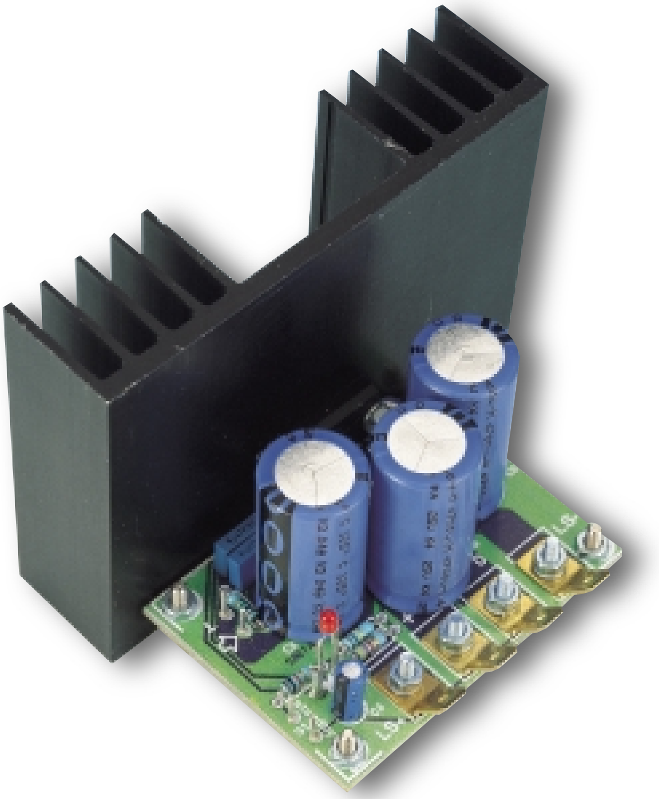


Figure 4.There cannot be many 50 W output amplifiers that are this small.

relevant protection circuit is enabled.

4. The IC is in the power-up state. When it is switched to the 'on' state, the diode goes out. If, however, the output impedance is not to specification during power-up, the diode remains lit.

A COMPACT BOARD

The amplifier is best built on the single-sided printed-circuit board shown in **Figure 3** (available through our Readers' services – see towards end of this issue). As mentioned earlier, the board is small for an output amplifier. In fact, most of its surface is taken up by the four car-type (spade) connectors via which the power supply and the loudspeakers are connected to the amplifier.

The small size of the board also creates a few difficulties. If, for instance, the electrolytic capacitors are mounted first, the fitting of the IC becomes pretty difficult. It is, therefore, advisable to start with fitting the IC onto the heat sink (using plenty of heat conducting paste).

Also, ensure that after the board has been placed in a suitable enclosure, its solder (track) side remains readily accessible. After the board and heat sink have been secured in the case, solder the pins of the IC to the board.

Figure 4 shows a photograph of the completed prototype as built up in our design lab.

Make sure that when the board is fitted in the case, the loudspeaker terminals are not (and cannot be) short-circuited to earth. Although the IC is protected against much abuse and misuse, it is always better not to tempt providence.

In most cases, two amplifier modules will be fitted in one enclosure: one for the left-hand channel and the other for the right-hand channel.

Constructors who find 2×54 watts insufficient may consider driving the front and rear speakers in the vehicle by separate amplifiers. This would take the total number of modules required to eight, and these deliver some 400 watts of audio power into the vehicle. A case of minding the windows!

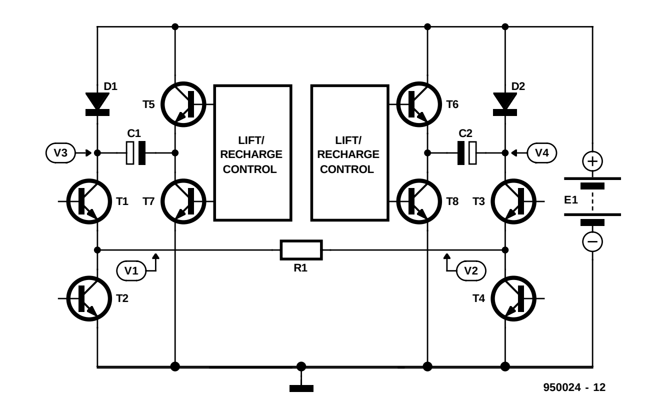
[000004-1]

Text (Dutch original): S. van Rooij

Class-H operation

A Class H amplifier is somewhat akin to a Class G amplifier, which is a power amplifier in which two Class B amplifiers with different supply voltages are combined. Small-amplitude signals are boosted by the one with the lower supply voltage, resulting in much higher average efficiency for speech and music. When signals exceed the low-voltage supply amplitude, the amplifier that operates from the higher supply voltage takes over, while the first one is switched off.

