JANUARY 2002
£3.20

ELEKTOR ELECTRONICS

THE ELECTRONICS & COMPUTER MAGAZINE

www.elektor-electronics.co.uk

IR Transceiver for PCs

Hard Disk Interface for Printer Port

SMS Interface for GSM

remote process control by GSM phone

- Quizmaster — Wireless!
- Microprocessor Basics Course
- Lighting and Gearbox Control
- Voice Extreme Toolkit

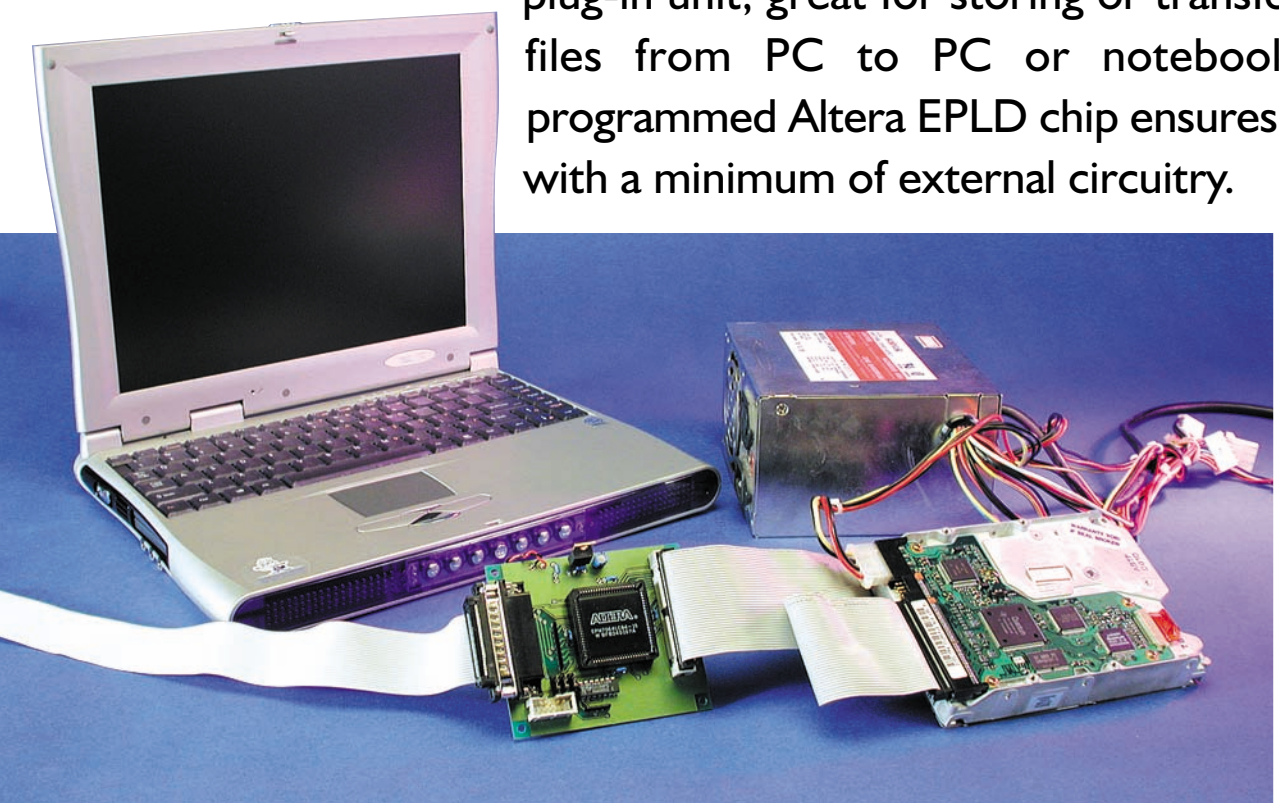
9 770268 451098 01

Hard Disk Interface for the Printer Port

IDE2LPT – removable storage via the printer port

Design by Andrew Buckin (from an idea by Leonid Slobodchikov) AndrewBuckin@aol.com

Can't even give away that hard disk drive on your redundant PC system? Why not use it as a portable drive? This interface adapter allows a standard low-cost internal hard drive to be used as a convenient mobile plug-in unit, great for storing or transferring large files from PC to PC or notebook. A pre-programmed Altera EPLD chip ensures a neat unit with a minimum of external circuitry.



It's difficult to believe that the first PC systems came without a hard disk drive. How did we ever manage? It was only when hard drives appeared that the real convenience of a PC was realised, we didn't mind that these drives could only store a few Mbytes and were about the same size and weight as a common house brick, it was certainly an improvement on reloading MS-DOS from a boot disk at every start-up.

Modern Hard Disk Drives (HDDs) are by comparison vastly improved, slim jewels of precision engineering. Floppy disks used to be sufficient to store and transfer complete programs but nowadays with ever-increasing program size a removable HDD seems a better alternative. Standard non-removable internal HDDs offer the highest storage capacity at the lowest cost but plug directly into the Integrated Disk Electronic (IDE) interface on the motherboard. The PC has no external

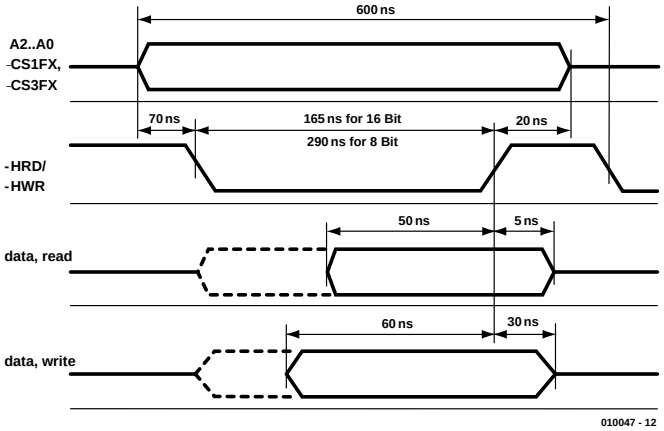


Figure 1. IDE bus read and write timing diagram.

Adapter Specification:

Maximum size of hard disk drive: Unlimited.
Maximum number of hard disk drives: 1
Data transfer rate: up to 100 kByte/s
Power requirements: 5 V / < 10 mA
(from an external 12 V supply)

IDE connector as standard. If we are to use such an HDD as a portable unit plugging into any PC, it will be necessary to adapt an existing external PC port.

PC's have always had a parallel printer port so it seemed like a good idea to develop an interface adapter (implemented as an Altera CPLD) along with some software that would allow data to be transferred to and from a hard disk via a printer port.

LPT to IDE

In the March 2001 edition of Elektor Electronics the article 'IDE hard disk interface for 8-bit controllers' described an adapter that enabled 8-bit controllers to use a hard disk with an IDE interface. That article described in detail the IDE interface (also known as the ATA interface) and its history so we will not repeat all the information here but just look at some of the important signals that are used in this design (a complete specification is available at www.t13.org). An overview of the signals on this 40-pin connector is given in Table 1.

The LPT2IDE interface makes use of the following signals:

HD15 to HD0 Data lines (8 or 16 bit wide bus, bi-directional).

CS1FX/Chip Select 0 this selects the command register blocks.

CS3FX/Chip Select 1 this selects the status register blocks.

HRD / I/O-read Read signal for I/O port address. Data is put onto the bus (HD0 to HD7 or HD0 to HD15) by the hard disk when this signal is low and latched into the

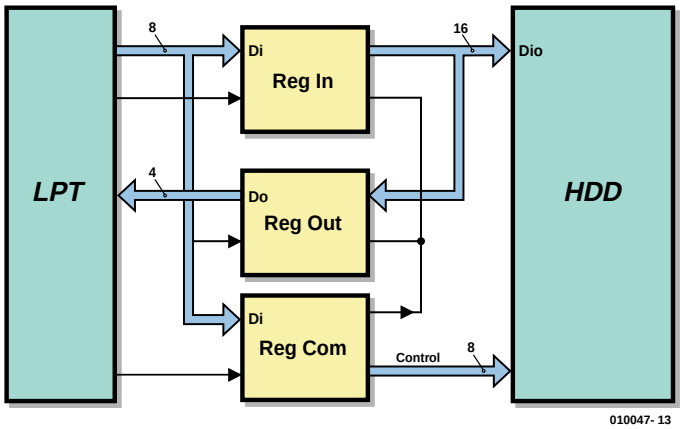


Figure 2. Block diagram of the LPT/IDE adapter.

Table 1. ATA Interface Pinning

Pin no.	Label	Description
1	-HRESET	RESET
2	GND	GND
3	HD7	Data bus bit 7
4	HD8	Data bus bit 8
5	HD6	Data bus bit 6
6	HD9	Data bus bit 9
7	HD5	Data bus bit 5
8	HD10	Data bus bit 10
9	HD4	Data bus bit 4
10	HD11	Data bus bit 11
11	HD3	Data bus bit 3
12	HD12	Data bus bit 12
13	HD2	Data bus bit 2
14	HD13	Data bus bit 13
15	HD1	Data bus bit 1
16	HD14	Data bus bit 14
17	HD0	Data bus bit 0
18	HD15	Data bus bit 15
19	GND	GND
20	N/C	key pin
21	DMARQ	DMA request
22	GND	GND
23	-HWR	I/O WRITE
24	GND	GND
25	-HRD	I/O READ
26	GND	GND
27	IORDY	I/O CHANNEL READY
28	SPSYNC:CSEL	SPINDLE SYNC or CABLE SELECT
29	-DMACK	DMA ACKNOWLEDGE
30	GND	GND
31	INTRQ	Interrupt request
32	-IOCS16	16 BIT I/O
33	HA1	ADDRESS BUS BIT 1
34	-PDIAG	PASSED DIAGNOSTICS
35	HA0	ADDRESS BUS BIT 0
36	HA2	ADDRESS BUS BIT 2
37	-CS1FX	CHIP SELECT 0
38	-CS3FX	CHIP SELECT 1
39	-DASP	DRIVE ACTIVE/DRIVE 1 PRESENT
40	GND	GND

host system on the rising edge.

HWR / I/O-write write signal for the I/O port address. The falling edge of this signal will latch the bus data (HD0 to HD7 or HD0 to HD15) into the hard disk register.

HA0, HA1, HA2 / Address bus Bit 0 to Bit 2. Registers or ports in the hard disk are selected with these address lines.

HRESET /Reset A low-level here will initialise the hard disk.

The diagram in **Figure 1** shows the timing for read and write cycles on the IDE bus. On the other side of the interface we have the parallel printer port using a standard 25-pin sub-D connector. The printer port has an 8-bit wide data bus together with six printer control signals (from PC to printer) and five printer status signals (from printer to the PC). Data to the LPT1 is sent by writing to address 378h (write only), the control register is (again write only) at address 37Ah and the status register is at address 379h (read only). A fuller description of the printer port addressing for both LPT1 and LPT2 is given in **Table 2**.

The printer port data transfer rate is less than 150 kBytes/s, but has the advantage that all the signal lines can be software controlled, this simplifies the design of both the control software and hardware.

Table 2. Signals and registers: LPT interface and LPT/IDE converter

Addresses LPTI/LPT2	Description					
378h/278h	Data Registers DATA7..DATA0 (write-only)					
379h/279h	Status-Register (read-only)					
	Bit	LPT Name	IDE2LPT Name	Status	Read Status	Function
	7	BUSY	LI3	I	0	DATA3 from converter
	6	SLCT	LI2	I	I	DATA2 from converter
	5	PE	LII	I	I	DATA1 from converter
	4	ACK	LI0	I	I	DATA0 from converter
	3	ERROR	—	I	I	—
	2	—	—	—	0	—
	I	—	—	—	0	—
	0	—	—	—	0	—
37ah/27ah	Control Register					
	Bit	LPT Name	IDE2LPT Name	Status	Read Status	Function
	7	—	—	—	0	
	6	—	—	—	0	
	5	—	—	—	0	
	4	—	—	—	0	
	3	SLCT_IN	RCWR	0	I	Write signal for Reg_COM
	2	INIT	RLWR	0	0	Write signal for Reg_IN from DATA7 - DATA0 to HD7 - HD0
	I	AUTO_FD	RHWR	0	I	Write signal for Reg_IN from DATA7 - DATA0 to HD15 - HD8
	0	STROBE	HRESET	0	I	Reset HDD

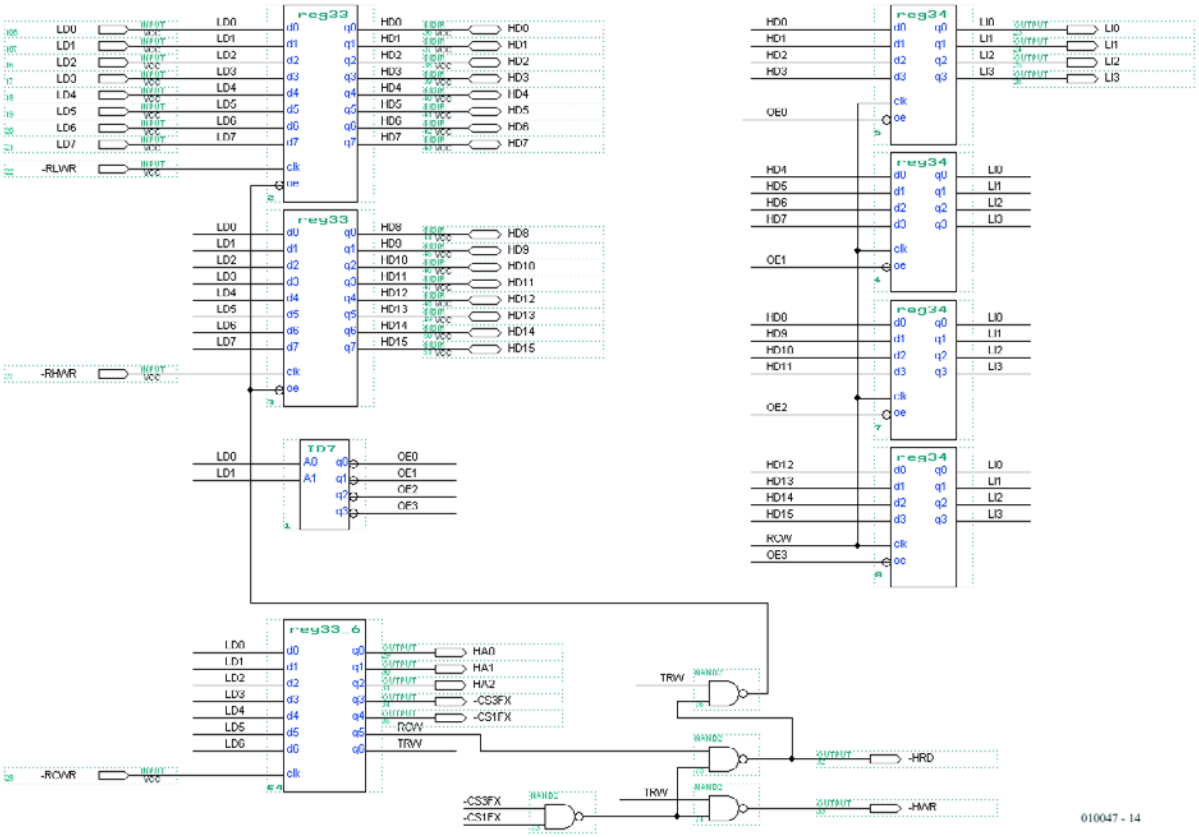


Figure 3. The chip circuit defined in AHDL and programmed into a CPLD.

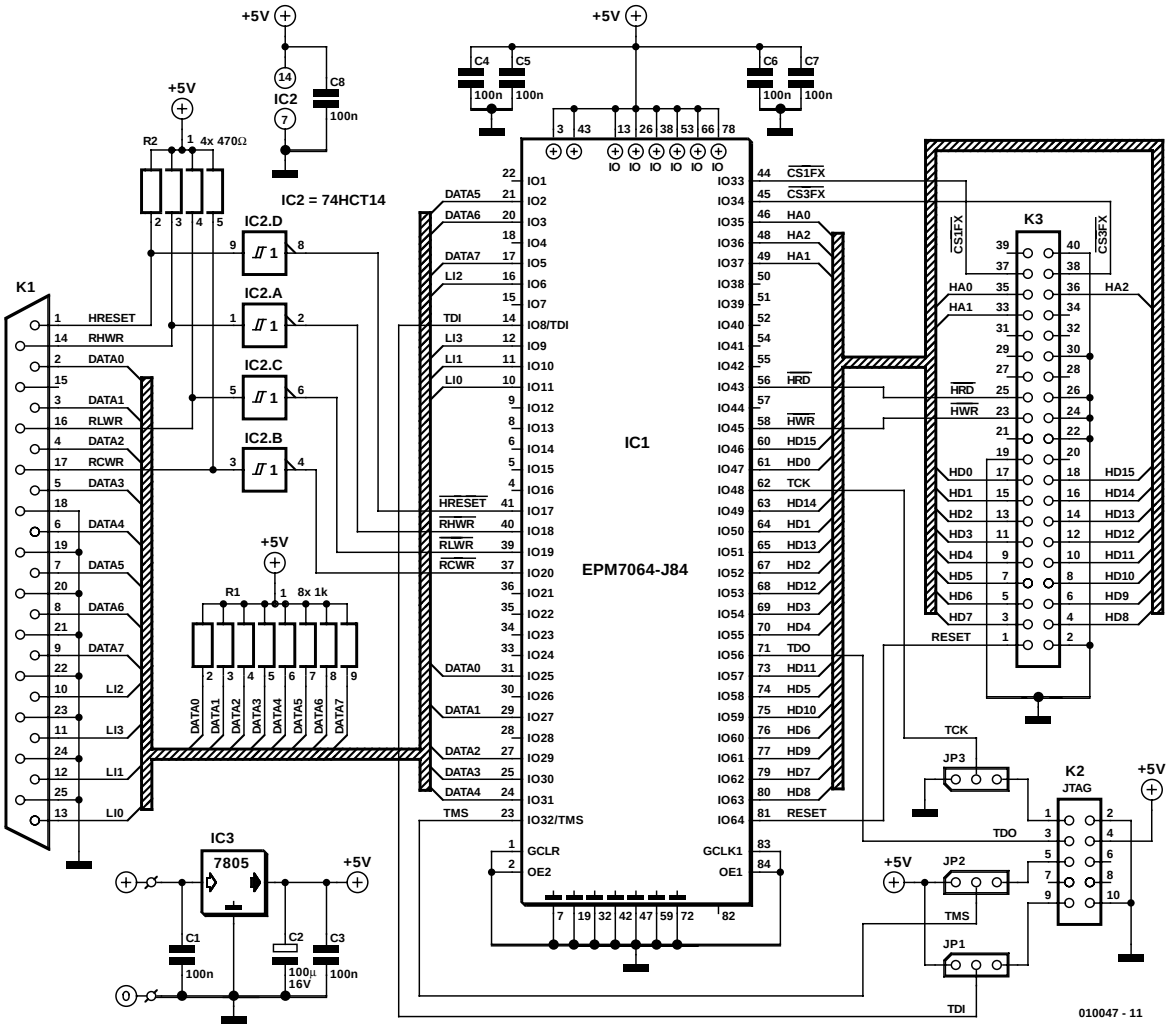


Figure 4.Circuit diagram of the LPT/IDE adapter. The JTAG connector allows in-circuit programming of the CPLD.

Table 3. Control Register signals in LPT/IDE converter

Bit	Name	Function	
7	—	—	
6	TRW	Write signal -HWR on IDE Bus.	Active level: 1.
5	RCW	Read signal -HRD on IDE Bus.	Active level: 1.
4	-CS1FX	CHIP SELECT 0.	Active level: 0.
3	-CS3FX	CHIP SELECT 1.	Active level: 0.
2..0	HA2 - HA0	ADDRESS BUS	

The adapter circuit

In the world of hard disk interfaces the data transfer rate of the printer port is not especially fast. It is therefore pointless to use all the features and speed enhancing techniques that are available on the IDE interface such as Direct Memory Access (DMA), 16 bit data transfer, standby and diagnostics. The IDE to LPT interface adapter is basically just a buffer between the two interfaces. **Figure 2** shows a block diagram of the circuit. Writing to the HDD is performed in two, eight bit wide bytes that are stored in the input register (Reg In) and sent to the HDD as 16 bit words. In the read direction the 16 bit wide data is stored in the output register (Reg Out) and read out in 4 bit wide nibbles to the LPT status register where it is read by the computer. Controlling the data transfer is performed by instructions sent to the command register (Reg Com) and are listed in **Table 3**. Additional control logic is used to prevent prohibited conditions on the IDE bus. (see **Table 4**).

The majority of the LPT/IDE adapter circuitry has been implemented in a CPLD. This entails defining the entire circuit of registers and logic in the Hardware Description Language AHDL. A pre-programmed Altera logic chip of the EPM 7064 family from Altera is available from the Elektor Electronics Readers Services and the source file for the chip is also available as a Free Download from the Elektor Electronics website. Figure 3 is the chip internal circuit diagram produced by this program. Figure 4 shows the circuit diagram of the adapter card where IC1 is the CPLD chip. In addition to this chip is IC2 which is a 74HCT14 inverting buffer with input hysteresis giving improved immunity to noise on the RHWR, RLWR, RCWR und HRESET inputs.

IC3 produces 5 V necessary for the adapter card from an external 12 V power source. (see 'plugging it all together'). Connecter K2 (JTAG) is used by IC1 and allows the chip to be re-programmed and de-bugged. The chip software 'MAX2plus' is available free of charge from the Altera website *(www.altera.com)* and the chip programming software (called 'BitBlaster') is also available from Altera but this time for a small fee. It is not necessary to purchase BitBlaster or use the socket K2 because IC1 is available pre-programmed from the Elektor Electronics (see Readers Services). The JTAG socket is used by any of the Altera CPLD 7000 family with the S suffix (here the EPM7064S). The three programming input signals to IC1 (TCK, TMS and TDI) should be jumpered to earth at JP1, JP2 and JP3 in normal circuit operation but for de-bugging and re-reprogramming the jumpers must be moved to connect these signals through to K2.

The double-sided PCB shown in Figure 5 can be ordered from Elektor Electronics. If you have the means to produce your own PCB then the necessary files are freely available from the Elektor Electronics website at *www.elektor-electronics.co.uk.*

COMPONENTS LIST

Resistors:
R1 = SIL array 8 x 1kΩ
R2 = SIL array 4 x 470Ω

Capacitors:
C1,C3-C8 = 100nF
C2 = 100µF 16V radial

Semiconductors:
IC1 = 7064LC84-15 (Altera), programmed, order code **010047-31**
IC2 = 74HCT14N
IC3 = 7805

Miscellaneous:
JP1,JP2,JP3 = 3-way pinheader with jumper
K1 = 25-way sub-D-plug (male), PCB mount
K2 = 10-way boxheader
K3 = 40-way boxheader
2 solder pins
Extension cable for PC PSU (see text)
Printer extension cable (1:1)
IDE cable
PCB, order code **010047-1**

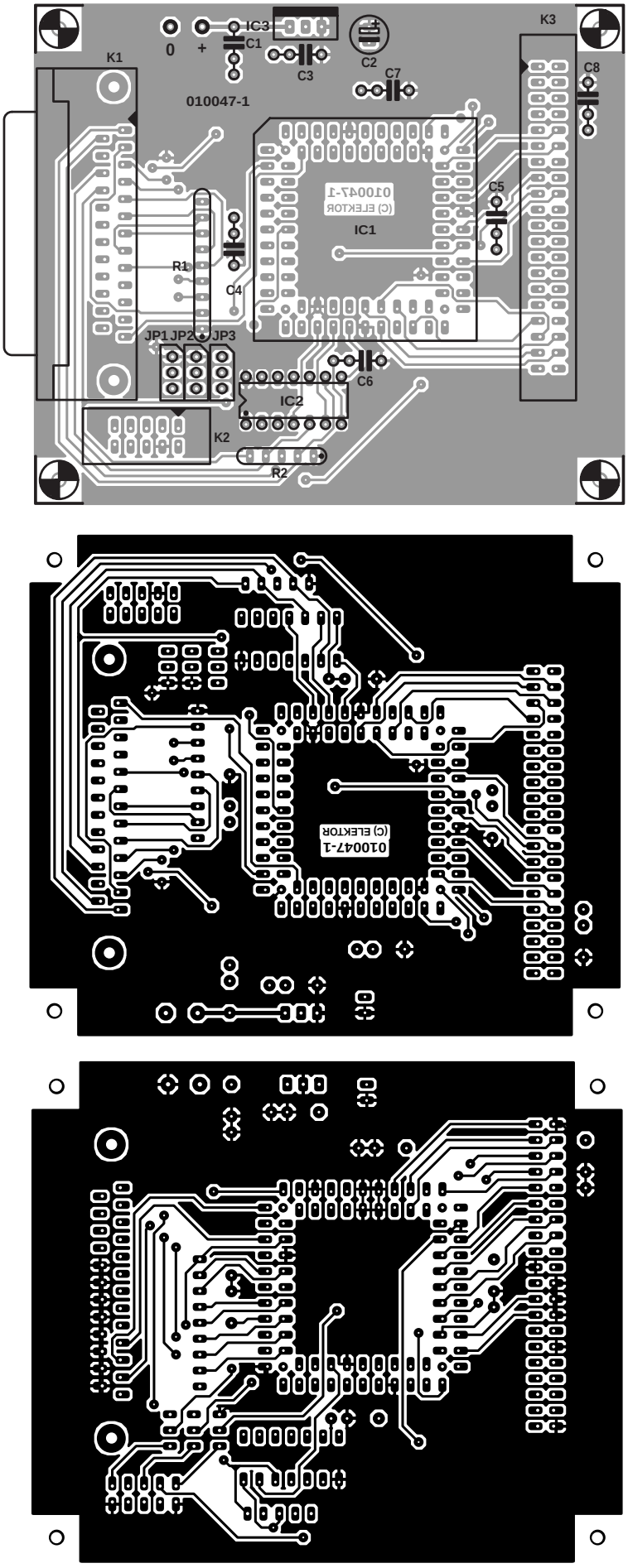


Figure 5. Layout of the double-sided PCB.

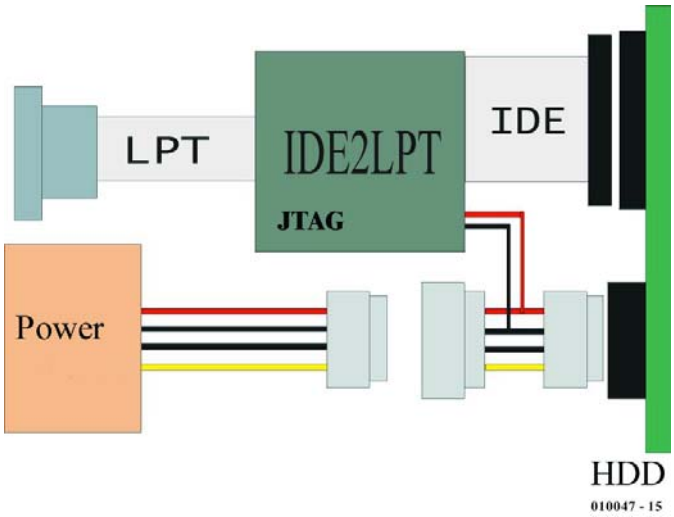


Figure 6. Wiring of the unit together with a PC power supply.

Software

As with any PC port a driver program is necessary to pass information in and out of the LPT/IDE adapter. Two versions of the 'LPT2IDE' driver program are available, one runs in DOS while the other runs in Windows. Firstly the program 'ide2lpt.exe' runs under MS-DOS from version 3.3 upwards. It is not unlike standard IDE driver routines, in fact we have adapted an existing IDE driver to read and write via the printer port. Additional test and search features have also been included and thanks go to Ewgenij Kuleschow for the driver source code. To install the driver a line must be added to the config.sys file:

```
device=[path]\ide2lpt.exe [options]
```

normally:

```
device=c:\lpt2ide\i2l4.exe
```

For different settings or tests the options can be added to the program name at the DOS command prompt e.g. the option /T (I2L4.EXE/T) for test.
Other options available are:

/H Help
/L:LPT The LPT port Address (378h for LPT1 and 278h for LPT2)
/G:SEC:HEAD Defines the HDD parameters (used for fixed drives operating in LBA mode): SEC indicates the number of sectors and HEAD the number of heads. The format is in decimal.
/LLBA mode
/V Request hard disk parameters.

The DOS driver uses FAT12, FAT16, BIG and Extended Partition while FAT32 is supported by the second driver that runs under Windows 9x.

Installation of the Windows driver in Win95 begins by first ensuring that you have the drivers stored on a floppy disk. From the Windows start up select 'Start' then 'settings', 'control panel' and 'Add new Hardware'. Now in the Hardware Wizard click 'Next' and select 'No' from 'Do you want Windows to search for new hardware?' now click 'Next', scroll down the list of controllers in 'Hardware types' and click-on SCSI controllers. Click 'Next' and select 'Have disk' now choose 'Browse' and find the folder on the floppy where the drivers are stored, select ide2lpt.inf and ide2lpt.mpd. Now click 'OK'. Driver installation will take place automatically but before you can use them it will be necessary to restart the computer.

If other Windows applications access the printer port while data is transferring to and from the hard disk it can cause a problem. It is therefore recommended that no other Windows programs are allowed to access the printer port while the IDE2LPT program is running.

Plugging it all together

Figure 6 shows all the interconnections between the parts of the complete adapter. A standard printer cable can be used to connect the portable hard disk unit to the LPT port of the PC. Alternatively a cable can be made-up using flat ribbon cable with crimp-on (IDC) D-type connectors. Ideally the cable length should be less than 30 cm. If you suspect that sources of radio frequency interference (RFI) are causing problems then the eight data lines DATA0 to DATA7 may benefit by connecting decoupling capacitors of between 12 to 22 pF from each data line down to earth at the adapter connector K1.

Power for the hard disk drive and adapter PCB is best provided by a separate PC Power Supply Unit (PSU). These units provide the necessary supply voltages with bags of current to spare. There should be no problem in finding such a unit especially if you are cannibalising a redundant PC system. Alternatively a new unit can be purchased quite cheaply. Power connections between the HDD and the PSU can be made with a standard computer internal power expansion cable with two wire taps taken off to provide +12 V and 0 V to the adapter card. These two wires should be soldered to the pins labelled + and 0 next to C1 on the PCB.

Gigabytes to go

Whether you are purchasing a new hard disk drive for this project or recycling a unit from a redundant PC system, this design gives even low-spec PCs (those without CD or CD-RW drives) the ability to store, back-up or transfer large files to and from other PC systems.

(010047-1)

Postscript:

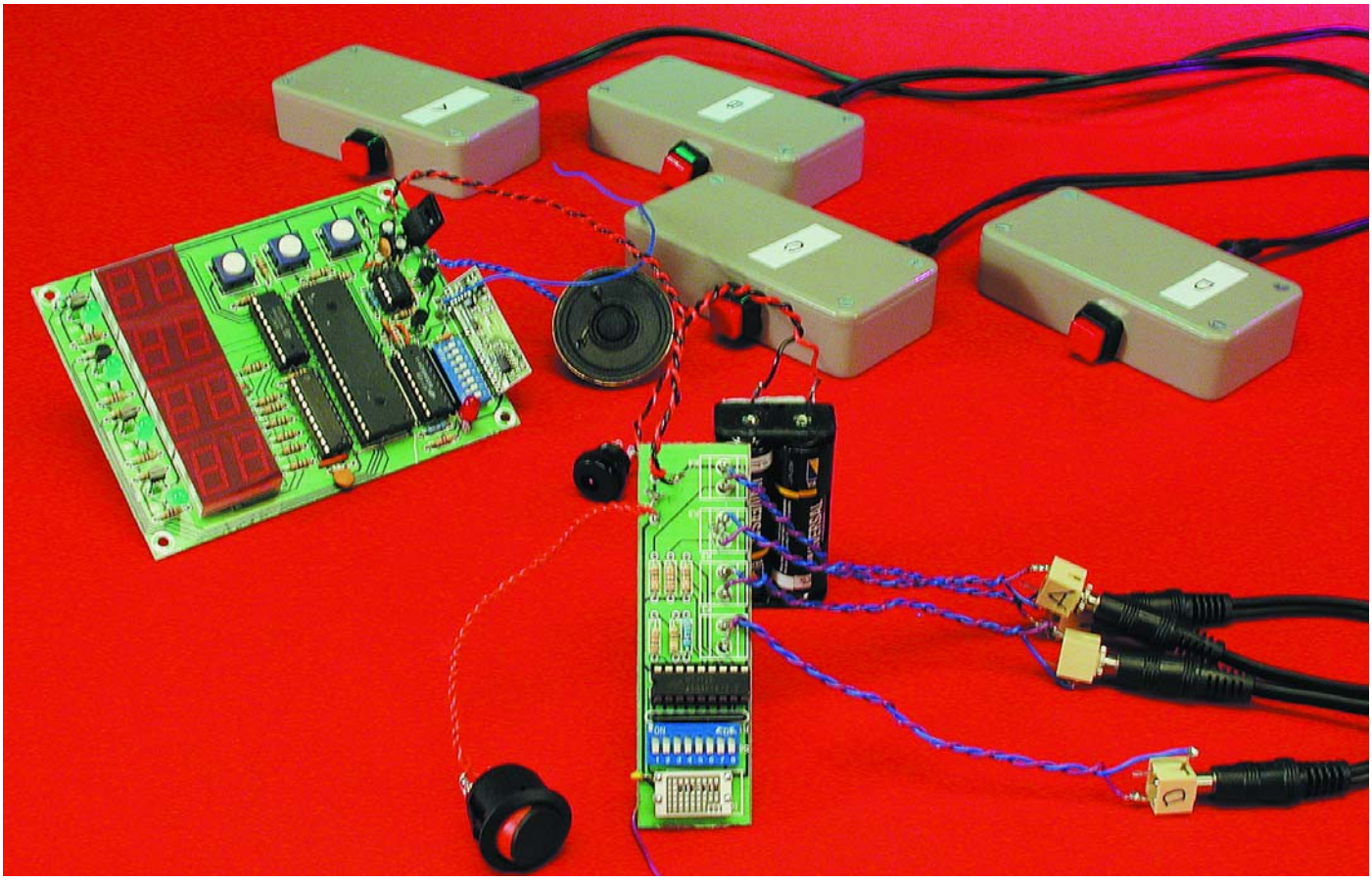
The interface described in this article was built on a prototype board (see photos) and tested with a number of PCs and hard disk drives. These test runs were completed without problems.
Just before the closing date of this issue, we received information from the author regarding possible timing problems with a number of older PCs. According to the author, such problems occur in rare cases, only on motherboards containing a Pentium I processor, and depending on the I/O chip set fitted.

Quizmaster

radio-linked!

By Andy & Rose Morrell

The Quizmaster described here is a four-player version with an extra player to operate the 'box' and to ask the questions etc. No wiring required between the player and quizmaster desks!



Knowledge-based quizzes of the 'University Challenge' variety are extremely popular, and have spawned many derivates that can be seen on TV every day. Where many people would be perfectly capable of organizing a quiz, persuading players and a quizmaster to participate, drawing up a list of questions (geared to the contestants' specific interest or level of education) and arrange a suitable venue, alas, building a 'machine' to detect the first key press, display the score, etc. often poses insurmountable problems. The present design should be easy to build by anyone with reasonable skills in DIY electronics.

One of the major problems in mimicking a quiz 'studio' is the cabling. The Quizmaster project discussed here overcomes that difficulty by means of a wireless link between the contestant desks and the quizmaster desk.

Here's how the system works. Immediately after the quizmaster asks the question he/she presses the start timer button. The player who

hits his/her button first will instantly lock out the other three so further presses of the buttons by any of the other contestants will have no effect. To indicate which player was first, that player's LED flashes and a siren is heard.

If none of the players press their button, a timeout will be reached in approximately 10 seconds. The four LEDs will then start to flash simultaneously until the quizmaster presses the button labelled 'reset timer'.

If the correct answer is produced by one of the players, all the quizmaster has to do is press the 'correct' button to add 1 to their score, or the 'wrong' button, which obviously leaves the score unaltered. Then, the next question is asked.

The Quizmaster project offers ease of use for players as well as the quizmaster, which allows everybody to concentrate on the questions, which is essential for the overall success of the quiz if an audience is present. The quiz night at the local pub is another, hopefully hilarious, matter.

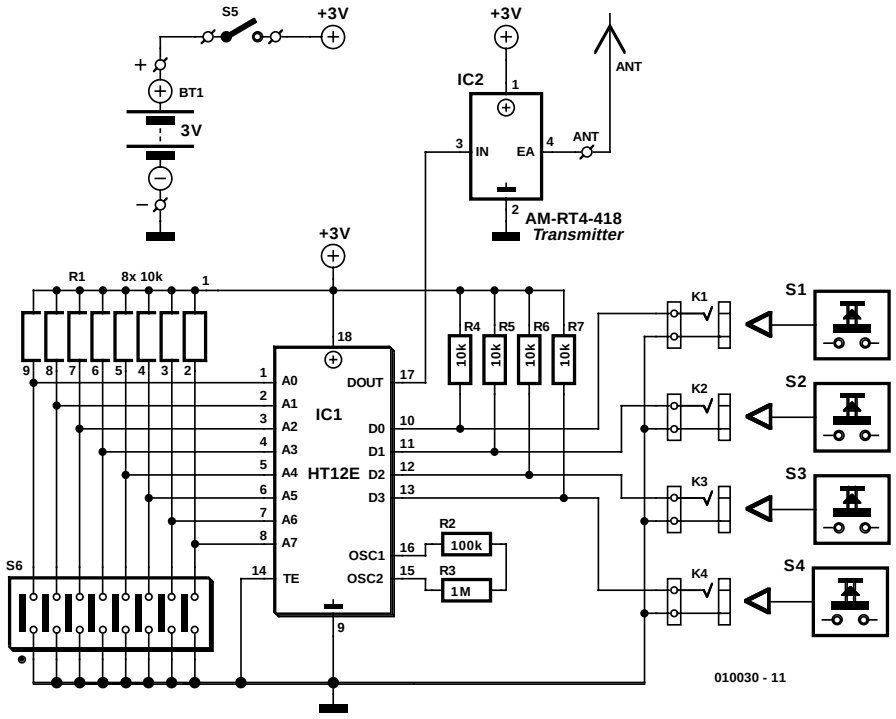


Figure 1. Circuit diagram of the Quizmaster central transmitter to be fitted in or near the players' desk.

How it works

Just as the transmitter and receiver unit discussed further on, two other components in the circuit are always encountered as a 'pair'. The HT12D and HT12E integrated circuits from Holtek have been used in a previous *Elektor Electronics* project, see 'Radio-Linked Caller ID' System in the January 2001 issue. Datasheets may be found at the manufacturers' website [1]. The circuit diagram of the transmitter does not amount to much, see **Figure 1**.

The HT12E encoder chip, IC1, has its four inputs D0-D3 connected to four pushbuttons in small individual boxes via 2.5-mm jack sockets and leads. The HT12E has its transmit enable (TE) pin wired to ground so it is constantly transmitting when the power is on (this will not affect the receiver). The whole transmitter/encoder circuit runs off two AAA or AA batteries. The output of the encoder chip is wired to the 418 (or 433) MHz transmitter which operates a good range on the same 3-V battery set. A 17-cm (7 inch) length of wire is used for a quarter-wave aerial.

The (one-way) communication between the HT12E and HT12D is encoded. The same 8-bit code (word) has to be set on the DIP switch blocks (S5, S6) for the receiver to recognize the transmitter.

All of the intelligence required to control the Quizmaster circuitry is crammed into a single microcontroller of the PIC variety. As you can see from the receiver circuit diagram in **Figure 2**, not much is needed to enable the PIC to handle all input/output functions, where 'input' really means reading data from the 433-MHz receiver and scanning the quizmaster's desk buttons, while 'output' boils down to enabling the siren and driving the LED indicators and displays. A run-of-the-mill 5-volt power supply completes the circuit.

Pressing the START button allows the PIC to recognise if the HT12D receiver chip receives a signal from player A to D. The PIC chip updates a software counter by interrupt every millisecond or so. If no data is received from the central transmitter, (i.e., none of the players knows the answer to the question) the counter will produce a timeout after about 10 seconds. The PIC will then flash all four LEDs labelled Player A-D (D3-D6) until the quizmaster presses the WRONG/RESET button.

When the correct answer is produced within the available time slot of about 10 seconds, the quizmaster will press the CORRECT button. The siren chip, IC6, fits in perfectly for the sound. PIC pin port line RD7 is pulled low, and via the BC327 PNP transistor (T5) turns on the UM3561 siren chip. A BC548 transistor at the UM3561 output enables a small 64-Ω loudspeaker to be driven so that the siren sound is clearly heard. Three popular siren sound types are available and may be selected by switch S4.

As already mentioned, communication between the player's desk(s) and the quizmasters desk is wireless. Here, a low-cost licence-exempt transmitter/receiver pair from RF Solutions is used. The units originally used by the authors operate in the 418 MHz band. Licence-exempt means that the transmitter can be used freely by anyone as long as it is not modified or connected to a directional antenna. The transmitter and receiver may also be 433-MHz versions since in the UK 433.92 MHz is also allocated for short-range licence-exempt communications. The use 433.92 MHz band has been harmonized across Europe, and it is expected that the

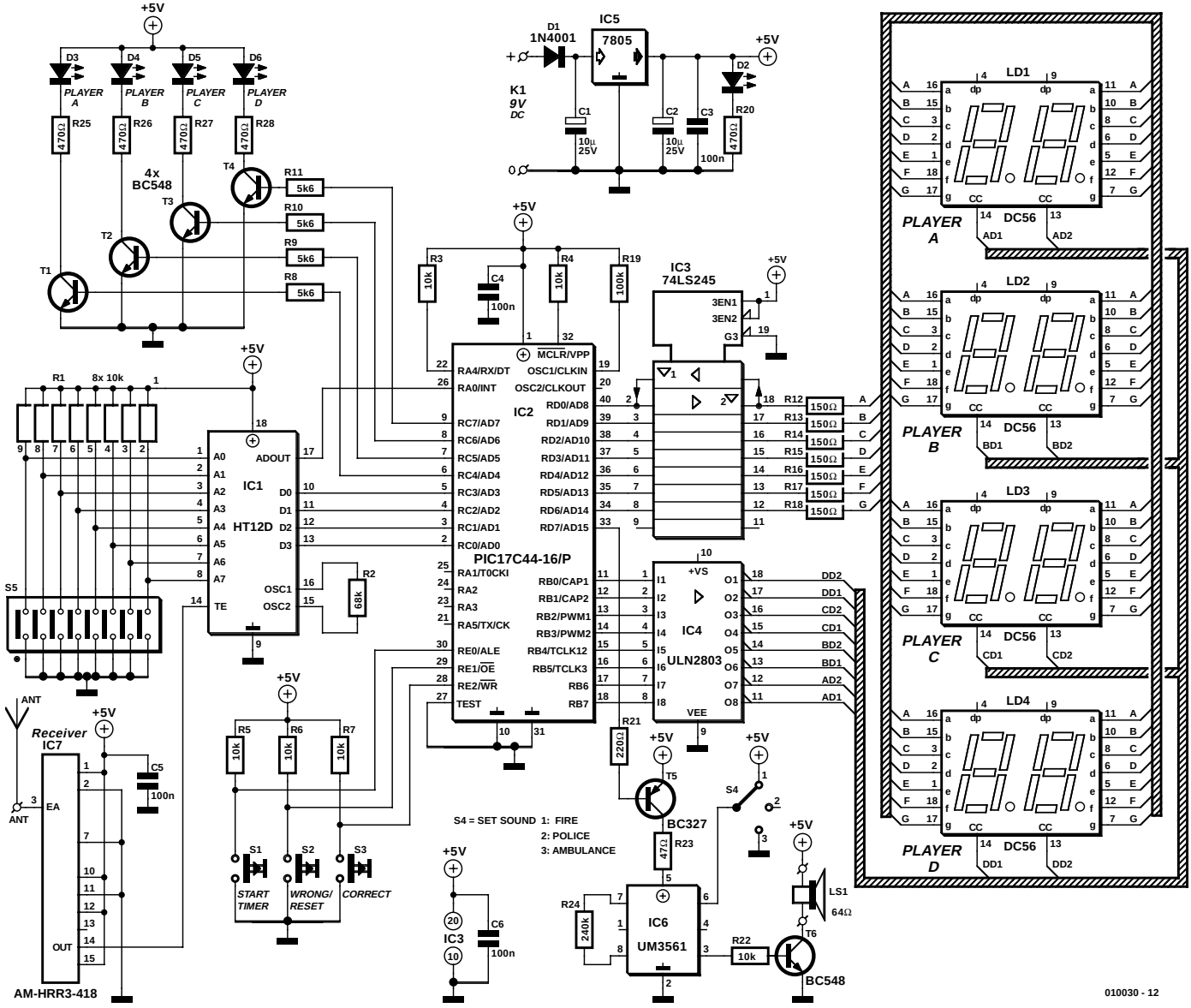


Figure 2. Circuit diagram of the Quizmaster central receiver. With a PIC in control, hardware is down to a minimum!

418 MHz allocation will be gradually phased out in the UK. Datasheets of the AM-RT4-418/433 and AM-HRR3-418/433 may be found on the Internet [1].

Returning to the main unit installed in the quizmaster's desk, the PIC17C44 chip drives the display units in multiplex mode via two driver ICs, a ULN2803 for the common-cathode (CC) lines and a 74LS245 for the digit (anode) lines. There are four 2-digit 7-segment displays. Four BC48 transistors (T1-T4) drive the 5-mm 'Player' indicator LEDs.

To close off the discussion of the circuit operation, the power supply is conventionally based on a 7805 3-pin voltage regulator that steps down the unregulated DC input voltage (max. 9 V) at K1 to 5 V as required by the receiver circuitry. LED D2 acts as the on/off indicator. In view of electrical safety we recommend the use of a mains adaptor with an output current rating of about 300 mA or better.

The control program

Since the source code file for the program that runs inside the PIC is far too large to print in this article, it available either on floppy disk, order code **010030-11** (at a nominal charge), or as a free download from the *Elektor Electronics* website at www.elektor-electronics.co.uk. If you have a suitable PIC assembler, you may decide to use the assembly code file to generate your own object code and burn your own PIC17C44 for this project. The commented lines should be easy to understand, and represent educational value even if you do not intend to build the Quizmaster. A ready-programmed PIC is also available from the Publishers, the order code is **010030-41**.

Construction

The transmitter and the main unit are built on printed circuit boards of which the layout is shown in Figure 3. Note that the two boards have to be separated by cutting. The photographs in Figures 4, **5** and **6** give a good idea of how the boards should

be stuffed.

The receiver board contains all components including the displays, and may have to be mounted in a position where the readout (score displays and Player LEDs) can be clearly seen by everyone. This may necessitate fitting the quizmaster's control buttons off the board and on a separate little panel.

We recommend using sockets for all ICs on the boards. In particular the programmed PIC represents considerable value!

The transmitter and receiver antenna are both 17-cm long pieces of straight wire which for optimum radiation (transmitter) and reception (receiver) should not be enclosed in a metal case.

Voltage regulator IC3 will not run excessively hot but to remain on the safe side you may still decide to fit it with a small heatsink.

The 64-Ω loudspeaker may be mounted in an external box or, space allowing, in the central receiver case.

Any code word you like may be used to

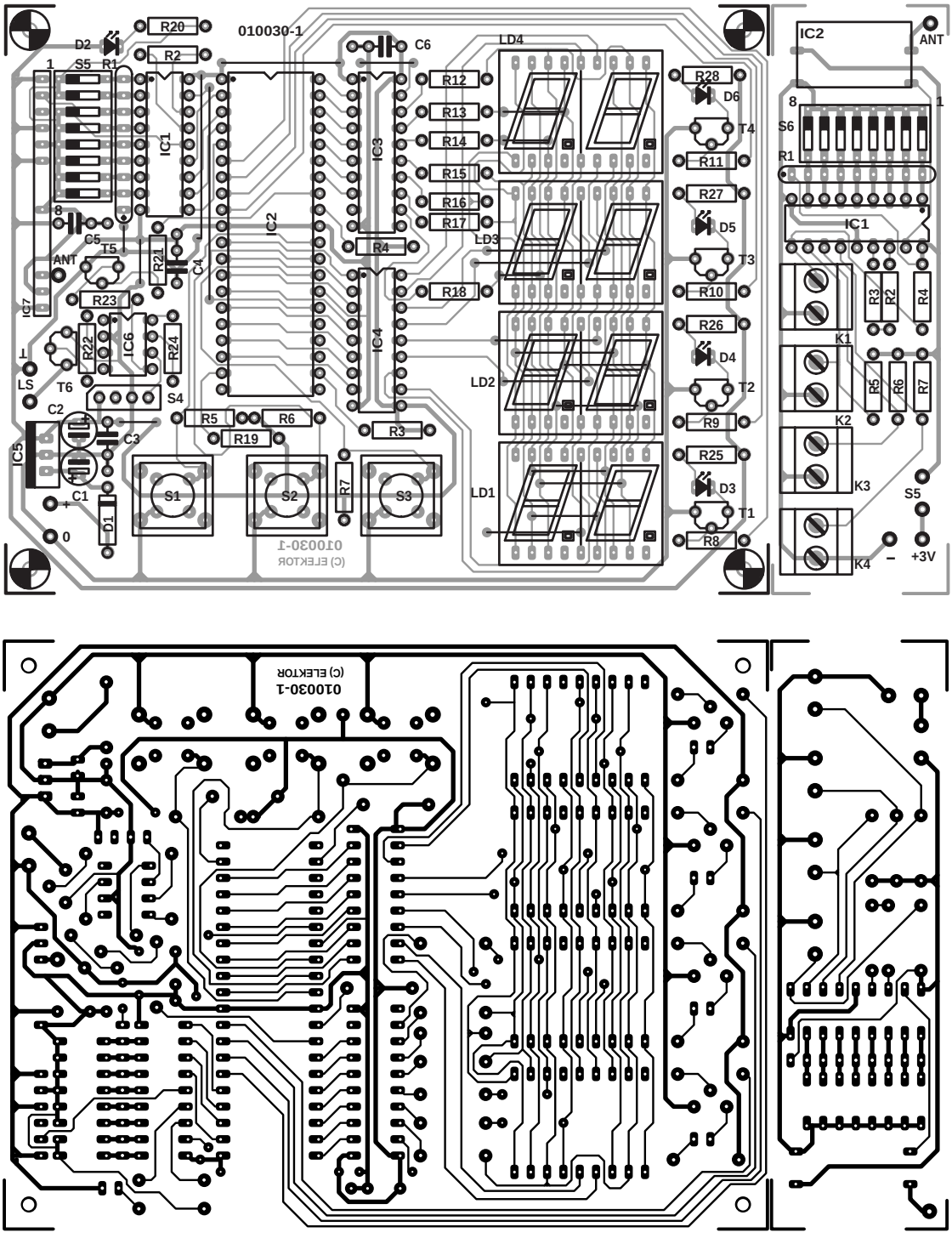


Figure 3. Copper track layout and component mounting plan of the combined receiver & transmitter board (separate by cutting, before populating). This board is available ready-made.

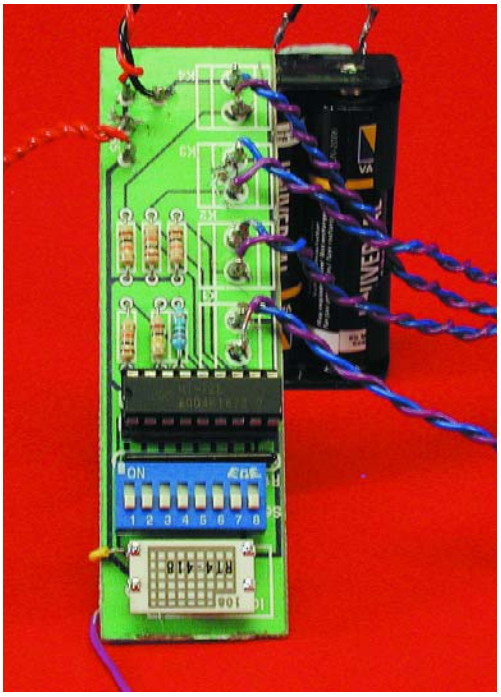


Figure 4. Completed transmitter board.

encode the transmitter, as long as you (1) set the same word on the receiver and (2) the code word is exclusive to other transmitters in an area of about 300 m.

(010030-1)

Web Addresses

[1] HT12E/HT12D datasheets: www.holtek.com.tw

[2] AM-HRR3-418/433 and AM-TR-418/433 datasheets: www.rfsolutions.co.uk

COMPONENTS LIST

Transmitter

Resistors:

R1 = 8-way 10kΩ SIL array
R2 = 100kΩ
R4..R7 = 10kΩ
R3 = 1MΩ

Semiconductors:

IC1 = HT12E (Farnell # 562415)
IC2 = AM-RT4-433 (RF Solutions)*

Miscellaneous:

K1-K4= jack socket, 2.5mm, mono, PCB mount
S1-S4 = pushbutton (external), 1 make contact
S5 = on/off switch
S6 = 8-way DIP switch
BT1 = 3-V battery with holder (or 2 off AA/AAA battery)

Capacitors:

C1,C2 = 10µF 16V
C3-C6 = 100nF

Semiconductors:

D1 = 1N4001
D2-D6 = LED, green, 5mm dia.
IC1 = HT12D (Farnell # 563250)
IC2 = PIC17C44-16/P, programmed, order code **010030-41**
IC3 = 74LS245
IC4 = ULN2803AP (Farnell # 3187032)
IC5 = 7805
IC6 = UM3561 (UMC) (Rapid Electronics # 82-0704, tel. 01206 751166)
IC7 = AM-HRR3-433 (RF Solutions) *
LD1-LD4= 2-digit 7-segment LED display, common cathode, e.g., Kingbright DC56-11EWA
T1-T4,T6 = BC548
T5=BC327

Receiver

Resistors:

R1 = 8-way 10kΩ SIL array
R2 = 68kΩ
R3-R7,R22 = 10kΩ
R8-R11= 5kΩ6
R21 = 220Ω
R12-R18 = 150Ω
R19 = 100kΩ
R20 = 470Ω
R23 = 47Ω
R24 = 240kΩ
R25-R28=470Ω

Miscellaneous:

K1 = mains adaptor socket, PCB mount
S1,S2,S3= pushbutton, 1 make contact, type CTL3 (Multimec) or DR-6
S4 = 3-way 1-pole switch
S5 = 8-way DIP switch
LS1 = 64Ω loudspeaker
PCB, order code **010030-1** (combined board for receiver and transmitter)

* 433 MHz transmitter/receiver modules available as a pair from Maplin, order code VY48C.