In a Class H amplifier the supply voltage is varied by an efficient Class S amplifier or the arrangement in the present design so that it remains just above the minimum value required to prevent saturation. This configuration also achieves a much higher average efficiency for speech and music signals. (A Class S amplifier is a pulse-width-modulated audio amplifier in which the active elements are switched by a control frequency several times higher than the signal frequency being amplified. This type of amplifier has an efficiency of some 90 per cent).



In the present amplifier, Class H operation is achieved by the use of two switched capacitors, C1 and C2. These are charged to the battery voltage and switched into series with the 12 V supply line during brief drive peaks.

In the diagram, transistors T1–T4 are part of the 'normal' amplifier, while T5–T8 and C1 and C2 are added for Class H operation. At small-signal amplitudes, T7 and T8 are on and C1 and C2 are charged to the battery voltage via diodes D1 and D2. When the signal amplitude tends to drive T1 or T3 into saturation, the internal sensor (lift/recharge control) detects this and causes T7 and T8 to be cut off and T5 and T6 to be switched on. When that happens C1 and C2 are in series with the supply line; diodes D1 and D2 prevent their being discharged into the battery.

When the input signal amplitude drops, the capacitors are linked to earth again by T7 and T8, whereupon their charge is topped up.

desoldering

a craft in its own right

We can assume that the majority of our readers have sufficiently mastered the noble art of soldering. However, the art of unsoldering already soldered components, or desoldering, is something else. Even rather experienced solder artists often prove to be extremely clumsy in this area. They go about it the wrong way, and lacking both patience and the right touch, they tend to mess around until they have ruined not only the component but also the printed circuit board.

By K. Walraven

Pumping or sucking?

The only good way to desolder is to patiently remove all of the solder from the points where the component is attached to the circuit board. This can be done using a special ‘solder-sucker’ soldering iron, a vacuum pump or desoldering braid.

Of these various options, the vacuum pump and the special soldering iron may seem to be the most convenient at first glance, but in practice this is not so. The nozzles of these tools are easily clogged, and the sucking action is not always perfect. If you use a separate pump, you have to heat the solder joint quite strongly, which is in itself risky. Even then you have to quickly switch between the soldering iron and the pump, since otherwise the solder will cool down and become hard again.

In short, even though using desoldering braid may appear to be the most primitive of the three options, our experience in the Elektor lab is that it yields the best results.

Braided wire

Desoldering braid or ‘Wick’ is actually a braided wire ribbon consisting of thin copper wires together with impregnated solder flux. The spaces between the wires soak up the solder, so that it is removed from the place where it is no longer desired. Desoldering braid can be obtained in small rolls a few metres long (see **Figure 1**).

Using desoldering braid is very easy. Before you start, remember that solder flows best when it is good and hot. You should thus not use too small a soldering iron. If you have a temperature-controlled soldering iron, set it to at least 350 degrees. Now all you have to do is to make sure that the desoldering braid is hotter than the joint that is to be desoldered, since the solder will

1

flow to the hottest point.

Proceed as follows:

- First place a clean piece of desoldering braid on the solder joint that you want to remove, and then place the tip of the soldering iron on top of the braid.
- Now press quite firmly with the soldering iron, to make good thermal contact (see **Figure 2**). The solder will be drawn into the braid.
- As soon as the braid is saturated, it will not absorb any more solder. You can easily see how saturated it is from its colour. It is initially copper-coloured, but becomes increasingly silver-coloured as it absorbs more and more solder (see **Figure 3**).

In order to fully clean the solder from a joint, you sometimes have to work in two steps. In the first step you remove most of the solder. After this, you can slowly draw the braid along under the soldering iron, while holding the soldering iron stationary (this takes a bit of practice). Following this, you can remove the last remains of the solder with a fresh, unused piece of desoldering braid.

FLUX

Once you have gained some experience in working with desoldering braid, you will quickly realize that this technique works well only if sufficient flux is present. If there is not enough flux, even liquid solder simply forms a ball and will not soak into the braid.