Figure 5. Completed receiver board. ▽

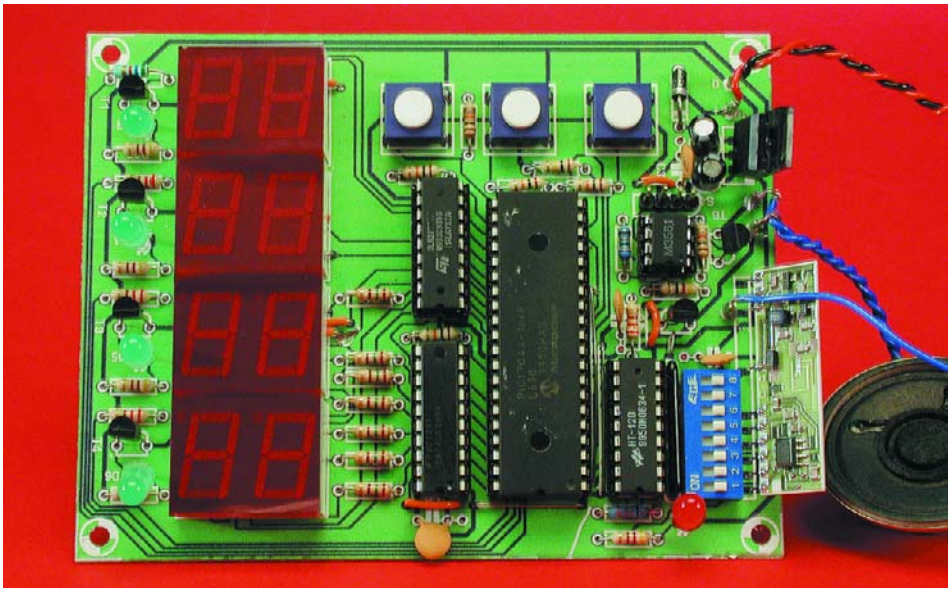
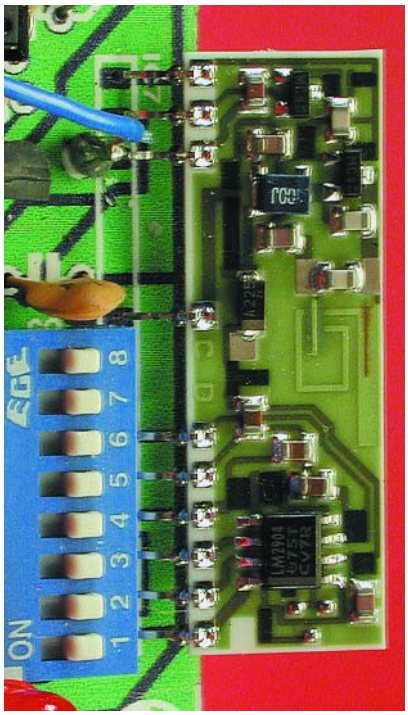


Figure 6. Close-up of the RF Solutions AM-HRR3-433 radio receiver module fitted on to the main board. ▷



ISAC (4)

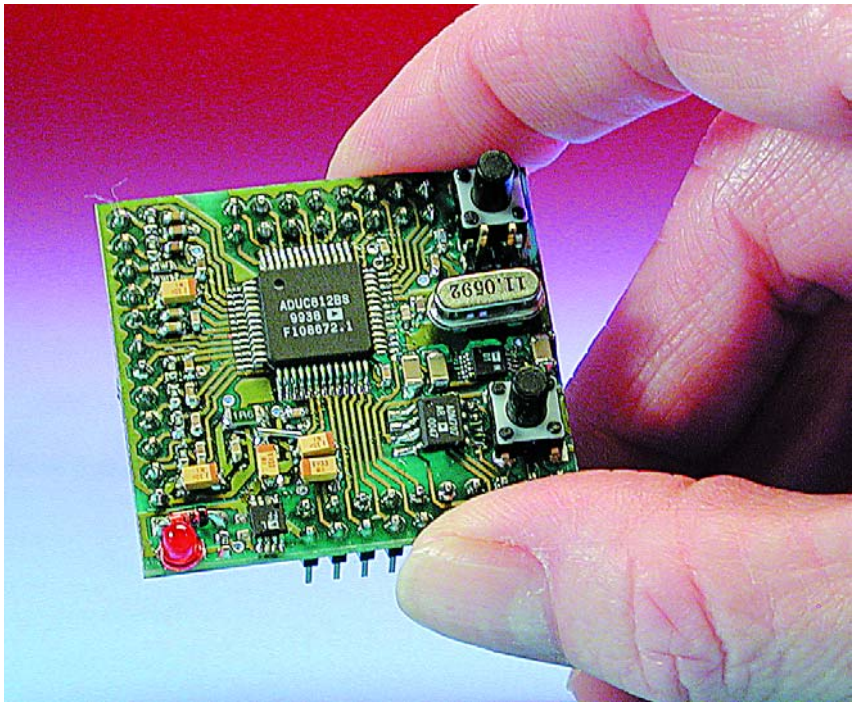
part 4: basic applications

By Prof. B. vom Berg, P. Groppe and M. Müller-Aulmann

In this fourth instalment of the series we will look at our first applications of the ISAC cube and the simple motherboard, which will give a good introduction to the possibilities of the device. Complete example programs developed for these applications, along with extra information, are available for download from the *Elektor Electronics* website.

Before we present the individual example applications, a few initial remarks are in order regarding the example programs and their use.

– The applications and demonstration programs are written in C51 code in such a way that (with one exception) they can be used with the restricted version of µVision2 (maximum code size of 2 Kbytes, no floating-point arithmetic). For larger application programs the full version of µVision2, or another 8051 tool, must be resorted to.
– In order that something 'visible' is produced, all the programs guide the user via, and produce output on, the PC's monitor. That means that you will need to run a terminal program on your PC. We have assumed that you will be using HyperTerminal, since this program is included with Windows. You can, however, use any other terminal program: the required communications parameters are 9600 baud, 8 data bits, no parity, 1 stop bit.
– Before starting a program on the ISAC cube, you should start up your chosen terminal program and then press the reset button on the cube.
– **Very important:** if you have been using HyperTerminal and then want to download a new program into the ISAC cube using the serial downloader, you must first clear the existing connection with HyperTerminal (using 'Disconnect' on HyperTerminal). Otherwise HyperTerminal will keep control of the COM port and the serial downloader will not be able to access it. An apparently meaningless error message will appear if the serial downloader is unable to find the ADuC812. In this case you will have to cancel the error message, release the connection in HyperTerminal, and then relaunch the downloader.
– Communication between the microcontroller core and the on-chip peripherals uses the 8051-style Special Function Register (SFR) (**Figure 1**). Further information on the meaning of the individual bits within the SFR can be found in the ADuC812 data sheet or in specialist 8051 books.



D/A converter application

Outputting different signals:
With the aid of program **dac_1.c** two different signals (a sawtooth and a squarewave) can be output using the D/A converter at two different resolutions (8 bits and 12 bits). Con-

nect an oscilloscope to the relevant outputs to verify the results.

The reference voltage used here is the digital power supply voltage V_{DD} (=+5 V), and so the output voltages produced lie in the range 0 to +5 V.

A/D converter application

Reading the voltages on the input channels:

On starting up program **adc_1.c** you will be asked to supply the number of the channel to be read, where you have the following possibilities:

– number 0-7: external measurement channels ADC 0 to ADC 7.

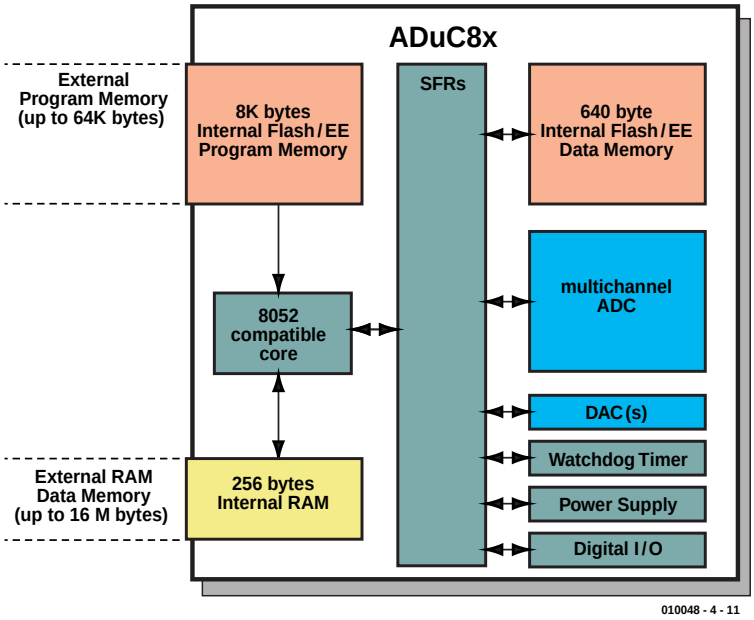


Figure 1. The SFR register.

– number 8: channel connected to the on-chip temperature sensor (see next example).

When you have entered a number, the voltage present at the selected input channel is converted once per second and the result displayed as a decimal and as a hexadecimal value on the PC's monitor. Readings are stopped by pressing the reset button, and a new input channel can then be chosen.

Note: The A/D converter has a resolution of 12 bits and can operate with two different reference voltages:

1. Using the internal 2.5 V reference voltage. In this case 1 LSB corresponds to a voltage step of 610 μV.
2. Using an external reference voltage in the range +2.3 V to +5.0 V, connected to the V_{Ref} pin. In view of the 12 bit resolution, this voltage should be specially filtered and regulated. If, for example, V_{Ref} is connected to AV_{DD} supply (+5 V), 1 LSB will correspond to a voltage step of 1.2 mV.

The input voltage to be measured must **always** lie between 0 V and the reference voltage used (whether internal or external), i.e. in the range 0 to +2.5 V when using the internal reference voltage or in the range 0 to +5 V if an external 5 V reference is used.

Measuring the internal die temperature and the ambient temperature:

The ADuC812 includes an on-chip temperature sensor that produces a temperature dependent voltage that is connected to the ninth channel (i.e., channel number 8) of the A/D converter. The datasheet gives the following information about this sensor: the output voltage is 600 mV at a temperature ϑ = 25°C, and the

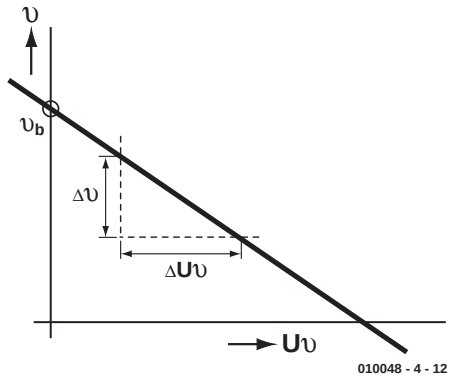


Figure 2. The linear relationship between voltage U_ϑ and temperature ϑ.

temperature dependence is –3 mV/°C. If the temperature rises by 1°C the temperature-dependent voltage will drop by 3 mV and vice versa. But there is a **very important point** to note here: these figures are only **typical values** and vary from device to device. If you compare ADuC812s from different production batches, you will find rather different values. Analog Devices do guarantee, however, that the temperature-dependent voltage is highly linear in die temperature.

This relationship is characterised by a (linear) equation $U_\vartheta = f1(\vartheta)$ (or $\vartheta = f2(U_\vartheta)$) (**Figure 2**). For the user, this means that two calibration readings have to be taken to establish the parameters of the linear equation before meaningful temperature values can then be obtained. If you then have an ADuC812 from the same production batch (which is usually the case if they come from the same tube), nothing needs to be changed. If, however, you buy ADuC812s from different batches, this calibration will have to be repeated several times over. In the worst case a different set of parameters for the linear relationship will have to be obtained for each

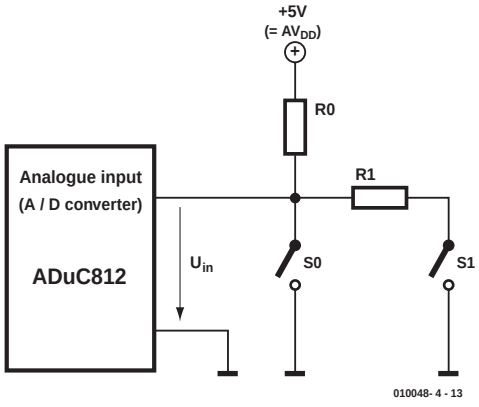


Figure 3. Voltage divider at the input to the A/D converter switched by push-buttons.

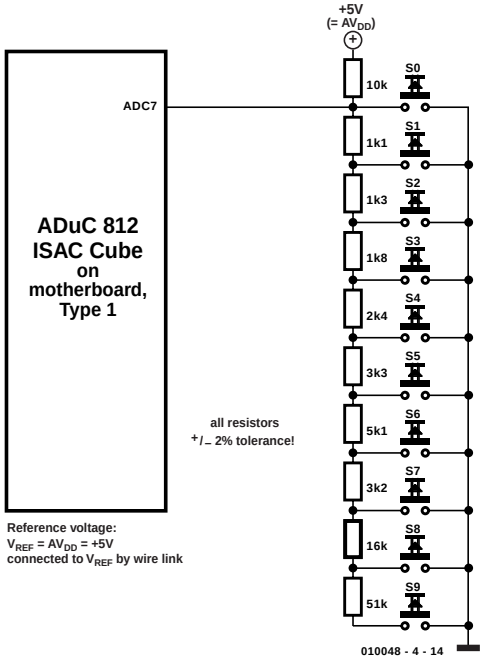


Figure 4. Series of push-buttons connected to an analogue input pin.

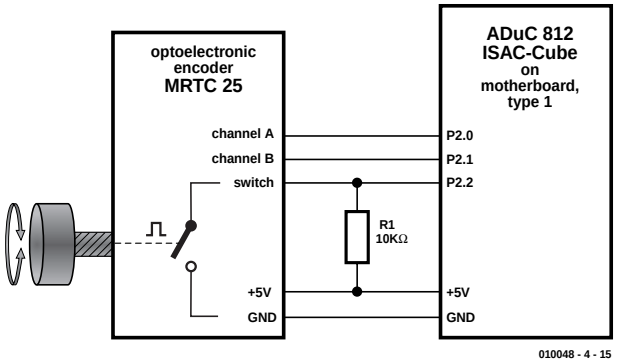


Figure 5. Connecting an opto-electronic shaft encoder.

Table 1

Characteristics of the MRTC25 opto-electronic shaft encoder

Electrical characteristics:	
Resolution (pulses/360°)	25
Channels	2 (A and B; 90° phase offset)
Supply voltage	5 V DC ±0.5 V
Current consumption	20 mA
Output signals	TTL-level squarewave; internal 10 kΩ
pull-up	
Push-button	5 V, 10 mA (non-inductive load)
Operating life	> 1,000,000 cycles

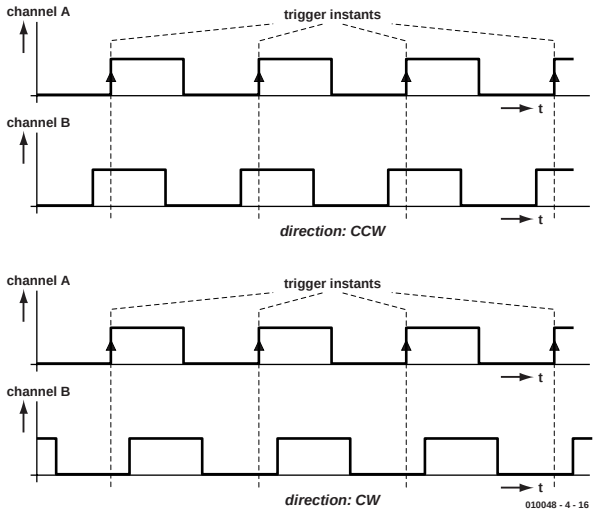


Figure 6. Timing diagram for the two directions of rotation.

ADuC812. The program **temp812.c** has been developed to help in the calculation. When the program is downloaded you will also find instructions on how to determine the parameters of the linear relationship, on **measuring the ambient temperature**, and on deriving the settings programmed into temp812.c. Observe the instructions given on how to use the program.

Push-buttons connected to an analogue input:

The basis for this method of connecting push-buttons, which allows an entire keyboard to be connected to one analogue input pin of the A/D converter, is a 'multiply switched voltage divider'. We start with a simple voltage divider (**Figure 3**).

With this circuit we have:

No button pressed: U_{in}=5 V
Button 0 pressed: U_{in}=0 V
Button 1 pressed: U_{in}=5 [R1/(R0+R1)] V

If the two resistors R1 and R0 have the same value, for example 1 kΩ, we have U_{in}=2.5 V.

For each button pressed, a different and distinguishable voltage appears on the analogue input and is converted by the A/D converter. In this way, for each button that might be pressed we obtain a different digital value and hence we can identify uniquely the button pressed. There is a further advantage to this circuit: if more than one button is pressed simultaneously, only the button nearest the analogue input will be recognised. In other words, we can build a keyboard in which the individual keys are prioritised. If, for example, button 1 is already pressed and subsequently button 0 is pressed, U_{in} will immediately be pulled to the voltage corresponding to button 0 (0 V). Thus button 1 is overridden and will no longer be recognised, while button 0 has the highest priority and will always be recognised. When button 0 is subsequently released, the other buttons can then be read.

In order to use a large number of buttons close-tolerance resistors are required, as well as a well-regulated power supply to the voltage divider. **Figure 4** shows a practical circuit for a keyboard with ten push buttons.

A small disadvantage of this idea

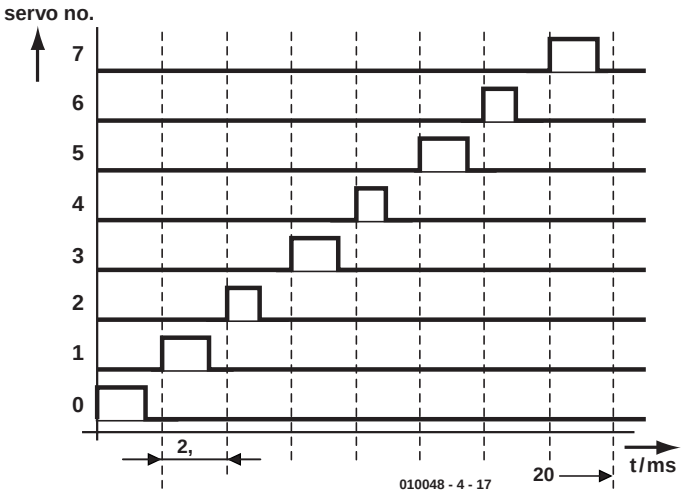


Figure 7. Timing diagram for controlling a servo.

should not be ignored: the buttons can of course only be read while they are pressed: there is no automatic keyboard buffer. This omission can, however, be rectified by dedicated software. A suitable program for reading a 10-key keyboard connected to analogue input pin ADC7 of the ADuC812 is provided in program **ana_tast.c**. When downloading this program you will also find a table showing the relationship between the button pressed, the input voltage U_{in} and the converted digital value, safe push button reading interval and button priority level (0=highest priority).

Digital I/O port application

Use of an opto-electronic shaft encoder:

An opto-electronic shaft encoder makes a simple and elegant input device. Here we use a type MRTC25 from Megatron, which offers two output channels (A and B) for count and direction indication (clockwise/counterclockwise). It also provides a contact that closes when the shaft is pressed down.

Connection to the ISAC cube or to the simple (type 1) motherboard proceeds as shown in **Figure 5**. The timing diagram for clockwise and counterclockwise rotation is shown in **Figure 6**. Electrical characteristics of the shaft encoder are given in **Table 1**. Program **drehge_1.c** shows how to interface to this interesting input device.

Control of eight servos:

Using the eight digital outputs of port 2 and a little software, in this case written in assembler, it is easy to control eight servos. For each servo a control pulse of between 0.9 ms and 2.1 ms must be generated every 20 ms (**Figure 7**). The simplest approach is to divide the 20 ms period into eight subintervals and then drive each servo in turn from the appropriate output on port 2 to the desired position. Program **servo.asm** generates the 2.5 ms intervals using Timer 2 in auto-reload mode and generates the required servo pulse durations using Timer 0 as a 16 bit timer.

Interval: 20 ms / 8 = 2.5 ms (Timer 2)
Servo pulse width: 0.9 ms to 2.1 ms (Timer 0)

In the Timer 2 interrupt service routine Timer 0 is started with the pulse width required for the servo, and the appropriate pin of port 2 is set. In the Timer 0 interrupt service routine all port 2 pins are cleared and the timer stopped.
In order to achieve exact timing in the main program, an extra software delay is employed in the Timer 2 interrupt service routine. This short program is readily tested using the ADuC812 simulator (**Figure 8**).

Of course, applications for the ADuC812 are limited only by your imagination. With a little effort on the software side up to 24 servos could be controlled using the 24 digital outputs. When used in combination with the I²C interface we can even construct a replacement 'I²C Servo Interface' (*Elektor Electronics*, September 2001) — and our replacement offers a resolution of more than 1000 steps.

The servos could also be controlled by sensors via the eight analogue inputs — which is exactly what this series (ISAC — Intelligent Sensor/Actuator Controller) is about.

(010048-4)

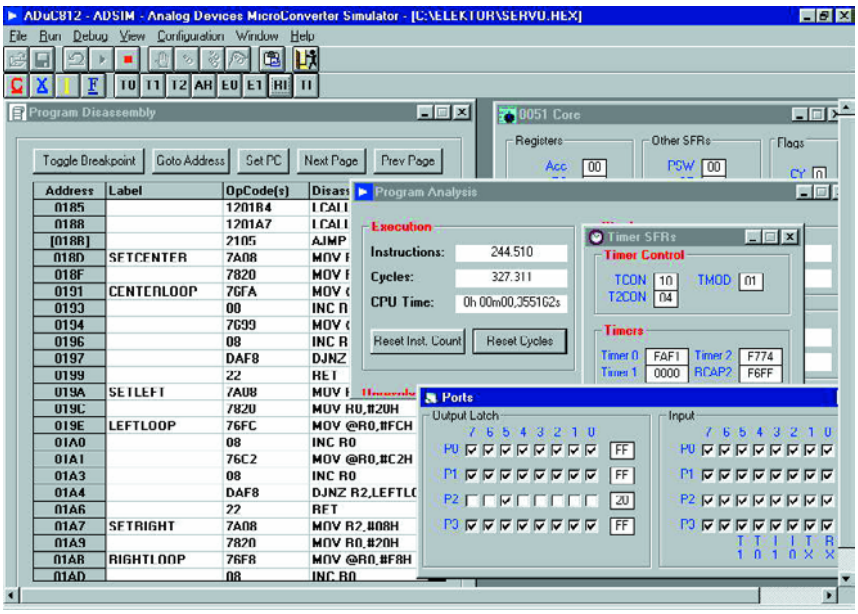


Figure 8. servo.hex in the ADuC812 simulator (pulses output for servo 5 on pin P2.5).

Component source:
MEGATRON Bauelemente,
Hermann-Oberth-Strasse 7,
D-85640 Putzbrunn, Munich, Germany.
Tel.: +49 89 46094 146

Digital Benchtop Power Supply (3)

part 3: the software

Design by R. Pagel

The digital benchtop power supply is controlled by a microcontroller programmed in PIC BASIC, while a Visual BASIC program is responsible for producing the control panel display on a PC.

Figure 1 shows the flowchart of the program in the microcontroller. At start-up a brief initialisation sequence runs, which resets the set values to zero and configures some of the microcontroller's pins. The next step, the measurement of the actual voltage and current values, already forms part of the main program loop. All the remaining parts of the program follow sequentially in this loop. One branch that can occur is when the push-buttons are being read. The procedure for reading the buttons is indeed as cumbersome as it (unavoidably) appears from the flowchart. The idea is to read each button in turn and, when one is found that is pressed, the microcontroller stores the corresponding key code. Finally, under `button pressed?' the microcontroller checks whether any button was in fact pressed. If so, a branch is taken to code which increases or decreases the appropriate set value, as long as the value remains within the permitted range. The new set point is then displayed. A half-second delay follows, before the push-buttons are scanned again. This provides an auto-repeat function. If no button is being pressed, the program branches back to the top of the main loop to measure the voltage and current again.

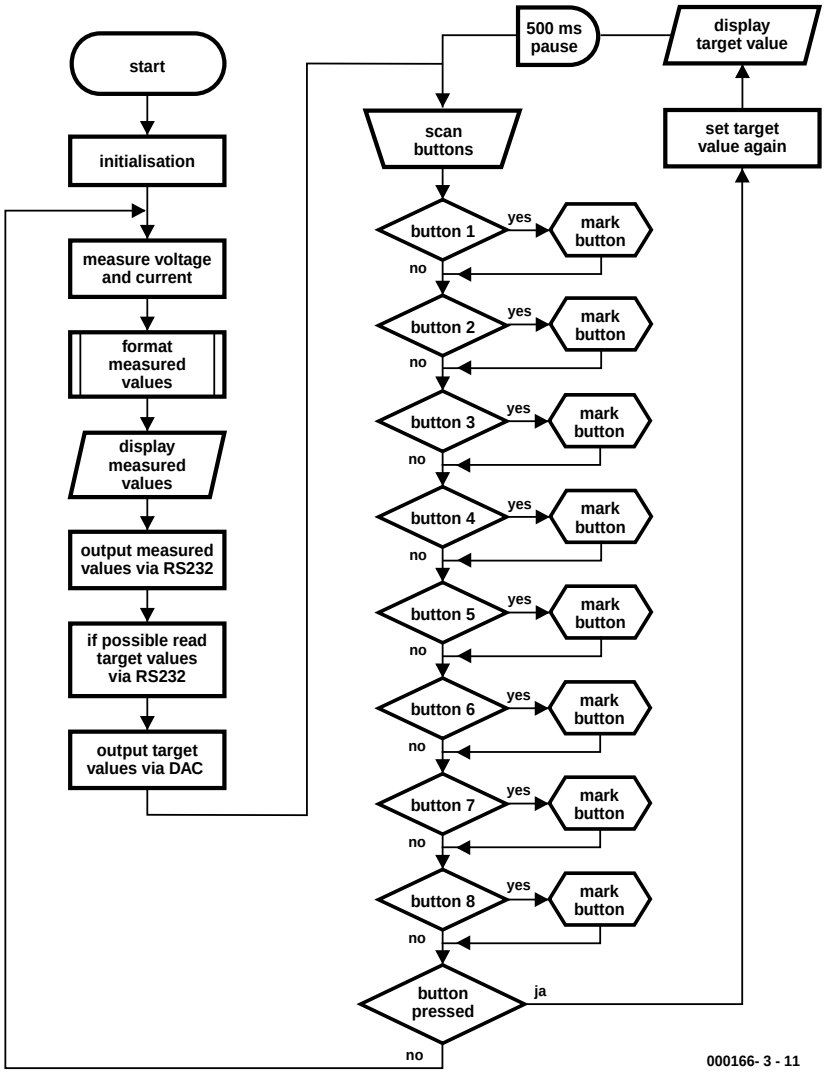


Figure 1. Flowchart for the microcontroller software.

BASIC Program

The source code listing for the microcontroller appears in **Figure 2**. The microcontroller program, written in PIC BASIC 1.3, can be downloaded from www.pic-basic.de.

PIC BASIC allows microcontroller programs to be written quickly and easily. It also

```
'D-PSU 25V, 2.5A or 20V, 1A

'attention: modifications to the program require that register
      numbers
'in the assembler subroutines are checked for changes!!!

'_____________________________
'declaring the variables
VarB Lh1, Lh2, Lh3, Lh5, Lh6, Lh7, Uvalue, Ivalue, y
VarB Buttonnumber, Accu, Callcounter, Bitpattern
VarW Meas_Voltage, Meas_Current

'_____________________________
'Main program
Init:
CV Uvalue, Ivalue  'set to 0 on each start'
Low A3           'ADC output at 0
High B2          'CTS: not ready to receive

Start:
'Measure voltage and current
'Using value 5??? allows ADC scale factor to be adjusted
'                              + - 20 equals approx. 1 digit
Low A4   'Mux to U
ADW A2, 5380, 0, Meas_Voltage  'Voltage measurement
Meas_Voltage = Meas_Voltage Shr 1   'equals / 2
High A4   'Mux to I
ADW A2, 5380, 0, Meas_Current  'Current measurement
Meas_Current = Meas_Current Shr 1   ' equals / 2   'line for
      2.5A
'Meas_Current = Meas_Current Shr 2   ' equals / 4   'line for
      1A

'Format measured values
Call Format

'Display measured values on LCD
LCD B5, " ", Lh1, Lh2, ",", Lh3, "V    ", Lh5, ",", Lh6, Lh7,
      "A "

'Send measured values over RS232
SerOut B3, 9600, "D", #Meas_Voltage, #Meas_Current, 13

'Allows a new target value to be received over RS232
Call RS232E

'Send target values over DAC
PWM A1, Uvalue, 64  'Set voltage (200 = 20V)
PWM A0, Ivalue, 64  'Set current (200 = 2A or 200 = 1A)


'Scan buttons

Entry:

Accu = %00010000  'Bit 4 High (reset by pressed button)
CV Callcounter, Buttonnumber
Call ButtonScan
Call ButtonScan
Call ButtonScan
Call ButtonScan
Call ButtonScan
Call ButtonScan
Call ButtonScan
Call ButtonScan
Branch Buttonnumber, Start, Button1, Button2, Button3, Button4,
      Button5, Button6, Button7, Button8


Button1:
If Uvalue > 240 Then Skip   'Line for 2.5A
'If Uvalue > 190 Then Skip   'Line for 1A
Uvalue = Uvalue + 10
Goto Display_Uvalue

Button2:
If Ivalue > 240 Then Skip   'Line for 2,5A
'If Ivalue > 190 Then Skip   'Line for 1A
Ivalue = Ivalue + 10
Goto Display_Ivalue

Button3:
If Uvalue < 10 Then Skip
Uvalue = Uvalue - 10
Goto Display_Uvalue

Button4:
If Ivalue < 10 Then Skip
Ivalue = Ivalue - 10
Goto Display_Ivalue

Button5:
If Uvalue > 249 Then Skip   'Line for 2,5A
'If Uvalue > 199 Then Skip   'Line for 1A
Inc Uvalue
Goto Display_Uvalue

Button6:
If Ivalue > 249 Then Skip   'Line for 2.5A
'If Ivalue > 198 Then Skip 2  'Line for 1A
Inc Ivalue
'Inc Ivalue  'Line for 1A (omit for 2.5A version)
Goto Display_Ivalue

Button7:
If Uvalue < 1 Then Skip
Dec Uvalue
Goto Display_Uvalue

Button8:
If Ivalue < 1 Then Skip   'Line for 2.5A
'If Ivalue < 2 Then Skip 2  'Line for 1A
Dec Ivalue
'Dec Ivalue  'Line for 1A (omit for 2.5A version)


Display_Ivalue:
'y = Ivalue Shr 1 'equals / 2  'Line for 1A (omit for 2.5A ver-
      sion)
LCD B5, "     ", #Ivalue, "0mA"  'Line for 2.5A
'LCD B5, "     ", #y, "0mA"  'Line for 1A
Pause 500
Goto Entry


Display_Uvalue:
LCD B5, "     ", #Uvalue, "00mV"
Pause 500
Goto Entry


'_____________________________

'Subroutines

'Depending on value in Callcounter, ButtonScan shifts one of
'eight bitpatterns to the pins of the HC164.
'Only the button at the pin with the 0 on it
'can pull PB4 Low. PB4 then indicates if a button was pressed
      or not,
'while Callcounter reveals the button identity


Sub ButtonScan
   LookUp Callcounter, %11101111, %11011111, %10111111,
      %01111111, %11111011, %11110111, %11111110, %11111101,
      Bitpattern
   EXPo B5, Bitpattern, 0  'only Button 0 of bit pattern can
      pull B4 Low
   Inc Callcounter
   PBI %00010000 = Accu        'read only bit 4 of Port B
   If Accu <> 0 then Skip        'skip when no button pressed
   Buttonnumber = Callcounter       'mark Button number
EndSub


'The Basic subroutine Read is called from
'assembler subroutine RS232E

Sub Read
   SerIn B0, 9600, #Uvalue, #Ivalue
   Uvalue = Uvalue Min 250           'limit to 25 Volt  'Line
```

Figure 2. Listing in PIC BASIC.

```
      for 2.5A
   'Uvalue = Uvalue Min 200          'limit to 20 Volt  'Line
      for 1A
   Ivalue = Ivalue Min 250          'limit to 2.5 Ampere
      'Line for 2,5A
   'Ivalue = Ivalue Min 200          'limit to 1 Ampere  'Line
      for 1A
   Y = 1          'Leave loop immediately
Endsub


'Assembler sub-routine Format employs the already available
'Resources for PB. It load the number registers Lh1-Lh8 with
      the
'ASCII values for Numbers 0-9 according to the values in the
'variables Meas_Voltage and Meas_Current.

'The auxiliary subroutine called Packer saves 8 bytes of pro-
      gram memory
'Packer calls machine code program SOSS°, which is contained in
      the
'PB compiler output, when the commands SerOut - #WordVar
'or LCD - #WordVar" was employed.
'It returns the decimal number equivalent of a wörd variable.
'It divides te value contained in HWERT2/R21 by the value from
      the
'jump table SOTT° (also contained in compiler output).
'The value(!) in the FSR has to be the ADD value
'of the jump table (Pos. 5 = 0, 4 = 2, 3 = 4, 2 = 6, 1 = Rest
      in R21).
'LWERT1 contains the ASCII code (characters 0-9) as the result.

Ass Format
      ;format voltage
   MOVF 24,W
   MOVWF HWERT2
   MOVF 23,W
   MOVWF 21
   MOVLW 2
   Call Packer
   MOVWF 27
   MOVLW 4
   Call Packer
   MOVWF 28
   MOVLW 6
   Call Packer
   MOVWF 29
      ;format current
   MOVF 26,W
   MOVWF HWERT2
   MOVF 25,W
   MOVWF 21
   MOVLW 2
   Call Packer
   MOVWF 30
   MOVLW 4
   Call Packer
   MOVWF 31
   MOVLW 6
   Call Packer
   MOVWF 32
   Return

Packer:      ;no repeating of lines; saves 8 bytes of program
      memory
   MOVWF FSR
   CALL SOSS°
   MOVF LWERT1,W
EndAss


'RS232E controls data reception at the interface. Each time it
      it called, the CTS line is pulled High for 1.5ms.
'If a character arrives via RxD within this period, the D-PSU
      goes into Receive mode i.e.
'subroutine Read is called. Next, 2 values with terminating CRs
'have to arrive at the interface before the controller is
      allowed
'to leave the subroutine

Ass RS232E
   CLRF 35        ;Clrf Y (= R35)
 RS232:
   BCF PB,2       ; CTS: ready to receive
   BTFSS PB,0     ; RxT pin test
   Call Read
   DECFSZ 35,F    ;
   GOTO RS232     ; repaet loop 256 times
   BSF PB,2       ; CTS: not ready to receive
EndAss
```

makes compiling the program and programming it into a chip easy. Further information on PIC BASIC, as well as the most up-to-date version of the program, can be found on the Internet at www.pic-basic.de. At the time of writing this article, the information on PIC Basic is only available in German. We hope that Mr. Pagel will eventually produce English translations.

Figure 3 shows in-system programming of the 1 A power supply using the PIC BASIC programmer.

First all the variables used in the program are declared. There are 13 byte-wide variables and two word-wide variables, occupying a total of 17 bytes of the microcontroller's RAM (and a further twelve bytes are reserved by PIC BASIC as a scratch area). Then follows the first part of the program: this is the part indicated in the flowchart by `initialisation'. The label **Start** marks the entry point for the main loop. The program is so thoroughly commented that a detailed description is not necessary here. A few remarks are, however, in order:

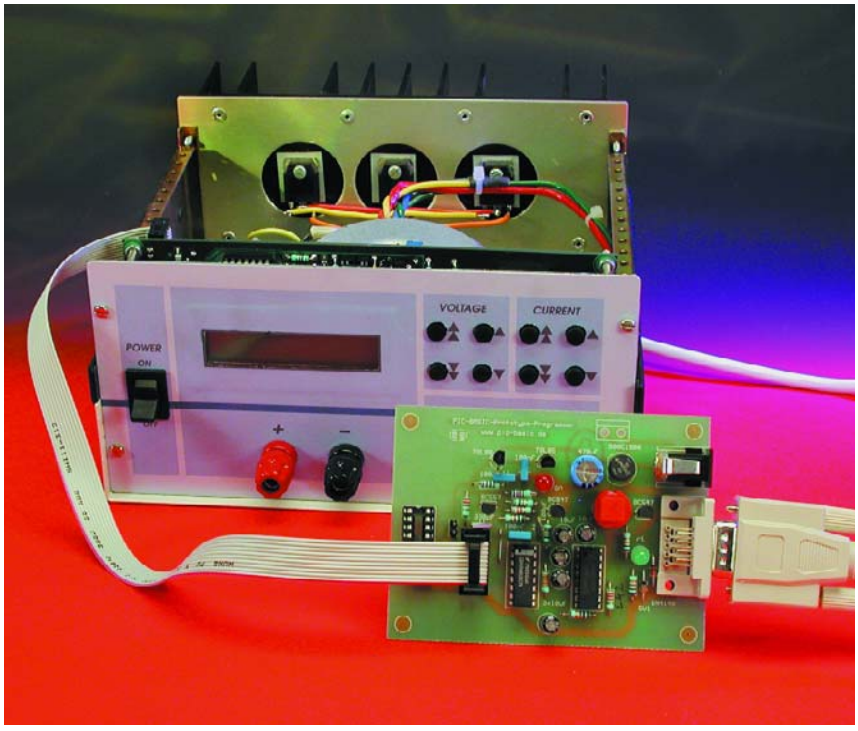


Figure 3. In-system programming of the power supply microcontroller.

A/D converter

The command

```
ADW A2, 5380, 0, Meas_Voltage
```

carries out an analogue-to-digital conversion and writes the measured value into the variable `Meas_Voltage` (measured voltage). The scale factor can be adjusted by changing the value `5380` above. The circuit is, however, designed so that this will not normally be necessary.

Assembly code subroutines

Under `Format measured values` the assembly code subroutine `Format` is called. This subroutine is 30 bytes long and uses some (PIC BASIC) subroutines, provided for the use of other BASIC commands, to format the measured values for display. Using this trick a large amount of precious program memory can be saved.

The eight-byte subroutine `RS232E` is also written in assembler. It sets the CTS signal high and waits for a period to check if data are being sent from the PC. If so, the assembler subroutine calls the BASIC subroutine `Read` which is responsible for actually reading the data in.

All the remaining parts of the program are written entirely in BASIC. The compiled code size for the complete program is either 1009 bytes or 1021 bytes (for the 2.5 A and 1 A versions respectively), and so just fits into the program memory of the PIC16F84. The program lines that need to be changed between the two versions of the power supply are marked in the BASIC program listing.

If it is desired to enable the watchdog timer in the microcontroller, the configuration word in the assembly output must be changed as follows:

```
CONFIG B'11111111110101'
```

Also, a `CLRWDT` instruction must be inserted at one point in the code, in the main loop and in the push-button scanning loop. The latter loop runs in just over 500 ms and the main loop runs in about 780 ms. With the values set in the Option register on power-up, a watchdog reset will occur after 2.3 s. This is enough time for both loops (indeed, even half the time would be adequate). The simplest place to put the `CLRWDT` instruction is immediately after the label `Entry`. The following code is then shifted down by one location, which makes no difference in this case.

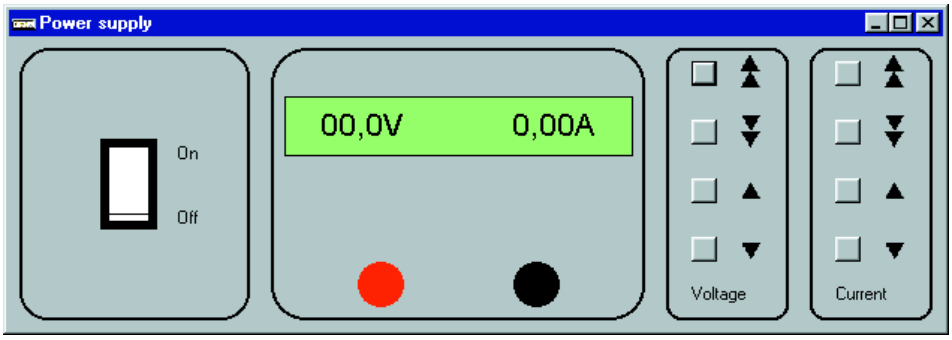


Figure 4. Power supply display on the PC's monitor.

The interface protocol

The data packet that the power supply sends out over its interface is structured as follows:

```
Duuuuuiiiii↵
```

First a `D` is sent, followed by five-digit values for voltage and current, and finally a carriage return character. The least-significant digit of the voltage value represents 10 mV, and that of the current value 1 mA. The two values are thus given to a resolution ten times higher than that shown on the LCD panel. The leading digit of the voltage and current values is always zero.

When set values are sent to the power supply, both voltage and current settings must be sent, one immediately after the other. In both cases up to 3 digits can be sent.

```
uuu↵iii↵
```

After each value a non-digit character (for example a carriage return character) must be sent. For either version of the power supply a voltage value of 20 V must be sent as `200`. For a current of 1 A, a value of `100` should be sent in the case of the larger model, `200` in the case of the smaller model.

Visual BASIC program

The control program, written especially for this project, runs under Windows 95 or 98. Its control interface resembles the front panel of the power supply (**Figure 4**). Just as in reality, the voltage and current set values can be adjusted using the eight buttons. When the mouse button is released the values are sent to the power supply. The set and actual values are shown on a simulated LCD panel. If you click in a region of the main program window other than on the buttons, a settings window appears. Here the version of the power supply (2.5 A or 1 A) and the COM port (1 to 4) used for data communication can be configured. If a mouse is connected to COM1 it can happen that the program does not work correctly with COM3, but this is a common problem with PCs.

In the right-hand part of the settings window the names of a log file and of a control file can be specified. A click of the mouse on the adjacent `Start' button and a file is either read or written: the file contains the displayed readings along with a time stamp indicating when they changed (in the case of a log file), or when they are to be changed (in the case of a ready-prepared control file). In the simplest case the control file can be created from a log file by modifying the time stamps.

The following example one-line data record shows the format used for the log and control files:

```
#2000-08-20
14:35:53#,"04.9V","0.97A"
```

Between the two hash symbols ('#') we have the date (in international format) and the time when the indicated change occurred or is to occur. Within the record the time stamp and the two electrical values are separated from one another by commas. This allows for the processing and, for example, the graphical presentation of the contents of the file using a spreadsheet program.

(000166-3)

Voice Extreme Toolkit

When it comes to speech recognition products the California based company Sensory (*www.sensoryinc.com*) have been at the forefront for some time now. Their latest offering is the Voice Extreme™ speech recognition kit.

This kit contains a complete development environment allowing applications to be programmed using VE-C, a high-level language very similar to ANSI-C. As you would expect this language gives a great boost to programmer productivity compared with Assembler based software environments and allows the development of speech-controlled applications in a Windows environment. The resultant code is compiled and downloaded to the Voice Extreme module flash memory. The module can then be run in the development board or removed and plugged into the target system. The kit is distributed in the UK by:

Milford Instruments
Leeds. LS25 5AQ
Tel:01977 683665
www.milinst.com

The development kit contains a versatile hardware development card containing the speech module together with a software Integrated Development Environment (IDE) including an editor with source code debug and the Quick Synthesis™ program which allows any sound files with the .wav extension to be used in any of the Sensory RSC applications.

One board – many possibilities

The development kit hardware includes all the electronics, a microphone, speaker and sockets for connection to external equipment. An RS232 cable and mains adapter is also included. On the card there are five pushbuttons, one is for system reset another for program download/program starting while the remaining three, along with three status LEDs are free to be used by your application software.

A prototyping area is also available on the board to mount additional application specific hardware. All the I/O signals of the module are available along with the supply voltage rails. A connector is also included if you prefer to build the additional hardware on an external card.

The heart of the development kit is the Voice Extreme module. It includes

the RSC-300/364 microcontroller, crystal oscillator, firmware ROM and a 2 Mbyte flash memory to store your own application software. This module plugs into the development board (and eventually into your target application) via a 34 pin header. For production purposes the Voice Extreme module is available in a QFP package or in die form. In-circuit reprogramming is also supported.

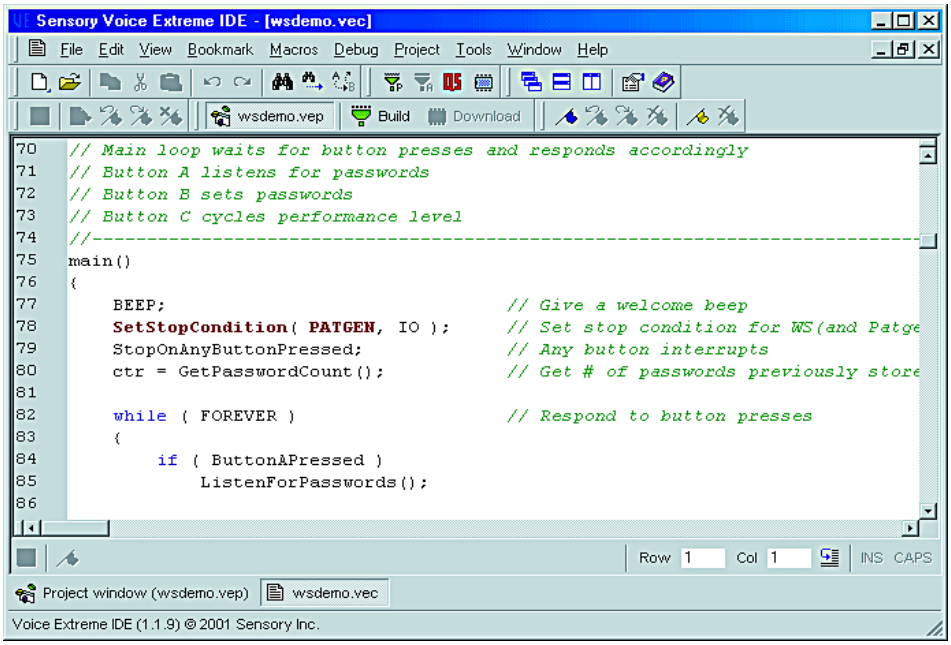
Software

The bundled software includes an Integrated Development Environment (IDE). This program suite contains all the tools necessary to write application specific software and includes a C editor, compiler, linker together with help functions and downloader. The editor supports C-Highlighting which improves program readability.

The IDE is supplied on a CD ROM together with the Quick Synthesis™ program that converts and compresses *.wav* files into a compatible file format used by the hardware. Also included on the CD are examples and extensive documentation.

In order to write speech recognition programs easily, it is of course necessary to make use of the library routines provided. Thanks to the numerous examples included on the CD together with the on-line help facility we found it a simple process to understand and make use of these library functions.

In the software documentation a distinction is made between 'speaker independent' and 'speaker dependent' speech recognition technology. In the first case the software will simply recognise a signal but in the second case it will recognise that signal only from a particular speaker. The difference between the two is a question of the accuracy with which the software evaluates the signal. The higher the accuracy then the more speaker specific the result will be. The software allows the recognition accuracy to be adjusted in steps. As well as the speech recognition routines are also software sound routines that can be used to generate synthetic music and allow you to play with samples and sound fragments.



In practice

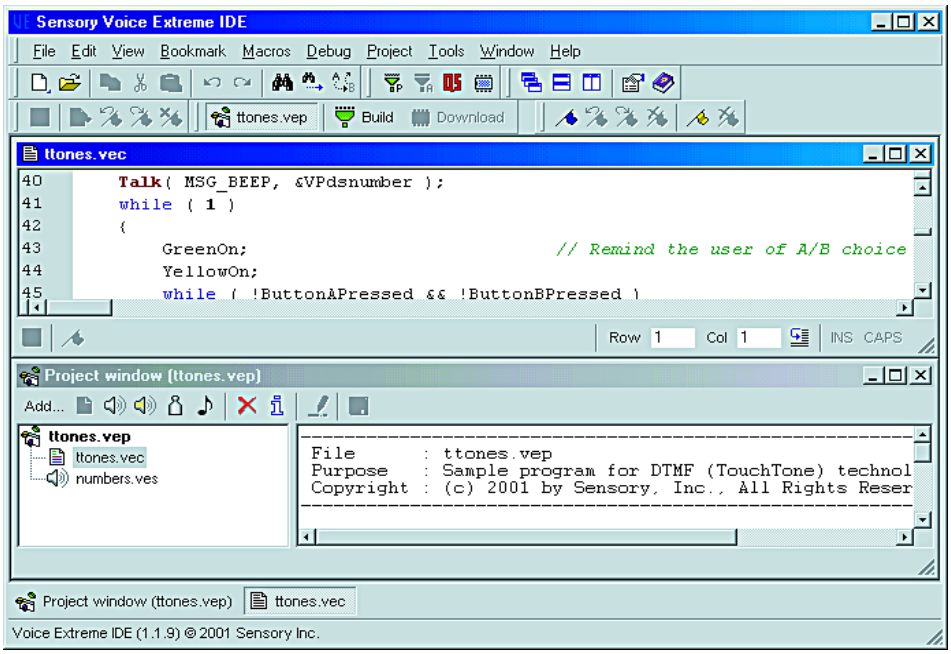
All of the included example programs were tested in the *Elektor Electronics* laboratory. The speech recognition functioned outstandingly well despite the low quality microphone fitted to the card. The manufacturer suggests substituting a high quality microphone to further increase software recognition performance.

The examples demonstrate just some of the possibilities of this powerful tool. The accompanying source code gives you a good insight as to how the card can be used in many different situations and is an ideal starting point for you to begin writing your own applications.

In addition to the speech recognition examples there are also interesting music generation demos and compression of tone samples that work with the library supplied.

The kit is ideal for anyone interested in the field of speech recognition and especially if you already have an application in mind. All you need is a PC running Windows (95+) with 16 Mbytes RAM, 15 Mbytes free hard disk space, an unused RS232 port together with a little C programming experience and of course the purchase price of the kit £109+VAT.

(010086-1)

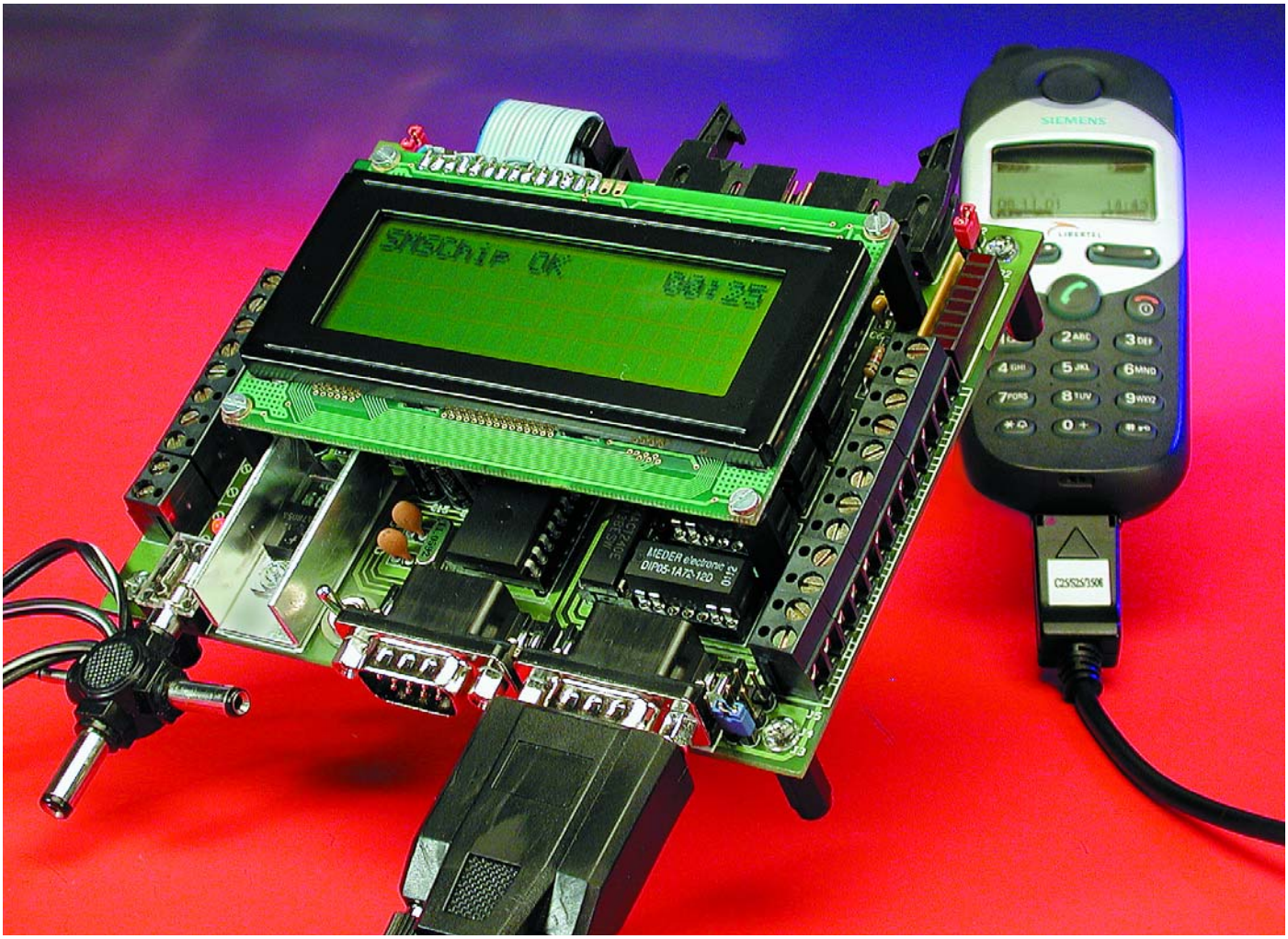


Remote Process Control using a Mobile Phone

for '35' series Siemens mobiles

Design by Prof. B. vom Berg and P. Groppe (Georg Agricola Technical University)

The Short Message System (SMS) of a standard mobile phone can be used for much more than just exchanging cryptic messages. This application finds the humble mobile working in a remote site monitoring and controlling external equipment.