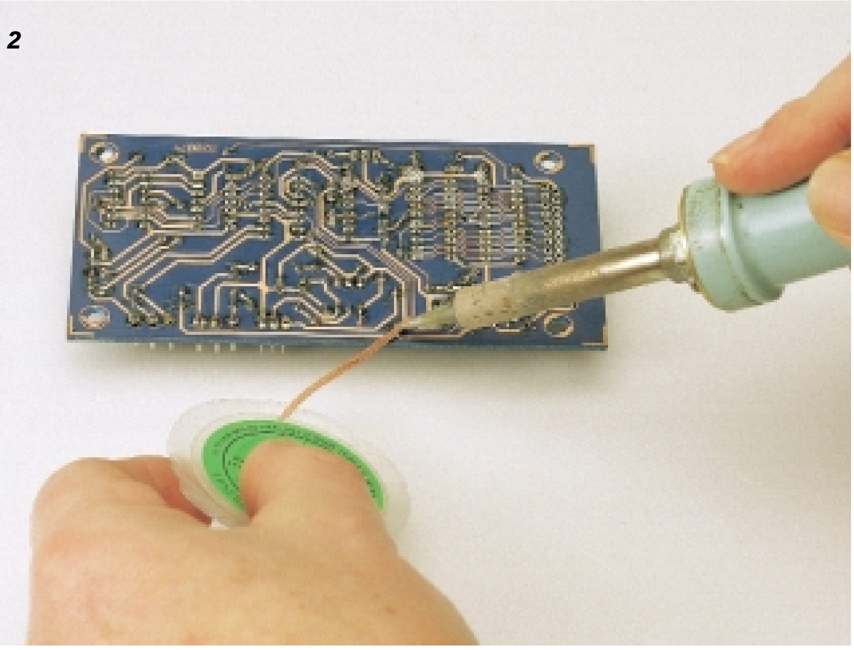
Sometimes certain parts of the braid do not have enough flux. This can be caused, for example, by repeated flexing of the braid at a particular location.

This fault can be remedied by putting additional flux on the solder joint that is to be removed. You can buy flux paste in tubes for this purpose. You can also use violinist's rosin, which is available in small tins, to impregnate the braid. Since rosin is hard, getting it into the braid is a bit tricky. Set your soldering iron to a low temperature (100 to 150 degrees); if you do not have a regulated iron, you will just have to be quick. Lay a small piece of braid on the surface of the rosin and warm the braid from the top with the soldering iron (see **Figure 4**). The rosin will melt and soak into the braid. Hold the iron stationary and pull the braid through underneath it.

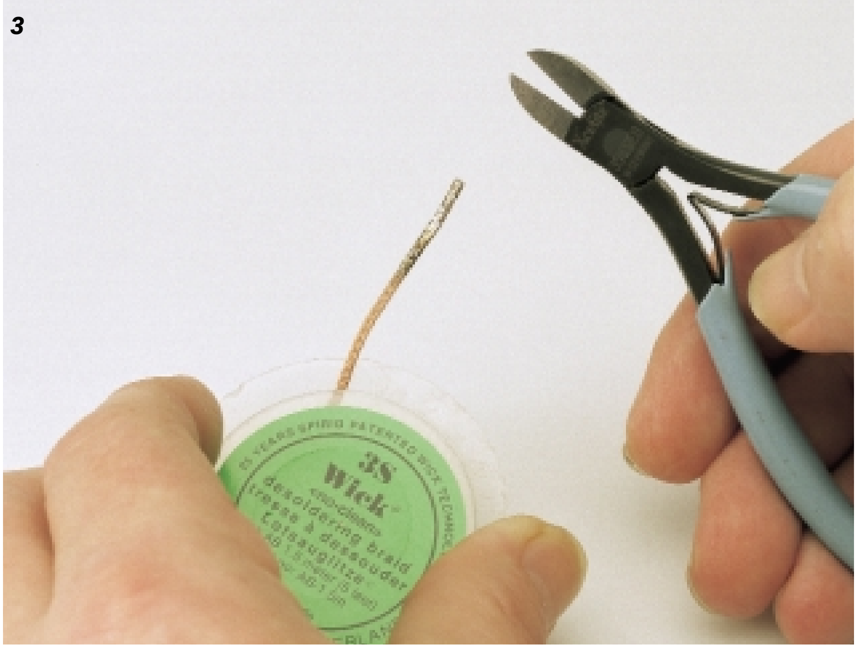
You can also make your own desoldering braid in this manner. You can use any piece of stranded wire (from a bit of mains cable, for example) — just remove the insulation and impregnate it with flux. This is handy to know if you run out of desoldering braid, or simply find it too expensive.