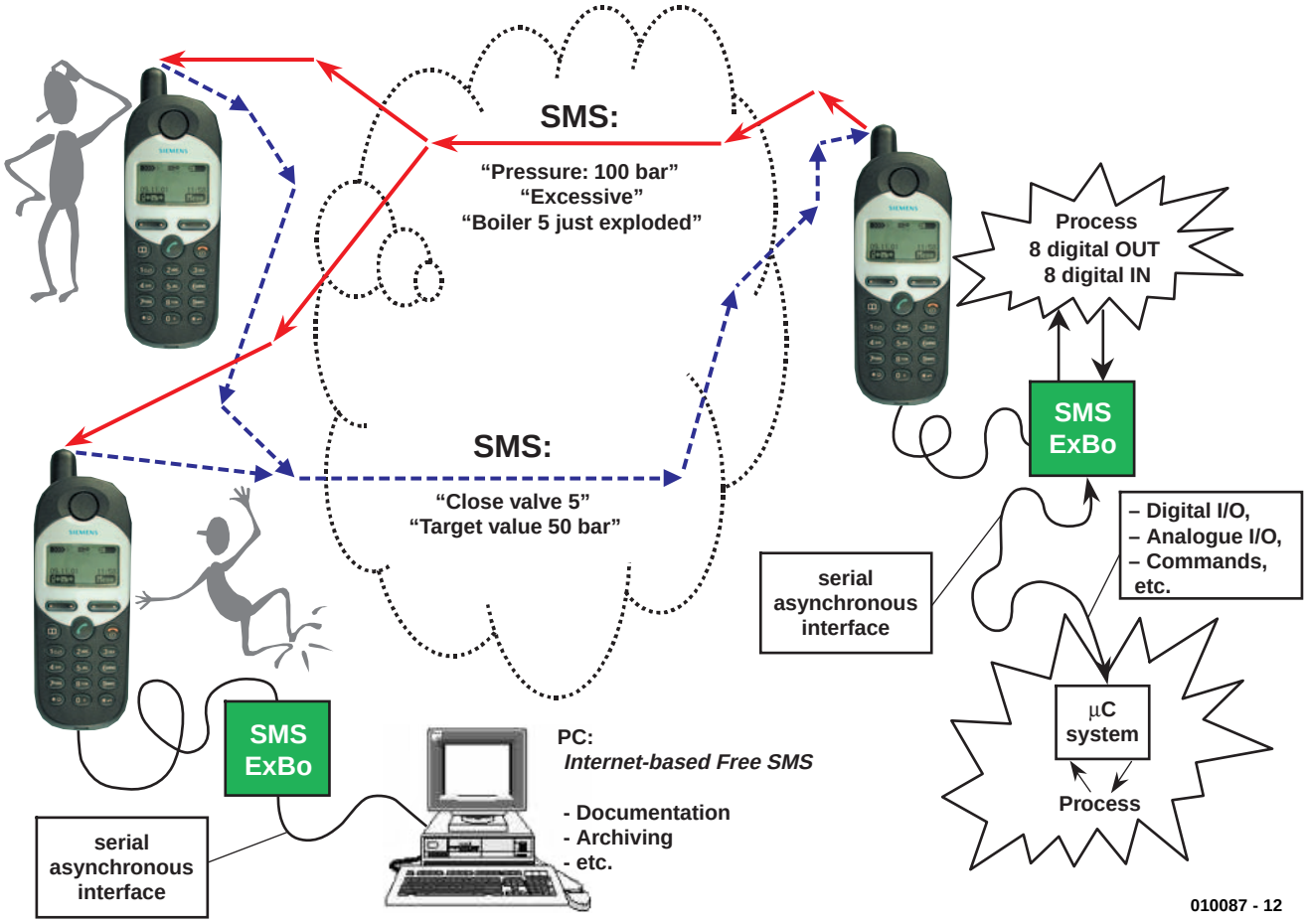


Figure 1. Remote process control using a mobile and SMS.

This design is the result of collaboration between a technical university (TFH Georg Agricola) and the company of Engelmann & Schrader. Together they have produced a flexible, professional remote process controller in the form of an experimentation card (The TFH SMS ExBo). This card connects between the process to be controlled and a mobile phone and allows the process to be controlled and monitored remotely using SMS messages.

Controlling the process using SMS

The SMS interface board offers two levels of control sophistication. At the basic level it will interpret SMS messages, check the incoming password and directly control output relays or indicators. In the other direction the SMS chip on the interface card inserts information into an SMS message which is then sent over the serial interface connector to the mobile where it will be sent to any SMS-capable phone worldwide. Control information in the SMS message will for example be **set 10** this will have the effect of setting output 10 on the interface card and switching on any LED, motor or relay connected to this output. Similarly in the other direction the SMS chip reads inputs to the interface card from the controlled process and generates an appropriate SMS message that will then be passed over the serial interface to the remote mobile where it will be sent out to any SMS-capable mobile worldwide. The message may convey information such as 'Over-pressure detected in tank 3' or 'Intruder alert door 5' or even 'Holiday cottage central heating boiler on' the possibilities are endless.

The TFH SMS ExBo ('Experimental Board') interface board is equipped with a second asynchronous serial interface port and this allows connection of an external computer or process controller at the remote site. This facility gives a higher level of control and monitoring allowing measured values, variables and process status information to be exchanged. With this setup the interface board will initialise the mobile as before but it will pass SMS messages unaltered through to the external computer. This gives the system a much greater flexibility with SMS messages being decoded and generated in the external computer.

The SMS-Chip

The microcontroller used in this design is the AT89S8252 or AT89LS8252 from Atmel. It is based on the 8051 processor and has an 8 kByte Flash program memory together with a 2 kByte Flash data memory. It is programmed to act as an SMS chip and essentially performs three basic tasks:

– Re-program the mobile and handle communications with the mobile over the serial interface.

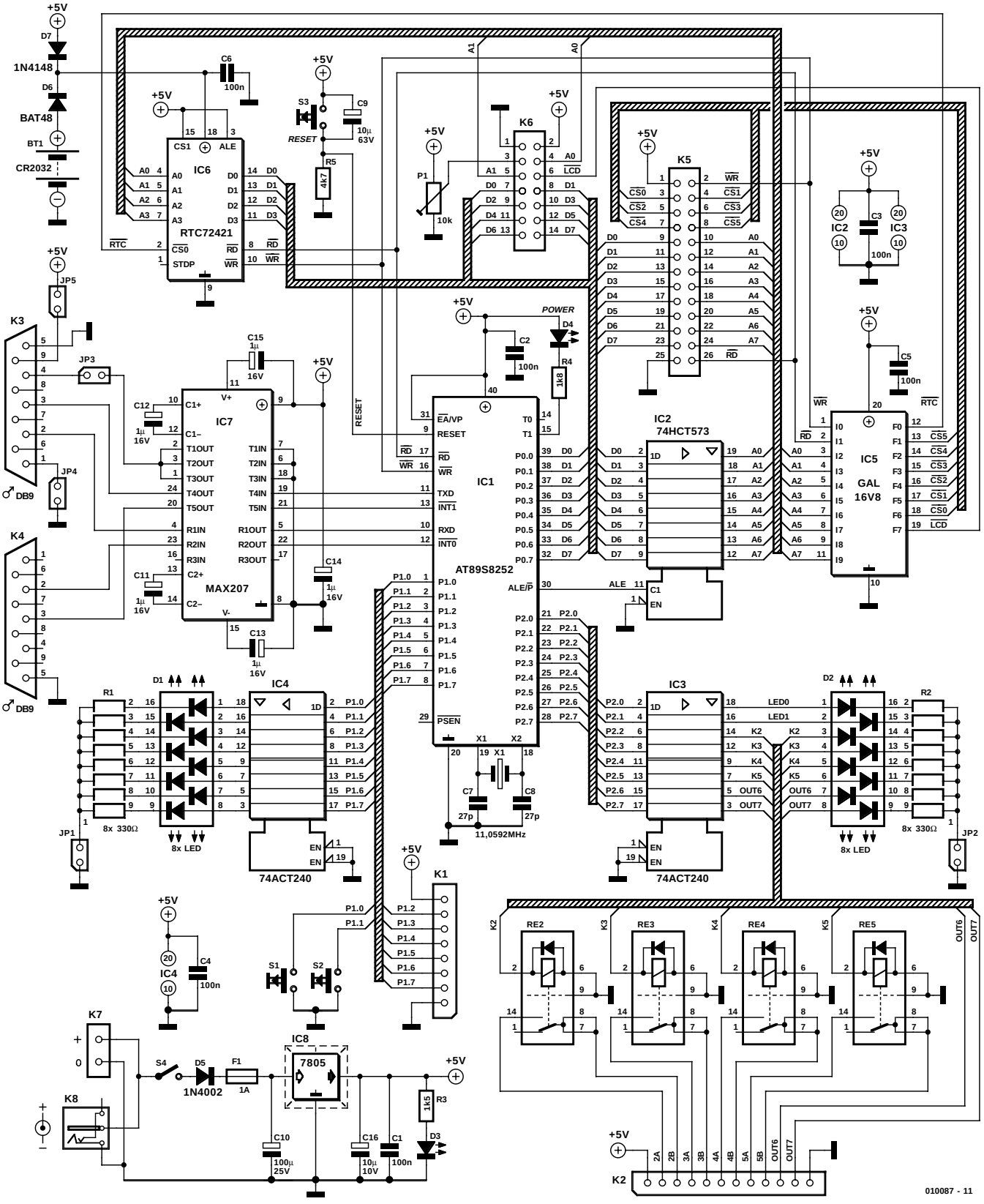


Figure 2. The circuit diagram.

Table 1. Input/output port specifications		
Port	**Pin No.**	**Function**
P1.0	0	INPUT : connection for switch S1
P1.1	1	INPUT : connection for switch S2
P1.2	2	INPUT : TTL level, unprotected
P1.3	3	INPUT : TTL level, unprotected
P1.4	4	INPUT : TTL level, unprotected
P1.5	5	INPUT : TTL level, unprotected
P1.6	6	INPUT : TTL level, unprotected
P1.7	7	INPUT : TTL level, unprotected
P2.0	8	OUTPUT: connection LED0 of LED array D2
P2.1	9	OUTPUT: connection LED1 of LED array D2
P2.2	10	OUTPUT: Relay Re2; max. 200 VDC, max. 1 A, max. 15 W
P2.3	11	OUTPUT: Relay Re3; max. 200 VDC, max. 1 A, max. 15 W
P2.4	12	OUTPUT: Relay Re4; max. 200 VDC, max. 1 A, max. 15 W
P2.5	13	OUTPUT: Relay Re5; max. 200 VDC, max. 1 A, max. 15 W
P2.6	14	OUTPUT: TTL level from 74AC/HCT240 driver chip
P2.7	15	OUTPUT: TTL level from 74AC/HCT240 driver chip

– Receive and decode SMS messages: monitor and control inputs and outputs of the board. Generate SMS messages.

– Pass messages to and from the mobile and external computer system.

Two 8-bit ports (port 1 and port 2) are available on this chip giving 16 programmable digital input/output lines. The chip also has a built-in Universal Asynchronous Receiver/Transmitter (UART) that handles serial communication and in this application it will pass serial data including SMS messages to and from the mobile. A second serial interface is implemented in software to communicate with the external PC or process controller at 9600 Baud.

The interface board also contains a real-time clock chip with battery back-up and a connector to fit a dot matrix LCD to display SMS messages.

Serial commands are sent to the mobile using standard Hayes modem (AT commands) sequences. For more on Hayes modem control see accompanying text box.

SMS Chip Hardware

The SMS chip together with some additional peripherals will produce a very basic SMS message handling design but if you look at the circuit diagram in **Figure 2** you will see that the TFH SMS ExBo interface card has been designed to be the basis of a very flexible platform for SMS message development allowing many connection possibilities. The (DIL) SMS chip IC1 is clocked by a standard crystal oscillator configuration (X1, C7 and C8) and a reset circuit is provided by R5, S3 and C9. A low power LED (D4) is driven by output pin P3.5 indicates that the GSM phone is ready. **Table 1** shows the I/O pin assignments.

Digital inputs (P1.2 - P1.7)
The connector block K1 allows connection of up to six input signals. All of these inputs connect directly to the microcontroller port P1. The state of these signals is displayed on LED array D1 via buffer IC4. If your application does not need this feature or you want to keep current consumption as low as possible then jumper J1 need not be fitted and these LEDs will remain off.

The two remaining inputs on P1.0 and P1.1 are used for switches S1 and S2.

Digital outputs (P2.0 - P2.7)
Outputs P2.0 to P2.7 are buffered by IC3. Four of the outputs are used to drive relays RE2 to RE5 and these provide four switched outputs at connector K2. Two TTL level outputs OUT6 and OUT7 are also available at this connector. The state of these outputs is displayed by LED array D2 and again if this feature is not required jumper J2 need not be fitted.

The Serial Interfaces
There are two serial interfaces supplied on the interface. K3 connects to the mobile phone while K4 connects to the serial port of an external computer.

IC7 (MAX207) converts the voltage level of the signals on both interfaces (V24) to TTL levels used on the interface card. A voltage of approximately 10 V is also produced by the outputs of T1OUT, T2OUT and T3OUT for the data cable. Removing jumper J3 will disconnect this voltage from the interface (see **Table 2**). Current to charge the battery in the mobile is supplied from pin 9 of connector K3 when jumper J5 is fitted. Sub-D connector K4 is used to connect to an external computer or process controller where SMS messages can be sent and received.

SMS Chip peripheral circuits
The SMS chip, like all other 8051 processor derivatives requires a little bit of external peripheral circuitry. Firstly the address and data bus need to be demultiplexed at port P0 and this is performed by an octal D type flip flop (IC2) using the ALE signal. These address lines are now decoded by GAL IC5 to generate chip select signals for the rest of the components on the interface card. The address lines together with $\overline{RD}$/$\overline{WR}$, chip select ($\overline{CS}$) along with the supply voltage are available on pin-strip K5.

The circuit also includes a real-time-clock and a connector for an LCD both of which are controlled by the SMS chip.

The RTC (Real Time Clock)
An accurate time reference is essential for some applications so a Real-Time-Clock (IC6) is included in the circuit. This chip maintains the correct time of day for the whole TFH-SMS-ExBo-System. A keep-alive battery (BT1) is charged via D7 and ensures that the RTC chip does not lose time if the main power from connector K8 fails.

The alphanumeric LCD
An LCD can be attached to connector K6. The picture at the beginning of this article shows

Chatting to the mobile

The SMS chip used in this interface card communicates with the mobile phone using Hayes compatible command sequences over its serial interface connection port. Back in the 70's modems were curious computer peripherals that just hung around whistling, waiting for the Internet to be invented. There were many modems on the market each model offering similar performance but with incompatible control commands. The pioneering US company Hayes came up with a set of commands that could be used to control the modem and it wasn't long before a modem was not worth considering unless it was 'Hayes compatible'. Even today the command sequences are still implemented in all modems and mobile phones.

The commands begin with the ASCII characters AT and the actual standardised commands start with the character string AT+C, all ending with the ASCII code for Carriage Return. These commands are also known as the 'AT' or 'AT+C' commands. These commands have more recently been adopted by the mobile phone industry and are defined in sections GSM07.07 and GSM07.05 of the GSM mobile phone specification for the control of phones over a serial interface. The interface can use V24 signal levels via a cable or IrDa infrared link. Altogether there are 55 AT commands listed which all of today's GSM phones must understand. These allow for example access to the telephone book in the mobile, managing SMS messages, adjusting ring tones and speaker volume etc.

Connecting a data link cable (Data cable) or using an infrared link between the mobile and the serial port of a computer means that it is now possible to control the mobile from a computer keyboard rather than the phone keypad. This is much easier on the fingers when sending text messages.

In addition to the standard commands each manufacturer has defined extra commands that will only be understood by their own mobiles. With the Siemens '35' series (S35i, C35i, M35i) there are 25 additional commands all prefixed with AT ^ S. This diversity creates problems for anyone considering building a universal SMS interface. It is necessary to study the phone specification closely to guarantee success. In this design all of these commands are pre-programmed into the SMS chip so that it can be directly connected to the Siemens '35' series of mobiles. Hardly any programming is necessary to develop a remote control application and even a second interface is provided at the remote site so that an external PC can be connected to provide more complex control possibilities. It is of course only necessary to specify this type of phone at the remote site, communication will occur over the air with any SMS-capable phone anywhere in the world!

a four line by 20 character display but most alphanumeric displays can be substituted provided that are compatible with the Hitachi HD44780 controller. Preset P1 allows the display contrast to be adjusted.

The GAL chip

Chip select signals on the circuit are generated by IC5, a 16V8 Gate Array Logic (GAL) chip. The GAL chip simply reads the addresses at its input and generates chip select signals for the peripheral chips. The six chip selects $\overline{CS0}$ to $\overline{CS5}$ are also available on the pin strip K5.

The power supply

IC8 is a fixed voltage regulator that supplies +5 V for the complete interface board and charging current for the mobile. The mains adapter unit should supply a voltage in the range of 9 to 12 V with a current of 800 mA, (including the mobiles charging current) connected to K7 or K8. D5 protects the circuit from accidental reversal of the power input leads and LED D3 is the power-on indicator. S4 is the on/off switch.

Jumpers

The finished PCB has several jumper options and their purpose is outlined in **Table 2**.

(010087-1)

In the second part of this article we will look closer at the connection between the mobile and this interface board, the basic configuration and command sequences of the SMS chip. We also look at the layout for the circuit.

Table 2. Jumper assignment

J1	Fit this jumper to activate LED-Array D1 (Displays the input status)	
J2	Fit this jumper to activate LED-Array D2 (Displays the output status)	
J3	Positive supply potential for the Data Link cable:	
	This jumper should be fitted when using an off-the-shelf Data Link cable, otherwise do not fit this jumper.	
J4	Controls the charging current to the battery in the mobile (not used on Siemens S35):	
	Fitted (Low level):	Standard charge with 5 V at 150 mA
	Not fitted (High Z):	Fast charge with 5 V at 400 mA. Only fit this if using a custom made Data Link cable with charging function. (See part 2 of this article).
J5	Positive charging potential for the battery in the mobile (not used on Siemens S35):	
	Fitted:	Allows mobile battery to be charged when using a custom-made Data Link cable (see part 2 of this article).
	Not fitted:	In all other cases.

Microcontroller Basics Course

part 1: the TASM assembler

By B. Kainka

This course is for everyone who always wanted to how microcontrollers work and how to use them, but was always afraid to ask. It is intended to explain the fundamentals, starting from scratch. The Elektor Electronics 89S8252 Flash Microcontroller Board (presented in last month's issue) is used as the hardware platform.

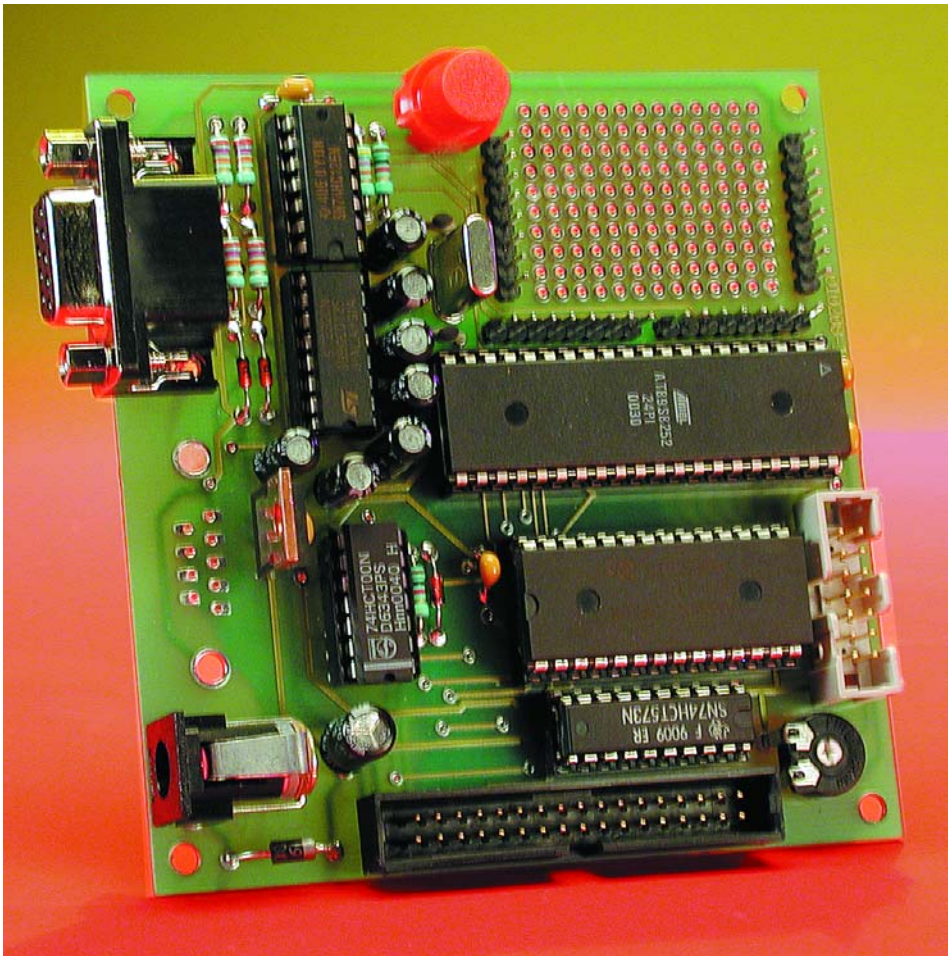


Figure 1. The 89S8252 Flash Board, which is used in this course, is a general-purpose microcontroller system.

Nowadays, we all take working with computers for granted, and not only that, we often work with quite powerful equipment. The heart of a computer is its processor, such as a Pentium III. Relatively speaking, a microcontroller is both much less and much more than the processor of a typical PC. It is less because it processes smaller programs, uses less memory and is usually much slower. However, it is also more because it already has many elements on a single chip that are spread out over the complete motherboard of a PC, namely working memory, timers, interfaces and port connections. What makes microcontrollers attractive is that is in the limiting case, a complex problem can be solved using only a single IC. Using programming alone, anyone can produce a special IC that does exactly what he or she wants — and at a relatively low cost.

A microcontroller is thus some-

thing like a logical circuit with many possible inputs and outputs. What this circuit does is determined by a program. Perhaps you want to build a digital counter, or would you rather have a stopwatch? Do you want to create a special logical gate, or perhaps a universal clock generator? Do you need to decode a complicated digital signal or control a digital circuit? In all of these cases, a microcontroller can help you. There are many examples of problems whose solutions previously required an enormous board full of ICs and now can be solved quite elegantly by a single IC, namely a microcontroller. Consequently, some knowledge of programming is worth having. There are many different approaches that can be taken to achieve this goal.

The hardware basis for this course is the **89S8252 Flash Microcontroller Board** described in last month's issue of *Elektor Electronics* (see **Figure 1**). As already announced, for programming software we will use the following three programming languages: assembler, Basic and C. Our first experiments will be carried out in assembler. Why should we use assembler in particular? Isn't it rather difficult, perhaps too difficult for beginners? The answer is no, since the initial examples will be very small and easy to understand. The advantage or using assembler is that it allows us to work very close to the hardware, so we can see exactly what is happening. High-level languages (such as BASIC), by contrast, hide much of what actually takes place.

In our first experiment, all we want to do is to switch the levels on one of the microcontroller's output ports. After all, operating a switch is the first step in automation. Also, the results can be observed using a voltmeter connected to lead P1.4 via connector K4 (see **Figure 2**).

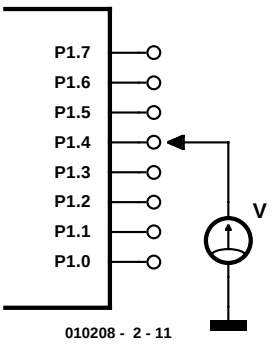


Figure 2. The results of the first experiment can be checked using a voltmeter.

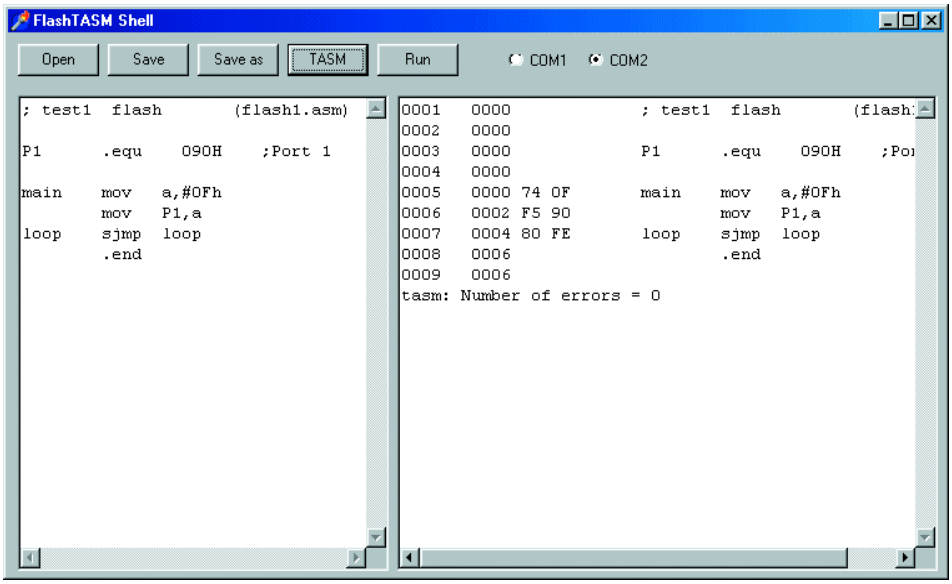


Figure 3. The first sample program in TASMedit.exe.

In order to change the level on this lead, we will use a small assembler program. Put briefly, assembler is a notation used to write instructions for a processor or microcontroller. Every microcontroller has an instruction set, which ultimately consists of numerical values and associated functions. The following series of six numbers represents a small, complete program for an 89S8252 microcontroller — in fact, it is what is called a machine-language program:

```
116,15,245,144,128,254
```

It is generally customary to write computer programs using hexadecimal numbers instead of decimal numbers, since the former are easier to read. In hexadecimal notation, the above program looks like this:

```
74 0F F5 90 80 FE
```

We have to write this sequence of numbers into the microcontroller's program memory. We can use a program called MicroFlash for this purpose. Ultimately, the program numbers end up in the program memory of the microcontroller. The microcontroller reads the numbers from the memory, one after the other, and it then knows what it has to do. In normal language, we can express this as follows:

74: So, I'm supposed to transfer a numerical value to the accumulator (that's my memory) — but which one?
0F: Here it is: OF —— good, I've made a note of it.
F5: OK, now I have to write the value to a register — but can you please tell me which one?
90: I see, the register for Port 1 is located at address 90. There you are.
80: And now I have to make a short jump — but to where?
FE: Two bytes back from the location that would have been the next one. OK, I'm jumping!
80: The same jump again — OK, I guess I'll just have to keep on running around in a circle.