(000006-1)

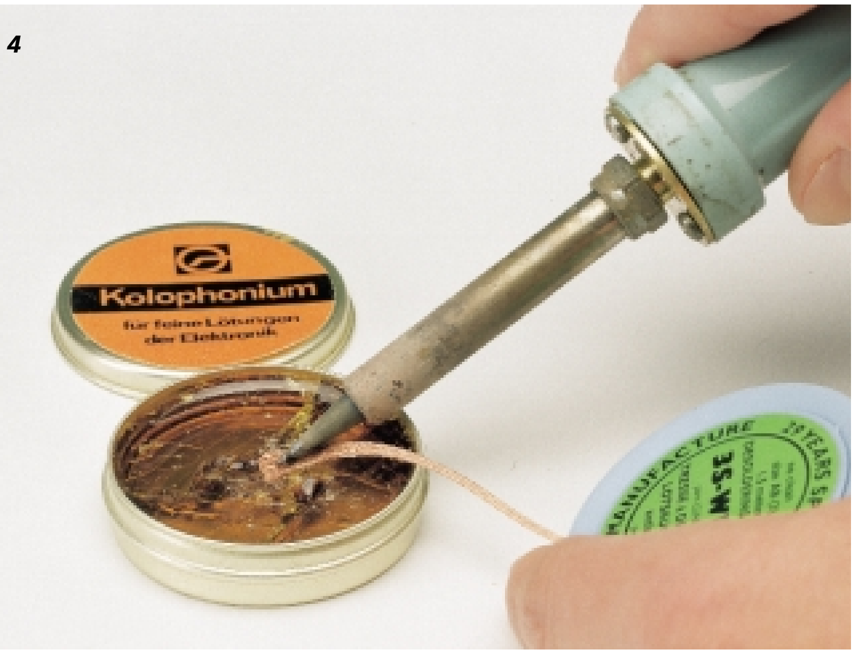
2



3



4



537 ‘Lite’ computer (2)

part 2: programming and practical use

Having built the board and successfully run the first communication tests, you are ready to start programming the 537 core system. In doing so, some rather special points should be observed which are discussed in this article. What’s more, we show you how the board may be used in practice by adding extensions and components.

Design by Prof. B. vom Berg and P. Groppe

MODES OF THE 537 ‘LITE’ COMPUTER

The fact that it can be used in three different modes adds considerably to the power and versatility of the the 537 Lite board. The modes are Stand Alone, Program Download and Run. **Table 1** shows how to set jumpers and switches to select between these modes.

Stand Alone mode is typically used for the final application of the board, when a program has been fully tested and is known to work error-free in the target system. After a reset, the user program is immediately executed from EPROM.

In **Program Download** mode, a terminal program is used to download the program (developed on a host PC) to the 537 Lite board. The program then has to be started from the host PC. The EPROM contains the monitor program, while the RAM chip acts as a combined program and data storage device. In this mode, jumper J7 has to be set to the 12 MHz crystal position. If not, the communications data speed is not properly set up. However, once the program has been downloaded, J7 may be changed at any time, if only to ensure that the program runs about 33% faster after a reset. Do remember, however, that the higher clock speed also results in time-dependent routines being executed faster.

The EPROM-resident monitor program allows simple direct entry of assembler commands as well as comfortable debugging at assembly-code level. You may enter *help* to call up an overview of all available functions.

In **Run mode** the previously downloaded user application program may be executed straight away by means of a reset, after which the program runs in the RAM area. The RAM component then acts as an EPROM program storage device, obviating the need to program your own EPROM to be able to employ the system in stand-alone mode running the final version of the program. The RAM component acts as a shared program and data memory. An essential component in this mode is the backup battery, because, as you will no doubt be aware, a RAM chip is a ‘volatile’ device which means that it loses its contents when the supply voltage disappears.

When the Monitor EPROM is used (mode 2) the serial interface SS0 is automatically configured for 9,600 baud, one stop bit and no parity (9600, N, 8, 1), which should enable flawless communication with the host PC. However, if the program is to run in Stand Alone or Run mode, this serial interface initialisation is sadly missing. Unfortunately, you have to provide and embed the necessary routine for SS0 yourself to make sure the communication with the PC operates as it should.

PROGRAMMING

Before you can start programming away, you have to know some elementary facts about the memory configuration of the 537 Lite board. After all, programs and user data are both stored in RAM component IC7. This shared memory space extends from 0000_H through $7FFF_H$, so its size is 32 kBytes. In the programming environment (8051 Compiler or Interpreter) you have to indicate exactly which ranges are reserved for the program and which ones for the data. If you fail to do this, memory contention may occur and program(s) and data may actually destroy each other!

Table 1. Jumper- and switch settings of the three modes

Mode	***Jumper J1A,J1B,J1C***	***Switch S1***	***EPROM IC2***	***RAM IC7***
Stand-alone	*1-2*	*don’t care*	*user program**	*Data**
Program download	*1-2*	*LOAD RAM*	*Monitor*	*user program and data**
Run	*2-3*	*RUN*	*not active*	*user program and data*

**: 32 kByte device, address range 0-7FFF$_H$*

You should also be aware that the EPROM-resident Monitor program, whose functions include control over the program download operation, also requires a small scratch area in RAM. Obviously, this area may not be overwritten by the program code or user data! The monitor scratch area is defined as $7F00_H$ through $7FFF_H$ in RAM.

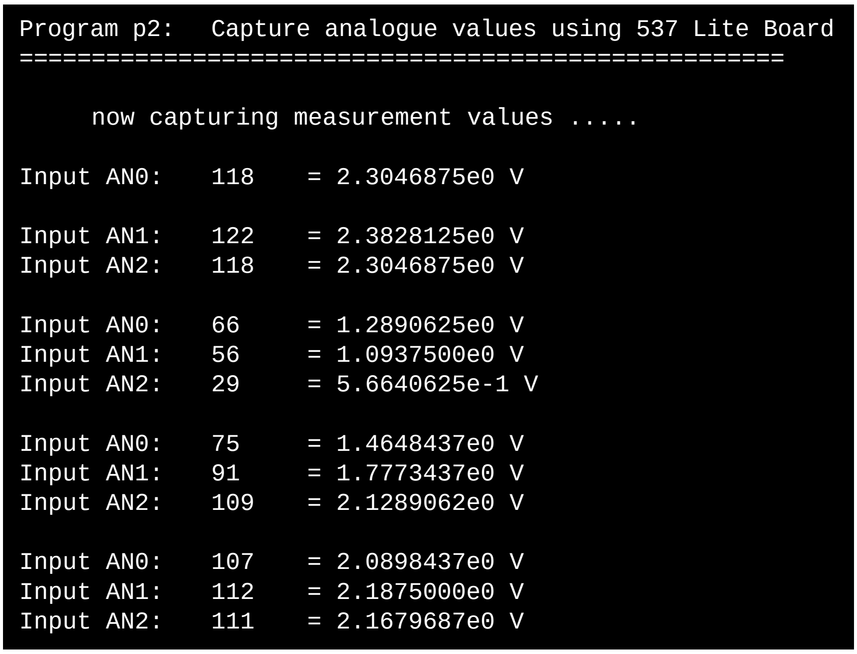
All this has the following implications if you want to start writing programs:

7FFF	*End of RAM range (fixed): 32 kByte*
	Size: 256 bytes
7F00	*Fixed start address of memory range for Monitor program*
7EFF	*End of data memory range (fixed)*
	Size: 7.75 kBytes
6000	*Start of data memory range (variable)*
5FFF	*End of program code range*
	Size 24.0 kBytes
0000	*Start of program code memory range (variable)*

Table 2. RAM memory map

Figure 1. P2.PAS listing.

```
(************************************************************************)
(***     p2: Capture analogue values with the 537 Lite Board          ***)
(***     ===================================================          ***)
(***     Version:  1.0,  12.10.99,  16:23                             ***)
(***     Programmer: v.Bg.                                            ***)
(************************************************************************)

program p2;

(*** Definition of Variables *******************************************)
const
  (* For A/D converter *)
  adcon0 = $d8;
  addat  = $d9;
  dapr   = $da;
  busy   = $dc;
var
  (* A/D converter *)
  adu:byte;

  (* Various *)
  i,zw: byte;

(*** Suroutine Collection   *********************************************)

(*** Main program  ******************************************************)

begin
  (*** Clear screen ***)
  write(chr($1a));

  (*** Intro text ***)
  writeln('Program p2: Capture analogue values using 537 Lite Board');
  writeln('========================================================');
  writeln;
  writeln('  now capturing measurement values .....');
  writeln;

  (***   Main loop   ***)
  repeat                              (* Start of endless loop *)

    (* Capture and display measurement values *)
    for i:=0 to 2 do                  (* Convert 3 measurement values *)
      begin
        adu:=reg(adcon0);             (* Read adcon *)
        adu:=adu and %11000000;               (* single conversion, internal start *)
        adu:=adu or i;                (* Channel selection *)
        writereg(adu,adcon0);         (* Copy value into SFR *)
        writereg(0,dapr);             (* Start conversion *)
        repeat until (bit(busy)=false);       (* Wait for done *)
        zw:=reg(addat);                       (* Read converted value *)
   write('Input AN',i,':  ',zw);      (* Put on display *)
   writeln('  =  ',(zw*5)/256,' V');(* Convert into Volt and display *)
      end;
    writeln;                          (* Supply empty line *)

    (* Wait: approx. 1000 ms until next measurement *)
    wait_25ms(40);

  until false;                        (* Exit from endless loop *)

end.
```



```
Program p2:   Capture analogue values using 537 Lite Board
====================================================

     now capturing measurement values .....

Input AN0:    118    = 2.3046875e0 V

Input AN1:    122    = 2.3828125e0 V
Input AN2:    118    = 2.3046875e0 V

Input AN0:    66     = 1.2890625e0 V
Input AN1:    56     = 1.0937500e0 V
Input AN2:    29     = 5.6640625e-1 V

Input AN0:    75     = 1.4648437e0 V
Input AN1:    91     = 1.7773437e0 V
Input AN2:    109    = 2.1289062e0 V

Input AN0:    107    = 2.0898437e0 V
Input AN1:    112    = 2.1875000e0 V
Input AN2:    111    = 2.1679687e0 V
```