This is how the microcontroller 'thinks' and acts, since clever engineers have trained it to behave this way. Ultimately, a microcontroller is nothing more than a very complex circuit made up of logic gates. This circuit responds to the states of its input lines, which in this case are the data lines connecting the program memory to the central processing unit.

If we wanted to know what goes on inside a microprocessor in detail, we would have a lot of work on our hands. However, it is sufficient for us to know the machine instructions of a processor and be able to use them. This means that the microcontroller itself remains a sort of 'black box', whose inner functions

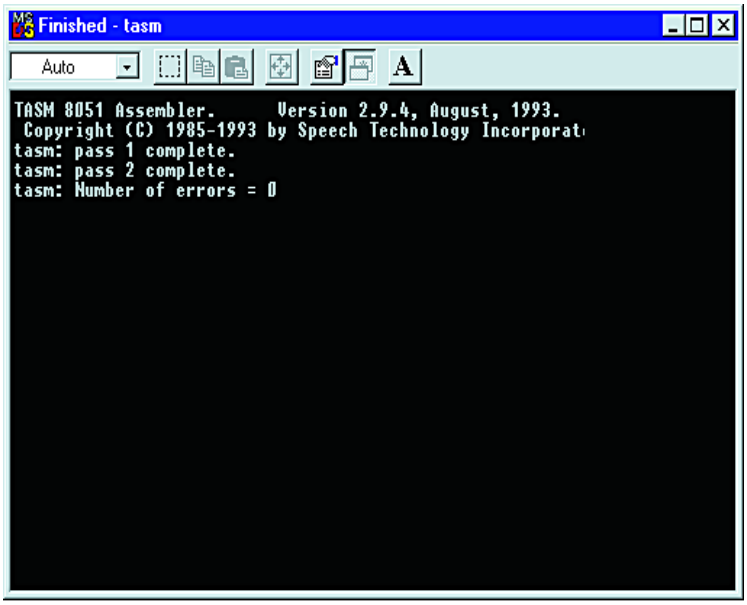


Figure 4. TASM in the DOS window.

we do not fully know but whose behaviour is easy to understand. That's how it is with modern technology — it has become nearly impossible to regard everything at all possible levels of understanding.

So, now we know that a microcontroller has its own language, which actually consists of nothing but numbers. However, there is an obvious problem: the language that a microcontroller can read easily and fluently is not exactly suitable for people. We are not made to work with numbers, but rather with words. Consequently, words (which are easier to remember) have been devised to represent the individual machine-language instructions. The programmer writes these words in a text file, and a special program then translates them into the language of the microcontroller. This program is called an assembler, and the programming language is also called assembler (or assembly language). Assembler is thus a notation that you and I can use to tell a microcontroller what it should do, as in:

```
main    mov   a,#0Fh
        mov   090H,a
loop    sjmp  loop
```

This is already much more readable. Actually, here we only need to know two special words: 'mov' and 'sjmp'. Both of these words are called mnemonics, which means markers used in place of the actual machine-language instructions. The word 'mov' (move) means 'move', 'shift' or 'load'. Following it comes first the location where something is to be loaded and then what is to be loaded. In the first line, the numerical value 0Fh (= 15), which is identified by the '#' symbol, is loaded into the accumulator a. The accumulator is a register or memory with a size of eight bits, so it can hold numerical values between 0 and 255.

In the second line, the value in the accumulator is then copied to address 90h (= 144). At this address there is a register whose leads are routed to the exterior of the IC, namely to the Port 1 pins. The word 'sjmp' (short jump) causes a jump in program execution, in this case to the location 'loop'. The word 'loop' has been chosen completely arbitrarily and simply represents an address, in this case a position in the series of instructions. The assembler treats such words, which are called labels, as addresses and replaces them with the appropriate numerical values. The sjmp instruction can cause a jump of up to 127 bytes backwards or 128 bytes forwards. A single byte is thus sufficient to specify the jump destination. Here the jump is calculated relative to the current position in the program.

The word 'main' at the beginning of the program is also arbitrarily chosen. The only actual assembler keywords here are thus 'mov' and 'sjmp'. We humans can easily remember such words without the aid of an electronic brain.

However, there is still a problem. Is it really necessary for us to remember that a particular register for Port 1 is located at position 90h? After all, it would be nicer if we could also write this as text. This is easily done; we simply have to define a certain bit of text as being equivalent to a numerical value. The assembler will then replace this text with the corresponding numerical value at every location where the text is found. To make such a definition, we use the assembler directive '.equ' (equate). An assembler directive always begins with a full stop, which informs the assembler that it is not an assembly-language instruction. In the following example, the word 'P1' is assigned the value 90h. In the actual program, the value 90h is thus replaced by 'P1':

```
; flash1.asm  port output

P1    .equ  090H   ;Port 1

main  mov   a,#0Fh
      mov   P1,a
loop  sjmp  loop
      .end
```

This listing also shows us something else: the jump labels and newly defined words are all located at the beginning of the line, and all assembler instructions are located somewhat indented. Furthermore, there is also a comment, which plays absolutely no part in the translation. A comment starts with a semicolon (;).

The specific notation varies somewhat from one assembler to the next. Here we are using the shareware assembler **TASM** (Table-Driven Assembler, a program written by Thomas N. Anderson). TASM is very simple and can translate programs for many different types of microcontrollers, as long as it has the appropriate instruction table.

At the end of the program there is a loop, in which a jump to the destination 'loop' takes place, always and forever. In other programming languages, such a situation would be called a fatal endless loop, which is practically equivalent to a crash. In such a loop, the processor is in a state that it cannot exit under its own power. In general, it should always be clear what should be

done once the current task has been completed. However, in this case this loop is very important. There is only one task, namely changing the state of the port. If we were to leave the processor to its own devices, it would execute commands that just happen to be in the program memory and perhaps belong to a completely different program. Consequently, a limit must be set by means of an endless loop: this far and no further! In fact, the processor is trapped in this loop, with the only means of escape being a reset. After that, the same program could be started again, or we could load a new program and then run it. Loading a program also takes place in the reset state; the program memory is thus not filled by the processor itself, but by special functional blocks in the microcontroller that program the flash ROM. Each time the board is switched on, a short reset is automatically executed. Following this, the microcontroller finds the most recently loaded program and runs it.

Using the TASM assembler

Now it's time to get down to business! What we want to do is to write this first program, translate it and send it to the microcontroller. For this, we need some software. We will use the well-known shareware assembler TASM, which is located on the working diskette for the course in the form of a zip file; it can also be downloaded from the *Elektor Electronics* website. The file TASM.ZIP must be unpacked into a working directory on the hard disk that will contain the program **TASMedit** and the sample programs.

The special feature of TASM is that is can be used for different types of microcontrollers. For each type there is a table of available machine instructions, which must be identified when the program is started. This is done using a command line; in this case we use

```
TASM -51 -b flash1.asm
  flash1.bin
```

to specify our particular example program, the instruction table TASM51.tab and binary output for-

Software

To load a program into the microcontroller on the 89S8252 Flash Board, you will need the Windows program **MicroFlash.exe**, which can be found on the *Elektor Electronics* website (www.elektor-electronics.co.uk) on the Free Downloads page, see the list for the December 2001 issue.

For the programming course, the TASM assembler is all you need to get started, but later on you will need the Rigel READ51 C compiler and the BASIC-52 Basic compiler. The TASM assembler is a popular program, which can be obtained together with TASMedit from the download list for this issue on the *Elektor Electronics* website. Please register the software and pay the programmer (T.N. Anderson) his well-earned fee. No payment is required for the C compiler, which Rigel make available free of charge for private and educational use. This compiler can be obtained from www.rigelcorp.com. BASIC-52 is a Basic interpreter created by Intel, which was mask-programmed in the program memory of an 80C52 microcontroller that received the designation 80C52-AH-BASIC. This microcontroller was used for nearly two decades by electronic engineers and programmers and became internationally famous, mainly as a result of articles in Elektor Electronics.. Several years ago, Intel ceased production of this IC, but they released the programming language as open source for general use. The language has been further developed and also adapted for use with other microcontrollers. Probably the most advanced version, V1.3, was presented in the February 2001 issue of *Elektor Electronics* and is available from Readers Services on diskette (order number **000121-11**).

The diskette for this course (Readers Services order number **010208-11**) contains the TASM assembler, TASMedit and the first sample programs, along with BASIC-52, MicrFlash.exe and a small Basic terminal emulator program with its own sample programs.

Number formats

The fact that we use a decimal number system is probably due to the fact that we happen to have ten fingers. The 'natural' number system for a computer is the binary system. The hexadecimal system represents a compromise, in which the range of numerals runs from 0 to 15, with the understanding that the numerals above 9 are represented by the letters A, B, C, D E and F.

Decimal	Hexadecimal	Binary
0	00h	00000000b
1	01h	00000001b
2	02h	00000010b
3	03h	00000011b
...	...	...
10	0Ah	00001010b
11	0Bh	00001011b
12	0Ch	00001100b
13	0Dh	00001101b
14	0Eh	00001110b
15	0Fh	00001111b
16	10h	00010000b
17	11h	00010001b
...	...	...
253	FDh	11111101b
254	FEh	11111110b
255	FFh	11111111b

In assembler programs, it is generally possible to choose which notation you want to use. If you are describing how an 8-bit port is being driven, binary notation is particularly clear. For example, the rightmost bit represents pin P1.0 and the leftmost bit represents pin P1.7. Here eight leads require eight bits, or one byte.

When TASM translates a program, you can specify the format in which the results are to be stored. In the binary format, only the bytes that represent the individual machine-language instructions are written. A text editor cannot make any sense of such a file.

The Intel hex format uses text lines containing hexadecimal numbers. In addition to the actual code, there is a start address and a checksum for each line. This format can also be viewed as text.

mat. Naturally, not everyone likes to work with command lines like this. Consequently, they won't be used at all in our course.

In general, we want to work only with Windows, but TASM is still a pure DOS program. For this reason, a Windows interface for the program, called TASMedit.exe, has been written. It includes its own editor and allows the user to immediately see the result of the translation, including possible error messages. The flash download tool for the *Elektor Electronics* Flash Board is also integrated into this program. No effort has been spared to make things as easy as possible for course participants!

The program has two text windows (see **Figure 3**). On the left there is the assembler source text editor. It can be used to manually enter a program or load a program from the hard disk. The TASM button starts the assembler in the background. The window on the right displays the assembler's list file along with any error messages that may be present. Download progress when the program is being sent to the microcontroller board is also shown in this window.

Clicking on the TASM button first generates a file called Work.asm, which contains the current content of the Editor window. This working text is then translated. TASM is called from the Windows interface using the command line

```
TASM -51 -b -work.asm work.bin
```

This means that the binary format is always used here. The result of the translation is stored in a file called Work.bin. This is the file that is read by the download module when the RUN button is actuated. The assembler also generates a file called Work.list containing the list file, which holds the translation in a readable form. The fact that the same file names are always used for the translation is an advantage for experimental work with the assembler, since the source text only has to be saved after the latest attempt has been successful. This means that we do not have a whole collection of garbage data on the hard disk from all the unsuccessful attempts, but instead only the intentionally saved source text and the work files for the most recent attempt. If you forget to save the latest version of the source text, or if the PC crashes while you are working, you can always use the Work.asm file to recover the fruits of your hard work.

When TASM is automatically started from the Windows interface, it appears in a DOS window (see **Figure 4**). This window must be closed before you can proceed. At first, it may be very enlightening to see how TASM goes about its work, but after the third time or so it will only annoy pampered Windows users. Consequently, from now on we would like to have the window be automatically closed. This is no problem, since Windows provides a solution. First click on the TASM.EXE file with the right mouse button and open the Properties menu. Under Properties / Program you will find the setting 'Close when done'. Enable this setting (see **Figure 5**). Windows then generates a link in the form of a file called TASM.PIF. From now on, the DOS window will automatically close after TASM has finished its job.

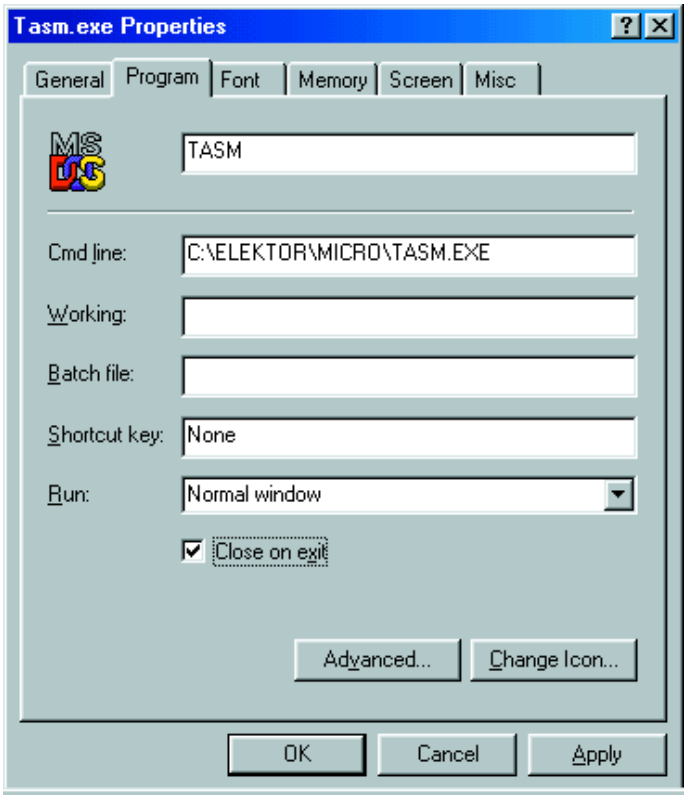


Figure 5. Setting the property 'close when done'.

Once a program has been successfully translated, the RUN button can be used to transfer it to the Flash Board system and start it. For this, you have to select a PC COM port and connect it to the board's programming connector (K2). If you have also looked after the most important prerequisite (applying the supply voltage to the Flash Board), you can then start to test the program. In the case of our first example, all you have to do is to observe the states of the Port 1 outputs in order to see whether the result is successful. Using a high-impedance meter, you should see almost exactly 5 V on port leads P1.0 through P1.3 and nearly 0 V on P1.4 through P1.7 (to be precise, around 30 mV flowing into ground).

In the ground state without any program running, or following a processor reset, all of the port leads take on the High state, with a voltage of 5 V on each pin. This can easily be checked using an oscilloscope or multimeter. The newly loaded and started program changes the states of four lines. P1.4 through P1.7 should now be Low, which means that they have a voltage of around 0 V, while P1.0 through P1.3 remain High. The program has transferred the value 15 (= 0Fh) to Port 1. This bit pattern can be seen on the port pins.

(010208-2)

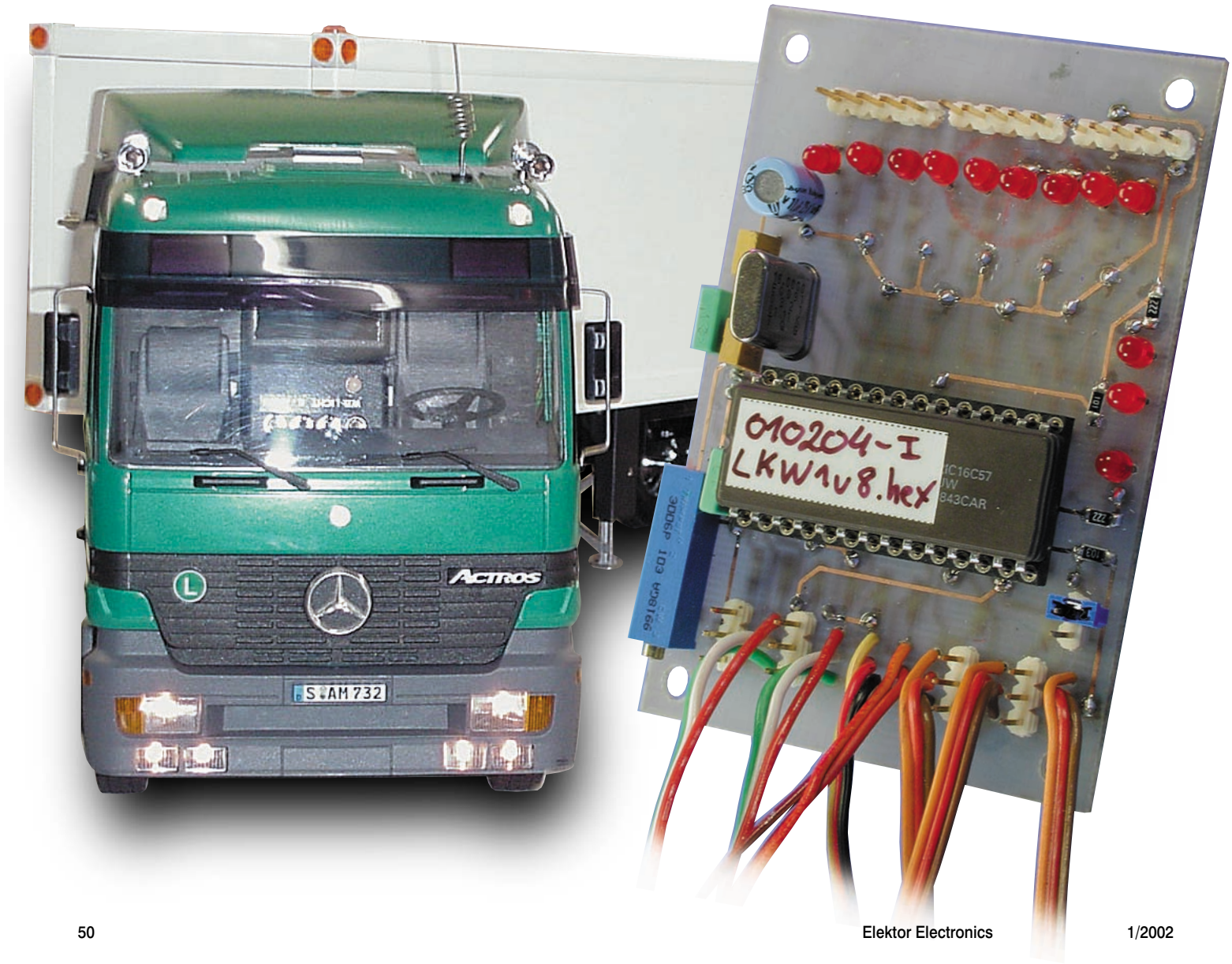
This concludes our brief introduction to working with the assembler. In the next instalment of the course, we will discuss small sample programs that are primarily intended to help investigate the port properties of the microcontroller. We will look at inputs, outputs and achievable speeds.

Lighting and Gearbox Control

for remote-control model lorries

Design by D. Dzida

The circuit described here uses a PIC microcontroller to take over the complete control of the lighting and gearbox in a model lorry. It can also be used with a fork-lift truck, excavator or similar type of vehicle.



Technical data

Supply voltage:	5V (from receiver)
Current consumption:	less than 10 mA (excluding lamp currents)
External supply voltage:	5–12V
Blinking rate:	fixed, approx. 0.5 Hz
Gearbox servo travel:	adjustable using a trimpot
Trailer coupling servo direction of rotation:	selectable (left/right) using JP1
Maximum current per output:	0.3 A

Keyed output for control only (max. 10 mA)
Relays may be connected (only with flyback diodes!)

Note regarding the maximum lamp current:
If the lamps are powered from the internal supply voltage (see Figure 5) using the receiver–speed controller BEC, the maximum output current of the speed controller BEC must not be exceeded!

The problem with these kinds of remotely controlled vehicles is that they have a large number of functions, which normally means that an expensive remote-control unit must be used. The project presented here consists of a small printed circuit board, featuring a microcontroller from the popular Microchip PIC family, which looks after the extended functions. This involves a combination of automatic function control and the decoding of supplementary functions.

The automatic functions are:
– turn indicators are automatically activated when the vehicle is turned;
– the brake lights go on when the accelerator is released;

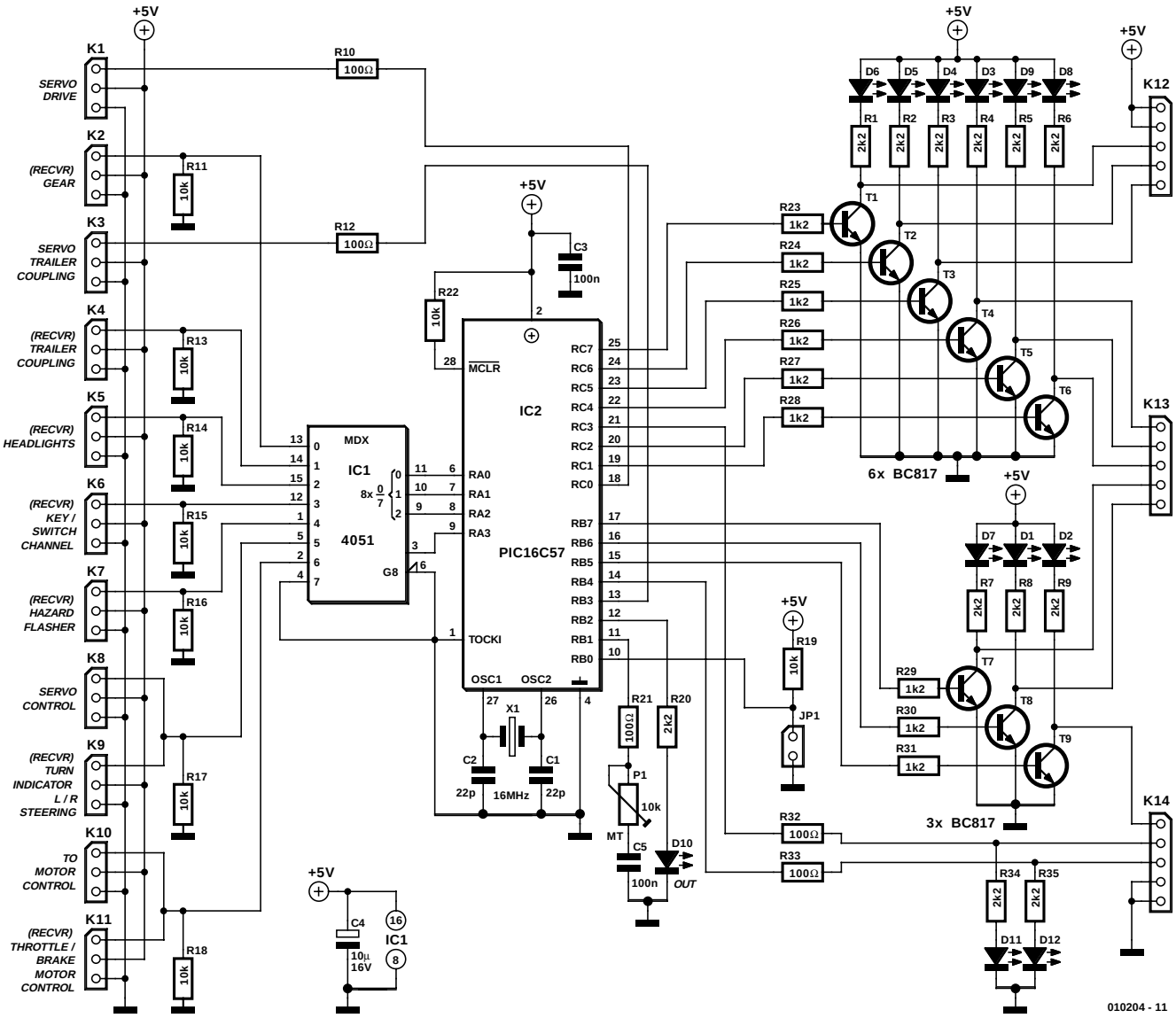


Figure 1. Schematic diagram of the lighting and gearbox controller. The microcontroller evaluates the pulse signals from the remote control receiver and generates the additional control signals for the lights and gearbox.

BEC

The acronym 'BEC' simply stands for 'battery eliminator circuit'. This is a circuit that eliminates the need for a separate battery to power the remote control by providing an adequately processed (stabilised and decoupled) supply voltage from the motor battery pack. In a model lorry, this BEC is built into the speed controller (the motor controller for the tractor). Such speed controllers always include a BEC with a 5-V output voltage. In the case of special regulators for lorries, this 5-V output can supply a heavier load than a normal regulator for cars. The amount of output current that can be supplied by the regulator depends on the model and ranges from 1 to 3 ampères.

– the back-up lights go on when the vehicle moves in reverse.

Supplementary switches on the remote control unit can be used to switch on the dipped beams, main beams, hazard flasher, flashing lights on the driver's cab, horn and additional output channels.

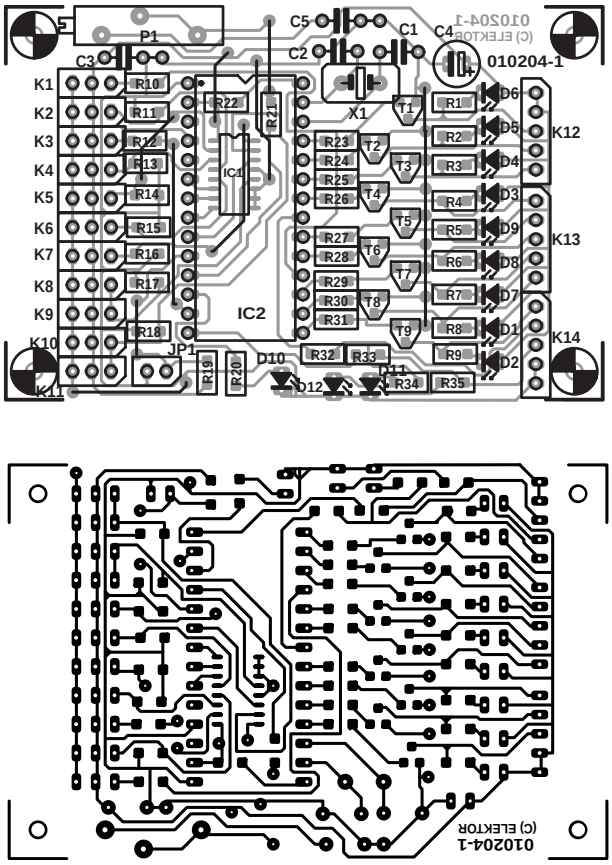


Figure 2. Printed circuit board track layout and component layouts for the copper and component sides of the single-sided circuit board.