Figure 2. Measurement results produced by P2.PAS on the PC display.

- The total RAM range available for program and data extends from 0000_H through $7EFF_H$, so its size is 32,512 bytes.

- The program memory range always starts at address 0000_H, and is immediately followed by the data memory range.

- The data memory range may not reach into the monitor memory range. The memory division is illustrated in **Table 2**.

At this point, an example may help to show what considerations play a role in practical programming.

Let's assume that a program with a size of about 20 kBytes and requiring about 5 kBytes of data memory is not quite finished and therefore requires some additional memory locations. The compiler/interpreter should be 'informed' that the program code range starts at 0000_H, and the data range, at 6000_H. Consequently the program space has a size of 24,576 bytes (24 kBytes), while the data memory measures 7,936 bytes (7.75 kBytes).

When the program is developed further, you should take care that the program code does not cross the $5FFF_H$ border, and the data, the $7EFF_H$ border. If that happens, the two memory blocks will corrupt one another.

Programming the 537 Lite board takes four steps:

- The user program is written in a popular and widely used programming language like Assembler51, C51, BASIC52, PL/M51 or Pascal51. BASIC52 and Pascal51 in particular allow very fast and powerful results to be obtained. Whatever programming language is used, the final product of your programming efforts, a block of object code, should be either an Intel-Hex file, a file with the extension .hex or an ASCII file containing BASIC tokens.

- The object code file is downloaded to the 537 board via one of the available RS232 interfaces on the PC.

- Test the program in combination with the target system hardware.

- Once all errors have been sorted out and the program works as it should, there are two possibilities to accommodate the final version of the software in the system.
  – The first option is to program an EPROM and install it instead of the Monitor EPROM. The system then runs in Stand Alone mode (mode 1), so that 32 kBytes of data/program memory is available.

  – The second option is to switch S1 to RUN (mode 3) after the program download operation, prompting the program to execute from RAM. When a backup battery is used, the RAM contents is not lost when the battery voltage is switched off. A Lithium battery should have enough energy for several years of problem-free use.

The advantage of the second option is that you do not have to burn an EPROM. Consequently there's no need to have an EPROM programmer and/or UV eraser available.

PRACTICAL APPLICATIONS

Of course, the 537 core should not lie idle among half a dozen or so other microcontroller boards in one of your drawers. We strongly suggest you actually use the board by giving it a practical application. A good example is that of a CAN bus system host as described in last month's issue of this magazine. Other experimental circuits and applications include peripheral devices with serial control (3-channel D-A converter, temperature sensor, LED display, etc.), keypad/keyboard interfaces, voice production, digital and analogue I/O cards, IrDA, radio and fibre-optic communication systems. Many such projects may be found in existing literature, or will be published shortly.

Even if space is at a premium in this magazine, an example should be included that illustrates the use of the 537 Lite computer in combination with the DOS version of Pascal51. This integrated programming environment also runs in a Windows 95/98 DOS box.

Having launched the editor *nilied.exe* you first have to set up the desired memory model. This is done in the menu Options | NiliPascal Parameters (**Figure 1**). The minimum settings are the definitions of data and program memory start addresses, the microcontroller clock frequency and the baud rate of the communication between the PC and the 537 Lite board. Your personal preferences and settings are saved and the menu is closed.

At this point you may enter a program or load an available piece of code, for example, one of the examples found on the project diskette, 976008-1. The example program *p2.pas* captures input voltages on the first three inputs AN0, AN1 and AN2 of the 537's on-chip A-D converter. The sampling rate is one per second. You may apply different voltages in the range 0-5 V to pins 3, 4, 5 and 15 of connector K3 and check the associated conversion results on the display.

The program *p2.pas* should be compiled without errors after pressing Shift-F9. Next, you quit the program editor by typing Alt-x which takes you back to the DOS level. The next step is to download the object code to the 537 Lite board and run it! **Figure 2** shows what you should see on your display.

(990054-2)

Design editing: K. Walraven
Text editing
(German original): R. Gerstendorf

HT12D

Integrated circuits
Special Function

HT12D
2^{12} Series of Decoders

Manufacturer
Holtek Semiconductor Inc.,

HOLTEK

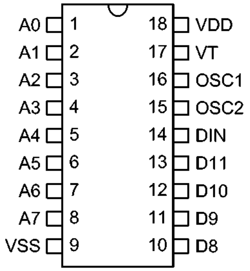
No.3 Creation Rd. II, Science-based Industrial Park,
Hsinchu, Taiwan, R.O.C. Fax: 886-3-563-1189.
Internet: http://www.holtek.com.tw

Features (HT12D only)

- Operating voltage: 2.4V-12V
- Low power and high noise immunity CMOS technology
- Low standby current
- Capable of decoding 12 bits of information
- Pair with Holtek's 2^{12} series of encoders
- Binary address setting
- Received codes are checked 3 times
- Address/Data number combination
- 8 address bits and 4 data bits
- Built-in oscillator needs only 5% resistor
- Valid transmission indicator
- Easy interface with an RF or an infrared transmission medium
- Minimal external components

Application example
IR Remote Control for Sony MZ-R30 MiniDisc Walkman, *Elektor Electronics* February 2000.

8-Address
4-Data



HT12D
–18 DIP

Pin Assignment

Pin Descriptions			
Pin Name	**I/O**	**Internal Connection**	**Description**
A0 – A11	I	NMOS TRANSMISION GATE	Input pins for A0-A11 setting. They can be externally set to VDD or VSS
D8 – D11	O	CMOS OUT	Output data pins
DIN	I	CMOS IN	Serial data input pin
VT	O	CMOS OUT	Valid transmission, active high
OSC1	I	OSCILLATOR	Oscillator input pin
OSC2	O	OSCILLATOR	Oscillator output pin
VSS	I	—	Negative power supply (GND)
VDD	I	—	Positive power supply

HT12A

Integrated circuits
Special Function

HT12A
2^{12} Series of Encoders

Manufacturer
Holtek Semiconductor Inc.,

HOLTEK

No.3 Creation Rd. II, Science-based Industrial Park,
Hsinchu, Taiwan, R.O.C. Fax: 886-3-563-1189.
Internet: http://www.holtek.com.tw

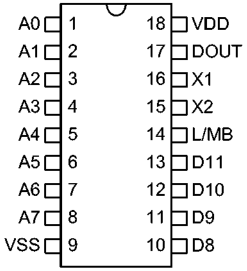
Features (HT12A only)

- Operating voltage 2.4V- 5V
- Low power and high noise immunity CMOS technology
- Low standby current: 0.1μA (typ.) at V_{DD} =5V
- 38kHz carrier for infrared transmission medium
- Minimum transmission: one word
- Built-in oscillator needs only 5% resistor
- Data code has positive polarity
- Minimal external components
- 18-pin DIP or 20-pin SOP package available

Application example
IR Remote Control for Sony MZ-R30 MiniDisc Walkman, *Elektor Electronics* February 2000.

General Description
The 2^{12} encoders type HT12A and HT12E are CMOS LSIs for remote control system applications. They are capable of encoding information which consists of N address bits and 12–N data bits. Each address/data

8-Address
4-Data

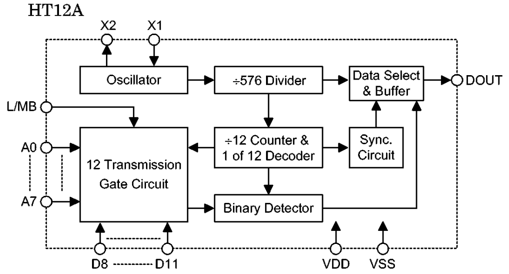


HT12A
–18 DIP

Pin Assignment

Pin Descriptions (HT12A only)			
Pin Name	**I/O**	**Internal Connection**	**Description**
A0 – A7	I	CMOS IN, pull-high	Input pins for A0-A7 setting. These pins can be externally set to VSS or left open
D8 – D11	I	CMOS IN, pull-high	Input pins for data D8-D11 setting and transmission enable, active low. These pins can be externally set to VSS or left open
DOUT	O	CMOS OUT	Encoder data serial transmission output
L/MB	I	CMOS IN pull-high	Latch/Momentary transmission format selection pin. Latch: floating or VDD. Momentary: VSS
OSC1	I	OSCILLATOR 1	Oscillator input pin
OSC2	O	OSCILLATOR 1	Oscillator output pin
X1	I	OSCILLATOR 2	455 kHz resonator oscillator input
X2	O	OSCILLATOR 2	455 kHz resonator oscillator output
VSS	I	—	Negative power supply (GND)
VDD	I	—	Positive power supply

DATA trigger



HT12A Block Diagram

input can be set to one of the two logic states. The programmed addresses/data are transmitted together with the header bits via an RF or an infrared transmission medium upon receipt of a trigger signal. The capability to select a DATA trigger on the HT12A further enhances the application flexibility of the 2^{12} series of encoders. The HT12A provides a 38kHz carrier for infrared systems.