Lamp current

The current consumption of the circuit with no lamps connected is less than 10 mA. Each of the nine outputs connected to the transistors can switch a maximum of 300 mA, which altogether yields (9 × 0.3 A) = 2.7 A! This amount of current can only be provided by an external power source (wired as shown in Figure 6).
If the internal 5-V supply voltage is used (wired as shown in Figure 5), the 5-V output of the vehicle's speed controller must supply not only the current for the remote control receiver and the servos, but also the total lamp current. Consequently, if there are many lamps, a speed controller that can provide 1.5 A or more should be used. The following is a suggestion for the lamp selection (5 V / 40 mA lamps):

dipped beams:	6 × 40 mA	=	240 mA
main beams:	2 × 80 mA	=	160 mA
brake lights:	4 × 40 mA	=	160 mA
back-up lights:	2 × 40 mA	=	80 mA
blinkers:	6 × 40 mA	=	240 mA
hazard flashers:	2 × 40 mA	=	80 mA
switched channel:	1 × 80 mA	=	80 mA
Total lamp current:			1.04 A

Furthermore, the gearbox servo can be sequentially shifted up or down using a switch or the control stick (just like a real Porsche, but with only three gears…).
Finally, there is a trailer function,

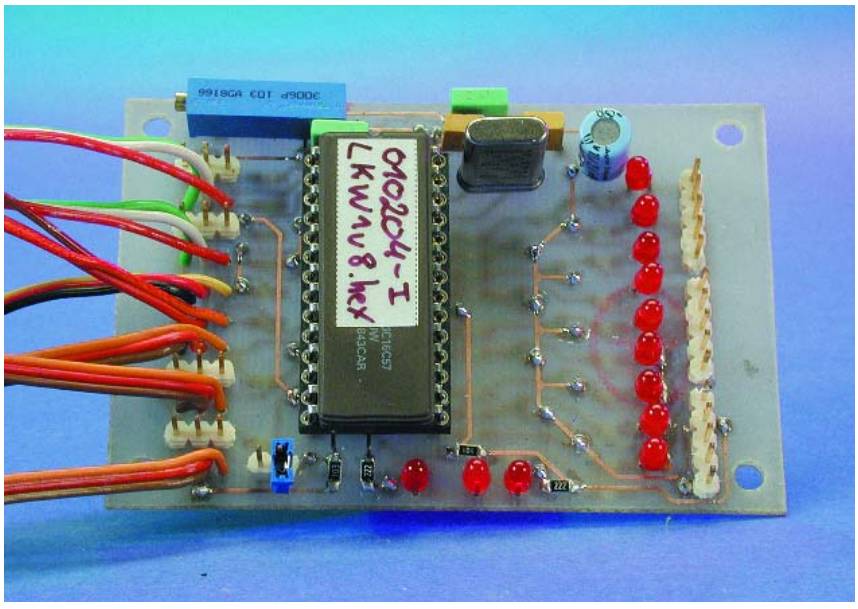


Figure 3. Top view of the prototype circuit board.

COMPONENTS LIST

Resistors:
R1-R9,R20,R34,R35 = 2kΩ2
R10,R12,R21,R32,R33 = 100Ω
R11,R13-R19,R22 = 10kΩ
R23-R31 = 1kΩ2
P1 = 10-turn preset, horizontal mounting

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

Semiconductors:
D1-D9,D11,D12 = LED, 3mm, red, high efficiency (low current)
D10 = LED, 3mm, green, high efficiency (low current)
T1-T9 = BC817
IC1 = 4051 SMD
IC2 = PIC16C57, programmed, order code **010204-41**

Miscellaneous:
JP1 = 2-way pinheader with jumper
K2,K4-K7,K9,K11 = servo connection cable
K1,K3,K8,K10 = 3-way pinheader
K12,K13,K14 = 5-way pinheader
X1 = 16MHz quartz crystal
PCB, order code **010204-1**
Disk, project source code files, order code **010204-11**

in which a single servo operates the coupling to the tractor and the associated support legs on the trailer. This double function is specially designed to be used with a Tamiya lorry trailer kit. If only the trailer coupling is needed, this function can naturally also be used with other model lorries.

Circuit operation

The heart of the controller is a Microchip PIC16C57 clocked at 16 MHz. The version in the DIL28 package is used, since this makes it easier to carry out possible software updates or extensions. The circuit, which is shown in schematic form in Figure 1, assumes the use of a battery eliminator circuit (BEC) providing a regulated supply voltage of 5 V.

The microcontroller evaluates the signals from seven output channels of the remote-control receiver. At each of these outputs, the receiver provides the well-known pulse-width modulated signal whose pulse width ranges from 1 to 2 ms (1.5 ms ± 0.5 ms). These signals, which control the gearbox, dipped/high beam, switched channel, hazard flasher, steering and motor ('gas') functions, are connected to pin headers K2, K4, K5, K6, K7, K9 and K11, respectively. From there, they reach inputs X0–X6 of the multiplexer (IC1), and from the multiplexer they arrive at port pins RA0–RA3 of the PIC microcontroller (IC2). The 4051 CMOS multiplexer could in principle be omitted, but then a larger and more expensive PIC would be needed.

The signals for the functions that are directly controlled by the remote controller (motor and steering) are not only passed on to the microcontroller, but are also looped directly through to output pin headers for connection to the steering servo (K8) and the motor speed controller (K10).

Port pin RB0 of the microcontroller is connected to jumper JP1. This jumper selects the direction of rotation of the servo for the trailer coupling. The amount of travel of the gear servo can be set using trimpot P1, which is connected to port pin RB1.

By programmed processing of the various input signals, the microcontroller generates control signals for two servo outputs (K1: gearbox servo and K3: trailer coupling servo) and eleven switched outputs that are available on pin headers K12, K13 and K14. Nine switched outputs are implemented as open-collector outputs using external transistors (T1–T9), while two output pins (of K14) are directly connected to port pins RC3 and RC4. **Table 1** presents a summary of the connectors with their designations and functions. If you are interested in the software that processes the signals in the PIC microcontroller, you can download it from the Free Downloads page on the *Elektor Electronics* website.

Each of the switched outputs is provided with a LED that indicates the output state (D1–D9 and D11). LED D10 has a special function. After the supply voltage is switched on, some of the signals from the microcontroller are first calibrated (such as the signals for the turn indicators, brake lights and back-up lights). The control stick must not be moved while this is happening. LED D10 is illuminated only after the initialisation has been successfully completed (which can take up to four seconds) in order to indicate that the circuit is ready to be used. LED D12 is connected to a switched channel that is not currently used (K14/3) and thus does not presently have any function. However, the author intends to use this output to support an infrared link between the tractor and the trailer.

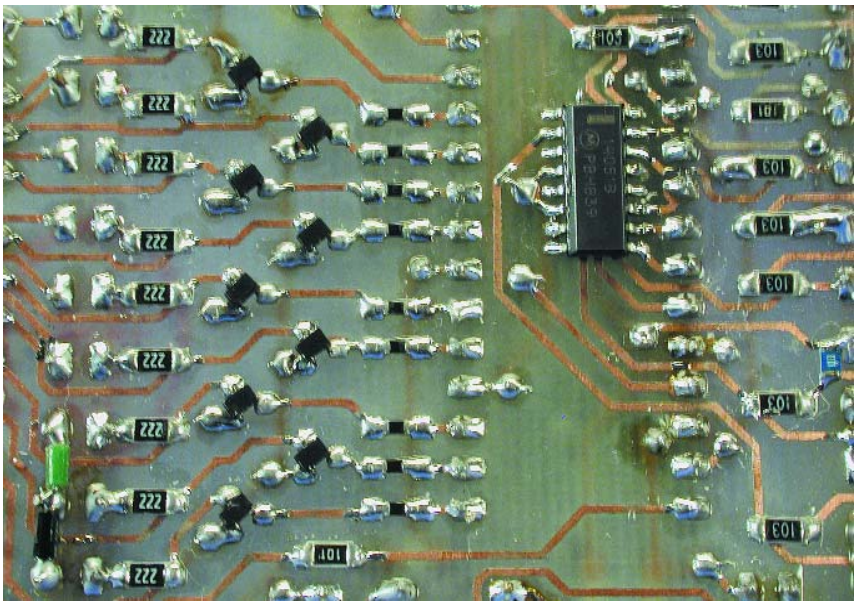


Figure 4. The SMD components are fitted on the bottom side of the circuit board.

Construction and use

Fitting the components to the printed circuit board (Figure 2) requires some special atten-

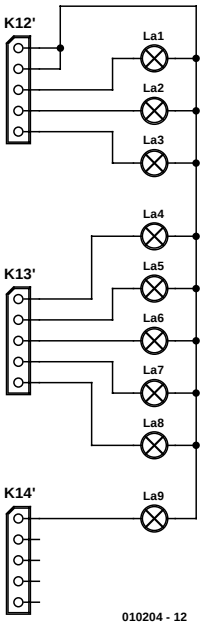


Figure 5. Lamp connections when using the internal supply voltage.

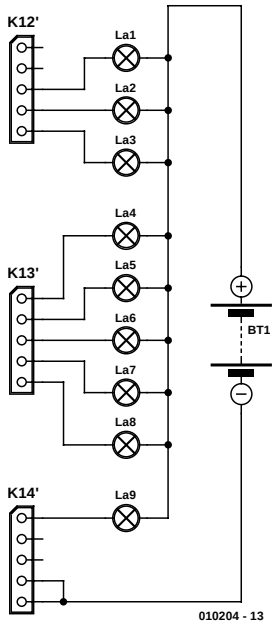


Figure 6. Lamp connections when using an external voltage source (5–12 V).

tion. Although the circuit board is single-sided, components are mounted on both sides. IC1 (a 4051 in an SMD package) is soldered to the copper side. Particular care is necessary in fitting transistors T1–T9, which are also mounted on the copper side. For fitting the components on the component side, it is best to start with the wire bridges. Attentive readers will probably notice that

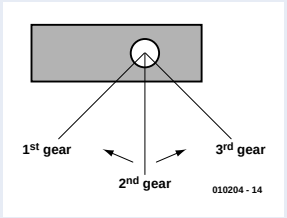
Table 1. Summary of connections and functions

Gearbox: K1 & K2

K1: Gearbox servo connection (only for a 3-speed gearbox)
K2: Connection for the cable from the receiver 'gearbox' output.
Function on the remote control unit: rocker switch with three positions: up / off / down

Function:
The amount of servo travel to the left or right of the middle position can be set using P1. Any change in the setting of P1 becomes effective only after the power is next switched on. When the power is switched on, the servo always moves to the middle position (second gear).

- rocker switch 'up' once → servo moves forward one position (e.g. from second gear to third gear).
- rocker switch 'down' once → servo moves backward one position (e.g. from second gear to first gear).

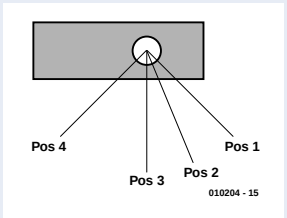


Trailer coupling: K3 & K4

K3: Connection for the trailer coupling servo (for a Tamiya lorry with electrically operated support legs – see text)
K4: Connection for the cable from the receiver 'trailer coupling' output.
Function on the remote control unit: rocker switch with three positions: up / off / down

Function:
- rocker switch 'up' → servo moves to position 1 (in this position, the support legs are lowered until a limit switch on the trailer is tripped).
- rocker switch 'off' → servo moves to position 3 (if the back-up light is not on, the coupling cannot be opened!).
- rocker switch moved to 'down' AND back-up lights ON → servo moves to position 2 (in this position, the support legs are lowered until a limit switch on the trailer is tripped).

After the rocker switch has been released (it returns automatically to the middle position), the servo moves to position 4. The trailer coupling is now open and the tractor can separate from the trailer. The trailer coupling remains open until the switch is again moved to the 'up' position (to raise the support legs and close the trailer coupling).



Dipped beam / main beam: K5

K5: Connection for the cable from the receiver 'dipped beam / main beam' output.
Function on the remote control unit: toggle switch with three positions: up / off / down

Function:
- toggle switch set to 'up' → all lights out (K12/3&4)
- toggle switch 'off' → dipped beam on (K12/3)
- toggle switch 'down' → dipped beam and main beam on (K12/3&4)

Switched / keyed channel: K6

K6: Connection for the cable from the receiver '**switched/keyed channel**' output.
Function on the remote control unit: rocker switch with three positions: up / off / down

Function:
- rocker switch 'up' once → keyed channel on (K14/1)
- rocker switch again 'up' → keyed channel off (K14/1)
- rocker switch 'down' → switched channel on (K14/2)
- rocker switch 'off' → switched channel off (K14/2)

Important note:
The switched channel is only suitable for control purposes (max. 10 mA) – do not connect any lamps or relays without using suitable driver stages!

Hazard flasher: K7

K7: Connection for the cable from the receiver 'signal / hazard flasher' output. Function on the remote control unit: rocker switch with three positions: up / off / down

Function:
- rocker switch 'up' once → hazard flasher on (blinker R/L, K13/4&5)
- rocker switch again 'up' → hazard flasher off
- rocker switch 'down' once → flashing signal lights on (K13/2/3)
- rocker switch again 'down' → flashing signal lights off

L/R turn indicators and steering servo: K8/K9

K8: connection for steering servo

K9: Connection for the cable from the receiver 'steering' output. Function on the remote control unit: steering control stick

Function:
After the supply voltage is applied, the neutral position of the control stick is stored. Do not move the control stick until LED D10 is illuminated!

- control stick to the left → left turn indicator on (k13/5)
- control stick in the middle → all turn indicators off
- control stick to the right → right turn indicator on (K13/4)

Brake/back-up lights and speed controller: K10/K11

K10: connection for speed controller

K11: Connection for the cable from the receiver 'speed controller' output. Function on the remote control unit: control stick forward / reverse

Function:
After the supply voltage is applied, the neutral position of the control stick is stored. Do not move the control stick until LED D10 is illuminated!

- control stick in the middle position → brake lights on (k12/5)
- control stick forward → brake and back-up lights off
- control stick reverse → back-up lights on (k13/1)
- control stick moved from the reverse position to the middle position → brake and back-up lights on.

Web addresses

www.robbe.de	full range of products
www.multiplex-rc.de	full range of products
www.schulze-elektronik.com/index_uk.htm	regulators, chargers
www.kontronik.com/gate_engl.htm	regulators, motors
www.hacker-motor.com/english/englisch.html	motors
www.wedico.de/index_eng.html	model lorries

the arrangement of the components on the circuit board does not completely match the arrangement shown on the prototype board in the photos (**Figures 3 and 4**). This is because the final circuit board layout was modified slightly from that of the prototype, but the two circuit boards are electrically identical.

All essential information regarding connector pin assignments, wiring and use of the circuit board is presented in **Table 1**, with one exception: the connections for the lamps are shown in **Figures 5 and 6**. Outputs RB5–RB7 and RC0–RC7 of the PIC directly control the connected lights via the BC817 transistors. Since these outputs are implemented as open-collector outputs, the lamps can be connected either directly to the circuit board (as in Figure 5) or via an external supply voltage (using a separate battery as shown in **Figure 6**). In the latter case, one lead of each lamp is connected to the positive lead of the external supply voltage, which may lie in the range of 5–12 V. When connecting the lamps, take particular care to avoid any possibility of a short circuit between the +5-V supply voltage on pins 1 and 2 of K12 and the external supply voltage. Such a short circuit could destroy both the PIC and the remote control receiver!

Note that if the lamps are connected directly to the circuit board (as shown in Figure 5), the entire current for the lamps must be provided by the speed controller. Special lorry regulators can provide more current at the BEC output (5 V) than normal regulators for cars. In any case, the ratings of the lamps must be chosen such that the BEC supply is not overloaded (see the 'Lamp Current' sidebar). Relays can also be connected to the open-collector outputs, but only if they are fitted with flyback diodes. The maximum relay coil current is subject to the same considerations as the maximum lamp current.

(010204-1)

USB UART (2)

part 2: setting the port currents

Although the *Elektor Electronics* USB interface described in the September 2000 issue allows the port current to be set for only one port, the port IC allows the current to be set for each pin.

The module `USBuart.bas` provides the subroutine `Sub WrIsink Pin, Wert`. The parameter `Pin` may have a value of 0 through 7 for port pins P00 through P07 or 8 through 11 for port pins P10 through P13. **Listing 1** shows a simple program for setting the sink currents of all four Port 1 outputs, while Figure 1 shows the associated screen display.

The microcontroller contains a simple 4-bit DAC for each port pin, consisting of four weighted current sources. We can ask ourselves if we can't use this for something more worthwhile than just controlling the brightness of a few LEDs. In principle, a resistor is all we need to convert the controlled current into an output voltage. A 200-Ω resistor connected to Port 1 (see **Figure 2**) yields a good output voltage range for a sink current of up to 15 mA.

To check the linearity of this arrangement, the output voltage was measured for all 16 current settings. The results are shown in **Table 1**. These results have also been evaluated graphically using Excel. As can be seen from **Figure 3**, the linearity is good.

The current output can be used to make a simple programmable power supply. **Figure 4** shows a sample circuit using an L272 power opamp. The zero crossing and slope can be independently adjusted. If necessary, the adjustment range can be made smaller, for example 3.5 to 5 V.

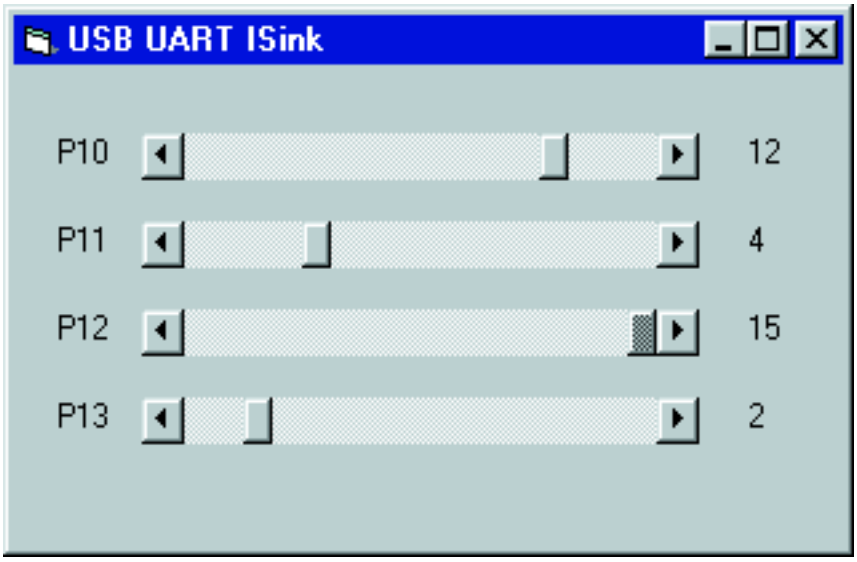


Figure 1. Screen display for the program LED.FRM.

A simple A/D converter

We can also utilise the D/A settings of the port pins to construct a simple A/D converter. In this case, all we want to do is measure resistance values. The resistor to be measured must be connected between a port pin and V_{CC}. The method is based on the experimentally determining the sink current setting needed to have the voltage on the port pin in question appear to be Low, given the connected resistor value.

Figure 5 shows how a simple analogue input function can be obtained by connecting eight potentiometers to Port 1. For Port 0, an adjustment range of around 2 kΩ to 12 kΩ gives the best results. Lower resistance values are needed for the same arrangement on Port 1, since this port has a greater sink current.

For each analogue conversion, the sink current is increased stepwise until the comparator switches states. This means that each measurement requires up to 16 output transactions and 16 input transactions. Since a control access via the USB port requires 4 ms, each measurement takes up to 256 ms. Consequently, a set of measurements for all eight channels will take around two seconds.

Listing 2 shows a program for

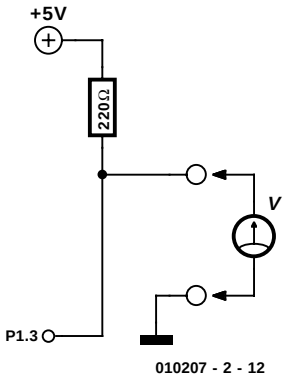


Figure 2. Controlling an output voltage via P1.3.

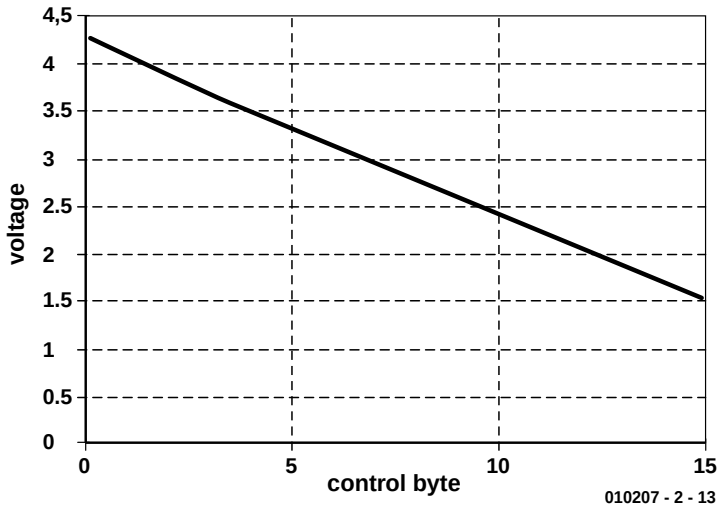


Figure 3. Output voltage vs. control byte value.

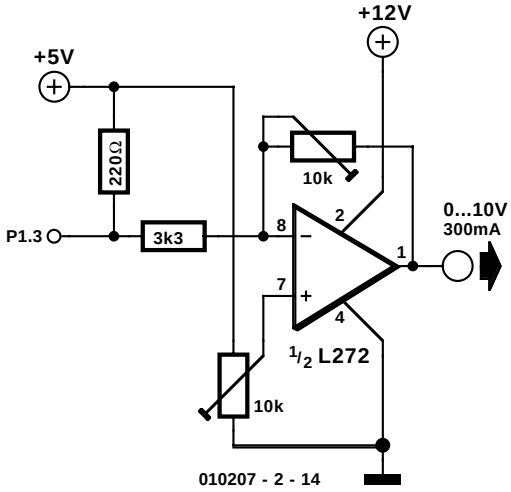


Figure 4. A programmable voltage source.

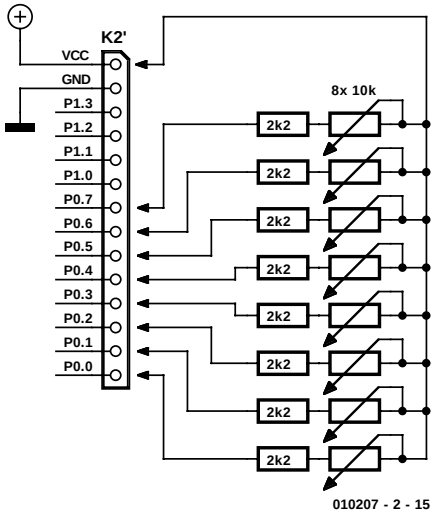


Figure 5. Connecting potentiometers for analogue inputs.

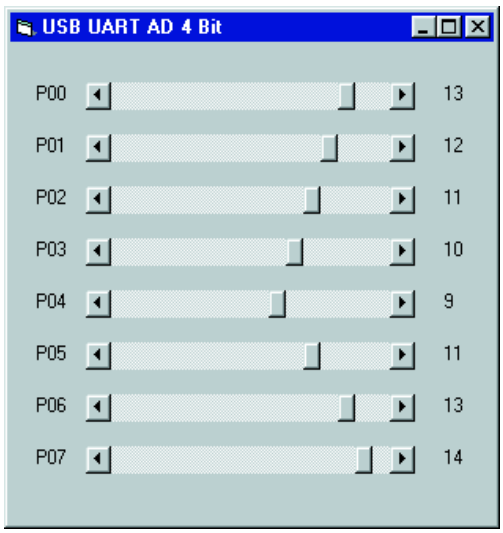


Figure 6. An 8-channel analogue display.

Table 1.
Voltages measured on P1.3.

Current step	Voltage
0	4.29 V
1	4.09 V
2	3.90 V
3	3.70 V
4	3.51 V
5	3.32 V
6	3.15 V
7	2.96 V
8	2.79 V
9	2.61 V
10	2.43 V
11	2.24 V
12	2.06 V
13	1.89 V
14	1.71 V
15	1.54 V

Listing 1.
USBuart2.vbp

```
Private Sub Form_Load()
  WrPort0 Wert
End Sub

Private Sub HScroll1_Change()
  Wert = HScroll1.Value
  WrIsink 8, Wert
  Label5.Caption = Str$(Wert)
End Sub

Private Sub HScroll2_Change()
  Wert = HScroll2.Value
  WrIsink 9, Wert
  Label6.Caption = Str$(Wert)
End Sub

Private Sub HScroll3_Change()
  Wert = HScroll3.Value
  WrIsink 10, Wert
  Label7.Caption = Str$(Wert)
End Sub

Private Sub HScroll4_Change()
  Wert = HScroll4.Value
  WrIsink 11, Wert
  Label8.Caption = Str$(Wert)
End Sub
```

polling and displaying eight port settings. For a change, here we use sliders as indicators rather than control elements. The sliders shown on the screen are remotely controlled by the potentiometers. (010207-2)

Listing 2.
Using the USB-UART as an 8-channel A/D converter

```
Function ADCh0()
  WrIsink 0, 0
  Ain = 0
  While ((RdPort0 And 1) > 0) And (Ain < 15)
    Ain = Ain + 1
    WrIsink 0, Ain
  Wend
  ADCh0 = Ain
End Function

Function ADCh1()
  WrIsink 1, 0
  Ain = 0
  While ((RdPort0 And 2) > 0) And (Ain < 15)
    Ain = Ain + 1
    WrIsink 1, Ain
  Wend
  ADCh1 = Ain
End Function

Function ADCh2()
  WrIsink 2, 0
  Ain = 0
  While ((RdPort0 And 4) > 0) And (Ain < 15)
    Ain = Ain + 1
    WrIsink 2, Ain
  Wend
  ADCh2 = Ain
End Function

Function ADCh3()
  WrIsink 3, 0
  Ain = 0
  While ((RdPort0 And 8) > 0) And (Ain < 15)
    Ain = Ain + 1
    WrIsink 3, Ain
  Wend
  ADCh3 = Ain
End Function

Function ADCh4()
  WrIsink 4, 0
  Ain = 0
  While ((RdPort0 And 16) > 0) And (Ain < 15)
    Ain = Ain + 1
    WrIsink 4, Ain
  Wend
  ADCh4 = Ain
End Function

Function ADCh5()
  WrIsink 5, 0
  Ain = 0
  While ((RdPort0 And 32) > 0) And (Ain < 15)
    Ain = Ain + 1
    WrIsink 5, Ain
  Wend
  ADCh5 = Ain
End Function

Function ADCh6()
  WrIsink 6, 0
  Ain = 0
  While ((RdPort0 And 64) > 0) And (Ain < 15)
    Ain = Ain + 1
    WrIsink 6, Ain
  Wend
  ADCh6 = Ain
End Function

Function ADCh7()
  WrIsink 7, 0
  Ain = 0
  While ((RdPort0 And 128) > 0) And (Ain < 15)
    Ain = Ain + 1
    WrIsink 7, Ain
  Wend
  ADCh7 = Ain
End Function

Private Sub Form_Load()
  WrPort0 Wert
End Sub

Private Sub Timer1_Timer()
  Value = ADCh0()
  HScroll1.Value = Value
  Label9.Caption = Str$(Value)
  Value = ADCh1()
  HScroll2.Value = Value
  Label10.Caption = Str$(Value)
  Value = ADCh2()
  HScroll3.Value = Value
  Label11.Caption = Str$(Value)
  Value = ADCh3()
  HScroll4.Value = Value
  Label12.Caption = Str$(Value)
  Value = ADCh4()
  HScroll5.Value = Value
  Label13.Caption = Str$(Value)
  Value = ADCh5()
  HScroll6.Value = Value
  Label14.Caption = Str$(Value)
  Value = ADCh6()
  HScroll7.Value = Value
  Label15.Caption = Str$(Value)
  Value = ADCh7()
  HScroll8.Value = Value
  Label16.Caption = Str$(Value)
End Sub
```

Music Industry Protecting Audio CDs

intentionally introducing errors to prevent copying

By Harry Baggen

The illegal copying of audio CDs costs the music industry a bundle every year. A number of companies are now using sophisticated techniques to try to protect their CDs against copying. However, it's questionable whether this actually helps and whether the consumer actually benefits.