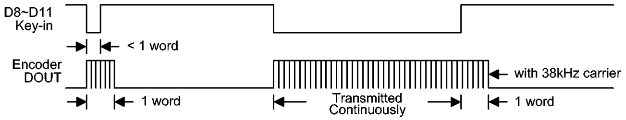
Functional description

Operation

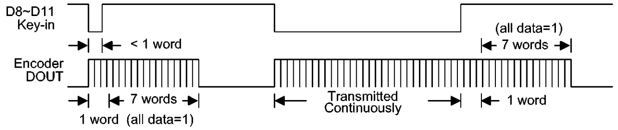
The HT12A encoder begins a 4-word transmission cycle upon receipt of a transmission enable (D8-D11, active low). This cycle will repeat itself as long as the transmission enable (D8-D11) is held low. Once the transmission enable returns high the encoder output completes its final cycle and then stops as shown in the transmission timing diagram.

Information word

If L/MB = 1 the device is in the latch mode (for use with the latch type of data decoders). When the transmission enable is removed during a transmission, the DOUT pin outputs a complete word and then stops. On the other hand, if L/MB = 0 the device is in the momentary mode (for use with the momentary type of data decoders). When the transmission enable is removed during a transmission, the DOUT outputs a complete word and then adds 7 words all with the '1' data code.

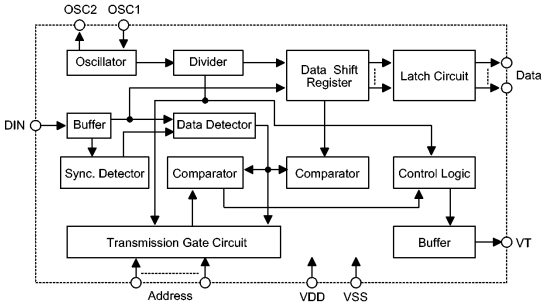


Transmission timing for the HT12A (L/MB=Floating or VDD)



Transmission timing for the HT12A (L/MB=VSS)

Transmission timing for HT12A (L/MB = VSS)



HT12D Block Diagram

General Description

The HT12D decoder is a CMOS LSIs for remote control system applications. It is paired with Holtek's 2^{12} series of encoders. For proper operation, a pair of encoder/decoder with the same number of addresses and data format should be chosen. The decoders receive serial addresses and data from a programmed 2^{12} series of encoders that are transmitted by a carrier using an RF or an IR transmission medium. They compare the serial input data three times continuously with their local addresses. If no error or unmatched codes are found, the input data codes are decoded and then transferred to the output pins. The VT pin also goes high to indicate a valid transmission.

The HT12D decoder is capable of decoding information consisting of N bits of address and 12–N bits of data. The HT12D is arranged to provide 8 address bits and 4 data bits.

Functional description

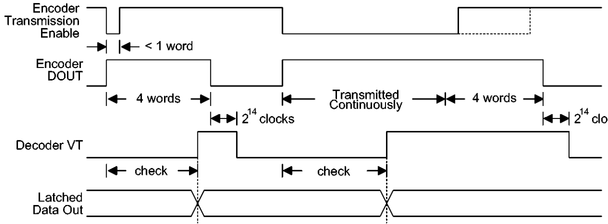
Operation

The HT12D decoder receives data that are transmitted by an encoder and interpret the first N bits of code period as addresses and the last 12–N bits as data, where N is the address code number. A signal on the DIN pin activates the oscillator which in turn decodes the incoming address and data. The decoders will then check the received address three times continuously. If the received address codes all match the contents of the decoder's local address, the 12–N bits of data are decoded to activate the output pins and the VT pin is set high to indicate a valid transmission. This will last unless the address code is incorrect or no signal is received. The output of the VT pin is high only when the transmission is valid. Otherwise it is always low.

Output type

The HT12D provides 4 latch type data pins whose data remain unchanged until new data are received.

Decoder timing



Decoder timing for HT12D