For years, the music industry has been complaining about heavy losses from the illegal copying of audio CDs. Most computer owners these days have a CD burner in their system, and with modern, fast CD recorders it only takes a few minutes to make a copy of a CD for a friend or neighbour. Naturally, this is not how it's supposed to be. Not only does the producer / distributor loose income, but the songwriters and musicians who have done their best to put something nice on the CD also lose out. According to the law, you can make a copy of a phonograph record or CD for your own use, but you're not supposed to buy a CD and then make ten copies (or let someone else make ten copies) for all your friends.

In order to put an end to home copying, various attempts have been made to devise methods to protect audio CDs against digital copying on a computer. Several companies have now come up with a number of different systems, and naturally they all insist that their methods work perfectly. Various large music producers also use these methods, but often without making known which CDs are affected.

The inventors of the various protection methods are naturally reluctant to provide very much information about the techniques used, but all of them essentially amount to modifying the data on the CD to such a degree that a PC can no longer make an exact copy of the CD using a CD-ROM drive. The best-known technique is that used by Macrovision. In their 'Safe Audio' system, the audio data and error correction codes on the CD are intentionally mutilated to the point that a CD-ROM drive in a computer will have trouble handling the data. As a result, a file that is read in digitally will produce all sorts of noise and

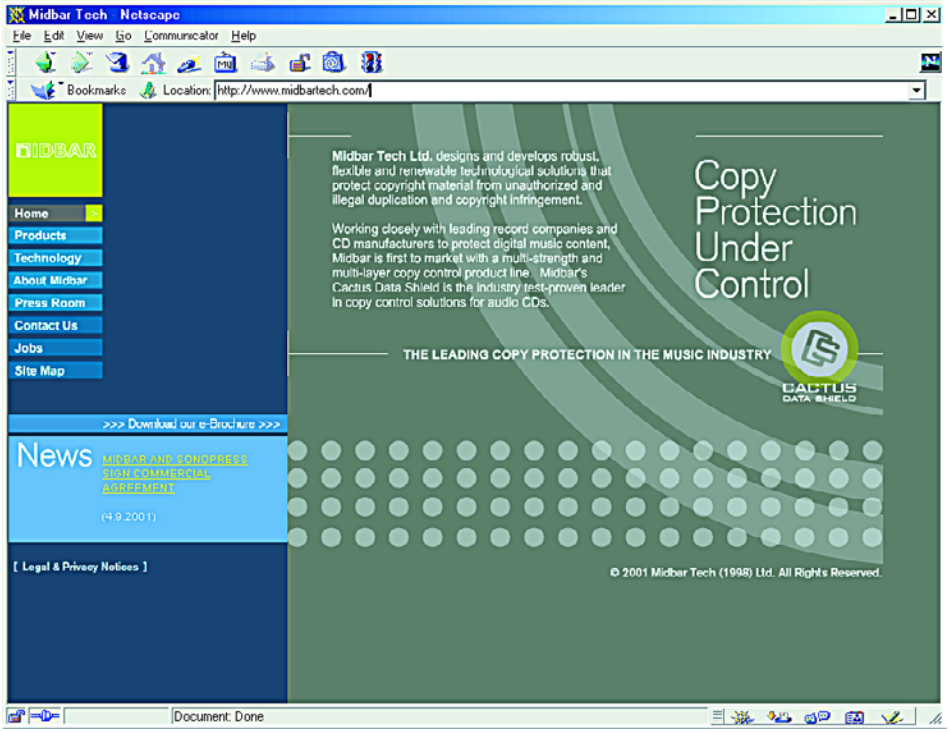
interference on playback, so that it is no longer worthwhile to copy it to a CD-R. Normal audio CD players prove to have no problems with such data, since even though the error correction system in such a player gets hopelessly confused by the bogus data, the player 'invents' intermediate values by means of interpolation.

We do not find such methods particularly attractive. First the industry does its best to devise a CD standard that provides the purest possible reproduction quality and a high degree of error tolerance (to compensate for scratches and the like), only to turn around and intentionally introduce gross errors.

Protests against this method are becoming slowly but steadily stronger and more numerous, particularly on the part of audiophiles, who fear that the protection techniques may have a negative effect on sound quality – and we must say that we agree! Among others, the well-known British hi-fi expert Martin Colloms is absolutely opposed to such methods. He compares them to splashing paint all over pictures in art galleries to prevent them from being stolen. You can find more on the subject in articles in *New Scientist* [1] and *New Media Music* [2]. In the US, a complaint has already been lodged against a music company that put protected CDs on the market without a clear notice on the packaging.

Which companies are involved in protecting CDs? The best known and largest is the already-mentioned Macrovision [3], known among other things for the video protection system with the same name.

Macrovision's 'Safe Audio' technique was originally developed by the Israeli company TTR Technologies. Macrovision claim that they carried out listening tests for several months using both lay persons and people with 'golden ears', with the result that no-one was able to detect any changes in the music signal resulting from the copy protection. However, they are not willing to identify the CDs to which this technique has been applied, although there are apparently a number of well-known titles that have been produced in large volumes and the number of complaints (according to Macrovision) is minimal.



Sony, which in addition to making consumer products is also one of the major players in the audio industry and a maker of CDs (Sony DAC C), use a technique of their own called Key2Audio [4]. They claim that this process does not corrupt the music data, but that instead a sort of digital fingerprint is added when the glass master for the CD is made, with the result that digital copying is no longer possible.

Cactus Data Shield is a technique developed by Midbar [5] that is presently widely used. Apparently, at least one million CDs have already been protected using this system. Not much is known about this system.

The MediaCloQ technique from SunnComm [6] uses yet another method. A CD that has been protected using this method can be recognised by a clearly different reflection from the region at the end of the audio track on the CD.

The website of CD Media World [7] provides a good summary of all current protection systems, with a brief explanation of each system.

With all this fuss about copy protection for audio CDs, you might almost forget that it is still perfectly easy to make a copy of a CD by just using the normal audio output. The primary disadvantage of this is that it takes so much time, and in addition there is a fractional loss of sound quality due to the extra D/A and A/D conversions. The question is whether the average consumer is really concerned – after all, most consumers are happy with the distinctly lower audio quality of MP3 files downloaded from the Internet.

(025005-1)

Internet addresses:

[1] NewScientist:
www.newscientist.com/news/news.jsp?id=ns9999998

[2] New Media Music:
www.newmediamusic.com/articles/NM01080292.html
www.newmediamusic.com/articles/NM01100072.html

[3] Macrovision SafeAudio:
www.macrovision.com/solutions/newtech/safeaudio.php3

[4] Sony DADC Key2Audio:
www.key2audio.com/key2audio/index.htm

[5] Midbar Cactus Data Shield
www.midbartech.com/cactus2.html

[6] SunnComm MediaCloq:
www.sunncomm.com/

[7] CD Media World:
www.cdmediaworld.com/hardware/cdrom/cd_protections.shtml

Infrared Transceiver for the PC (1)

combined remote control and data link

Design by B. Kainka

This article does more than just describe an IR transceiver that you can use with your PC for remote control and data transfer. Based on the successful *Elektor Electronics* 'PC Peripheral Design' series, it also provides information about the principles of the infrared transfer technique used and interface technology, which you can use for your own projects.

Infrared data transfer is becoming increasingly more important. Television remote controls use infrared light, but so do PC mice, keyboards, printers and other peripherals. Infrared light is also used in fibre-optic cables. In a sense, the PC infrared transceiver presented here is a practical application that represents a continuation of the series of articles on PC interfaces and Visual Basic that appeared in the 'PC Serial peripheral Design' course (*Elektor Electronics* 9/2000 through 3/2001).

Infrared remote controls for television sets, video recorders and other entertainment electronics devices frequently use the Philips RC5 standard. It is easy to determine whether a particular remote control employs RC5 by using the program described in this article. This standard employs light signals modulated at a frequency in the range of 30 to 40 kHz. The remote control unit transmits 'bursts', which are individual pulse packets. In our case, these bursts have a duration of either 0.888 ms or 1.776 ms. At a modulation frequency of 36 kHz, a short burst contains 32 individual pulses, while a long burst contains 64 pulses. A complete data packet has a duration of approximately 25 ms and is repeated every 100 ms as long as a button is held depressed.

An infrared remote control unit can easily be used for other purposes. For example, it can be used to control certain functions of your own program. A typical application is

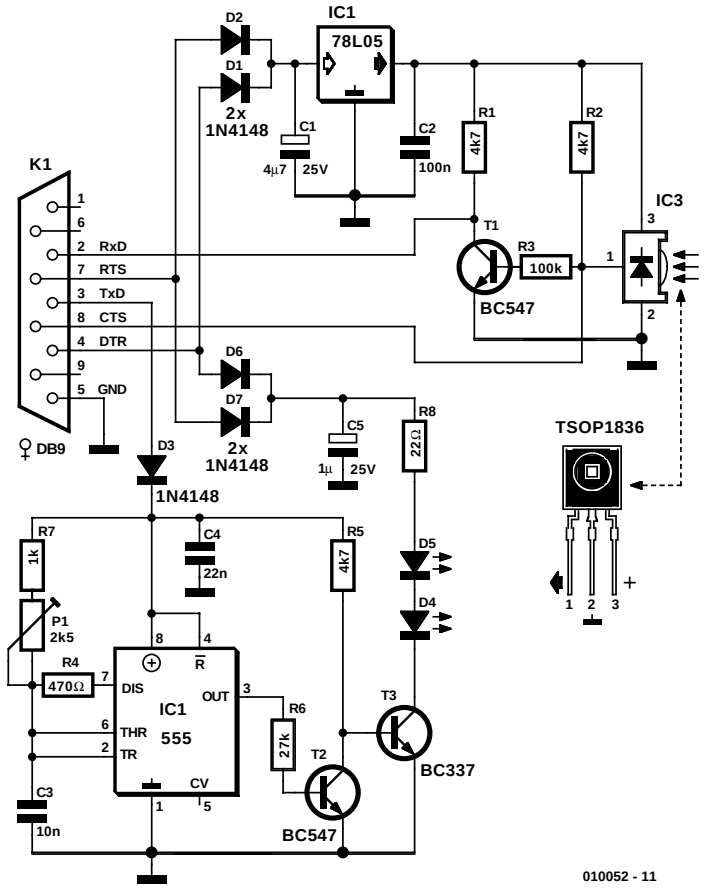


Figure 1. Schematic diagram of an IR transceiver for connection to the serial interface of a PC.

controlling a PC slide show. After starting the program, you can sit back and operate everything from where you are sitting.

Transceiver hardware

Thanks to the availability of integrated receivers, the reception of standard infrared signals is relatively simple. The well-known Siemens SHF506 is available with fixed modulation frequencies of 30 kHz, 33 kHz, 36 kHz and so on. The filter curve is relatively broad-band, so deviations of a few kilohertz cause only a relatively small reduction in sensitivity. The Vishai/Telefunken TSOP1836 is a similar IC. Both types of IC need only a 5-V supply voltage and draw less than 2 mA. They can thus be powered directly from the PC serial interface.

The IR transceiver described here has a modulated IR transmitter in addition to the receiver. This light transmitter works with a carrier frequency between 30 and 40 kHz. It can be used for the remote control of devices such as video recorders and television sets, but it can also be used for data transfer between two PCs.

The schematic diagram shown in **Figure 1** reveals the receiver IC (IC3) and a 78L05 voltage regulator (IC1). The supply voltage is taken from the DTR and RTS outputs of the RS232 interface, which are connected together via the isolating diodes D1 and D2. A voltage of around 10 V can be activated here using a program running on the PC. These two leads also power the transmitter portion of the circuit via D6 and D7. Since high pulse currents are needed for transmitting, a relatively large electrolytic capacitor (IC1, 4.7 µF) is used to smooth the input voltage of the voltage regulator.

If the IR receiver IC receives an infrared signal modulated at 36 kHz, it produces an output signal on its middle pin with an active-low level.

IR Transceiver

Technical specifications

- Reception frequency: 30, 33 or 36 kHz, depending on the IC version fitted
- Transmission frequency: 30–40 kHz, continuously adjustable
- Power supply: from the serial interface
- Range: approx. 10 m
- IR receiver: for remote control per the RC5 standard
- IR transmitter: RC5 compatible
- IR data transceiver: serial data, 2400 baud max.

These output pulses are connected directly to the CTS lead, where they must be decoded using software. A supplementary pull-up resistor is necessary here, since the CTS lead has a relatively low input resistance. The signal is also inverted by a transistor stage (T1) and applied to the RxD input of the serial interface. This lead serves for the reception of fast data, for example from a data link between two PCs.

The infrared transmitter consists of a modulation stage (IC2) and a pulse amplifier (T2 and T3) driving two infrared diodes (D4 and D5). IC2 is a 555 timer IC wired as an oscillator, which generated narrow negative pulses with a width of around 2 µs. The frequency can be set between approximately 30 and 40 kHz using P1. Depending on the application, the trimpot can be used to tune the circuit to the suitable frequency in order to achieve the greatest possible range. The 555 receives its supply voltage from the TXD lead, which modulates ('keys') the transmitter by switching it on and off.

The energy for the two IR transmitter diodes also comes from the serial interface. The DTR and RTS leads charge a 1-µF electrolytic capacitor (C5) via D6 and D7. The brief pulses on the output of the timer IC (pin 3) force the driver stage consisting of T2 and T3 into full conduction. This results in pulsed currents of approximately 200–300 mA. The range that is thus achieved is around 10 m. Although the charging current from the serial interface is relatively small at 40 mA, there is enough time between the pulses to allow the capacitor to recharge.

This simple hardware can be constructed using the printed circuit board shown in **Figure 2**. All that you need to watch out for when fitting the components is to make sure that the diodes, electrolytic capacitors and ICs are soldered in or inserted the right way around. You should also avoid the common

mistake of using a sub-D socket (9 holes) instead of a sub-D plug (9 pins).

RD5 software decoder

The data output of the receiver IC is connected directly to the CTS lead. Signals from a remote control unit that have been demodulated by the receiver IC thus appear on this lead. A program for decoding the signal only has to evaluate the incoming pulses in order to recognise which button has been pressed on the remote control.

Figure 3 shows a signal received from a RC5 remote control unit. The diagram was captured using a logic analyser. A program directly records the changes of the signal level on the CTS lead. The RC5 protocol uses what is called a 'bi-phase' signal, with the actual information being contained in the phase changes. The signal level changes at least every 1.776 ms. The receiver can continuously resynchronise to the signal by means of these changes.

The signal begins with a start sequence that is always the same. Following this come three data regions, in which level changes spaced 1.776 ms apart represent the actual data bits. Following each level change, the receiver first waits for slightly longer than 0.888 ms and skips any level change that may occur in this interval. The next following level change is both a synchronisation signal and a data bit. In principle, this technique can be used to transfer data words of any desired length. In the case of RC5 signals, the word length is exactly 12 bits, composed a follows:

- The Control Bit (Ctl) changes between 0 and 1 each time a button is pressed. The receiver can use this information to decide whether a button has been pressed and held only once or has been pressed several times in succession.
- The Device Address (Addr) consists of five bits, with the most significant bit being transmitted first. Some standard device addresses are '1' for a television set and '5' for a video recorder. The Device Address allows several different remote controls to be used in the same room.
- The Data Region (Dat) consists of six bits for up to 64 different buttons. The number buttons (0–9) generate the codes '0' through '9'. Here again the most significant bit is transmitted first.

Listing 1 shows the actual software decoder program in Visual Basic. The routine 'RC5' receives the data. Here the PORT.DLL from the book *PC Interfaces under Windows* is used for all accesses to the serial interface and for timing control. (PORT.DLL can be downloaded free of charge from the *Elektor*

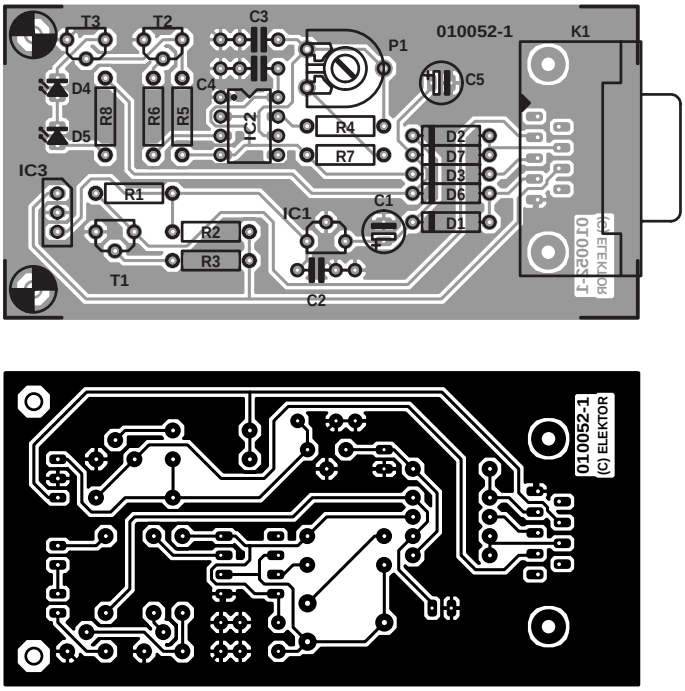


Figure 2. Printed circuit board layout and component layout for the IR transceiver (board not available ready-made).

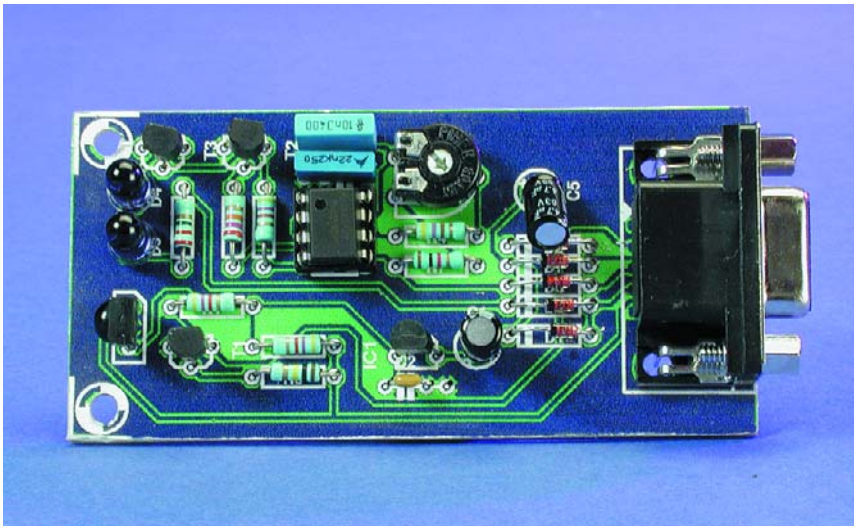
COMPONENTS LIST

Resistors:
R1,R2,R5 = 4kΩ7
R3 = 100kΩ
R4 = 470Ω
R6 = 27kΩ
R7 = 1kΩ
R8 = 22Ω
P1 = 2kΩ5 preset

Capacitors:
C1 = 4µF7 25V radial
C2 = 100nF
C3 = 10nF
C4 = 22nF
C5 = 1µF 25V radial

Semiconductors:
D1,D2,D3,D6,D7 = 1N4148
D4,D5 = IR-LED, e.g. LD271
T1,T2 = BC547
T3 = BC337
IC1 = 78L05
IC2 = 555
IC3 = TSOP1836, SFH506-36

Miscellaneous:
K1 = 9-way Sub-D socket (female), angled pins, PCB mount



Listing 1

Receiving and displaying RC5 data

```
Dim Ctr
Dim Adr
Dim Dat

Sub RC5Error()
  Ctr = -1
  Adr = -1
  Dat = -1
End Sub

Function RC5Bit() As Integer
  TIMEINITUS
  If CTS = 0 Then
    While ((TIMEREADUS < 500) And (CTS = 0))
    Wend
    If TIMEREADUS > 499 Then RC5Error
    DELAYUS 444
    If CTS = 0 Then RC5Error
    RC5Bit = 0
  Else
    While ((TIMEREADUS < 500) And (CTS = 1))
    Wend
    If TIMEREADUS > 499 Then RC5Error
    DELAYUS 444
    If CTS = 1 Then RC5Error
    RC5Bit = 1
  End If
End Function

Sub RC5()
  Ctr = 0
  Adr = 0
  Dat = 0
  Startbit = False
  REALTIME True
  TIMEINIT
  While Not Startbit
    While (CTS = 1) And (TIMEREAD < 500)
    Wend
    Startbit = True
    DELAYUS 444
    If CTS = 1 Then Startbit = False
    DELAYUS 888
    If CTS = 0 Then Startbit = False
    DELAYUS 888
    If CTS = 1 Then Startbit = False
    Adr = Startbit
    If TIMEREAD > 499 Then Startbit = True
  Wend
  DELAYUS 888
  Ctr = RC5Bit
  Adr = 0
  For N = 1 To 5
    DELAYUS 888
    Adr = Adr * 2
    Adr = Adr + RC5Bit
  Next N
  Dat = 0
  For N = 1 To 6
    DELAYUS 888
    Dat = Dat * 2
    Dat = Dat + RC5Bit
      Next N
  REALTIME (False)
End Sub

Private Sub Form_Load()
  OPENCOM "COM2"
  DTR 1
  RTS 1
End Sub

Private Sub Form_Unload(Cancel As Integer)
  CLOSECOM
End Sub

Private Sub Timer1_Timer()
  RC5
  Text1.Text = Str$(Ctr) + " " + Str$(Adr) + " " +
Str$(Dat)
End Sub
```

Electronics website; see the note at the end of the article.) This task is relatively time-critical and requires the use of REALTIME=True. This routine initially waits for a low level, which acts as a start pulse. In order to prevent the PC from hanging in an infinite loop if no signal is present, a timeout condition is built in. If no signal has been received after 500 ms, the program terminates with an error message.

Infrared controls are always subject to

Listing 2

Loading pictures for a slide show

```
Private Sub Timer1_Timer()
  RC5
  If Dat <> Dat_old Then
   If Dat = 1 Then Picture1.Picture = LoadPicture("D:\Homepage\Bast11.jpg")
   If Dat = 2 Then Picture1.Picture = LoadPicture("D:\Homepage\Bast21.jpg")
   If Dat = 3 Then Picture1.Picture = LoadPicture("D:\Homepage\Bast31.jpg")
   If Dat = 4 Then Picture1.Picture = LoadPicture("D:\Homepage\Bast41.jpg")
   If Dat = 5 Then Picture1.Picture = LoadPicture("D:\Homepage\Bast51.jpg")
   If Dat = 6 Then Picture1.Picture = LoadPicture("D:\Homepage\Bast61.jpg")
   If Dat = 7 Then Picture1.Picture = LoadPicture("D:\Homepage\Bast71.jpg")
   If Dat = 8 Then Picture1.Picture = LoadPicture("D:\Homepage\Bast81.jpg")
   If Dat = 9 Then Picture1.Picture = LoadPicture("D:\Homepage\Bast91.jpg")
  End If
  Dat_old = Dat
End Sub
```

interference from other light sources. Fluorescent lamps, which generate rapidly flickering light, are typical interference sources. The RC5 reception routine thus checks the start sequence of the received signal to verify that it is correct. If the signal departs from the expected pulse sequence, an interference pulse must be involved. In this case, the routine waits for the next start sequence. This makes it possible to securely receive RC5 signals, even in an environment with a relatively high level of interference.

After the start sequence, the individual bits are read by the routine 'RC5bit'. If a 0 level is read at the start of the routine, this should represent a 0 bit. Next, the routine waits for the signal level to change. After half the pulse width (444 µs), a new query is made to see whether the same level is still present. If this is not the case, an error is detected and all data read up to this point are overwritten with the value –1. After slightly more than 444 µs, the routine 'RC5bit' returns the value of the bit that was read. The calling routine ('RC5') evaluates this bit and waits 444 µs before again calling 'RC5bit'. In this manner, any subsequent level change that may be present is skipped. This is proper, since the next valid change is only expected to take place after 1.776 ms.

Signal reception is controlled by a timer in the VB program. All received data are displayed in a text window. Figure 4 shows the screen output of the received data for a remote control unit with device address '5' (video recorder) and button '7' pressed.

If no signal is received, a suitable error message is displayed with Ctl = –1 and Addr = –1. In an operational version of the program, the error information would not be displayed but would instead be used to distinguish usable data from unusable data.

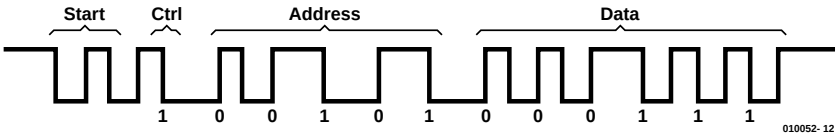


Figure 3. Sample RC5 signal with address '5' and button '7' pressed.

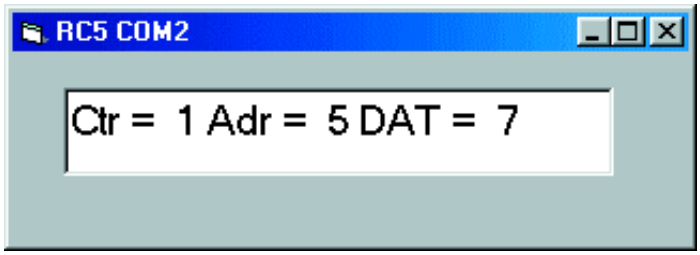


Figure 4. Outputting RC5 data.

Figure 5. A slide show on the PC monitor.

You can use this program to familiarise yourself with the individual codes of your own remote control unit. This can form the basis for later control applications in which the PC replaces a remote control unit. Complete lists of the commonly used codes can be found on the Internet.

PC remote control

Another useful application is remote control of the PC. For example, you could control your own slide show using a remote control unit. **Listing 2** shows a program segment with the modified timer routine. Here nine different button codes are evaluated to load pictures from the hard disk. The fact that this program supports only nine pictures may come as a relief to those of you who have suffered through seemingly endless slide shows, but there is nothing to prevent this number from being increased. For example, you could use the '+' and '–' buttons to control the slide sequence.

(010052-1)

In next month's issue, the software for a RC5 transmitter using a PC and the IR transceiver circuit board will be described. These can be used to control a video recorder and other devices using a PC and to transfer serial data between two PCs via the infrared link.

Download note

The PORT.DLL used in this article, as well as the program listings and the printed circuit board layout, can be downloaded free of charge from the Elektor Electronics website at *www.elektor-electronics.co.uk*. On the home page, you can find the download pages by clicking on the 'Free Downloads' button and then the month in which the article in question was published.
The software files can also be obtained from the home page of the author:
http://home.t-online.de/home/B.Kainka

Reference:

B. Kainka, **PC Interfaces under Windows**,
Elektor Electronics (Publishing), Dorchester (ISBN 0 905705 65 3