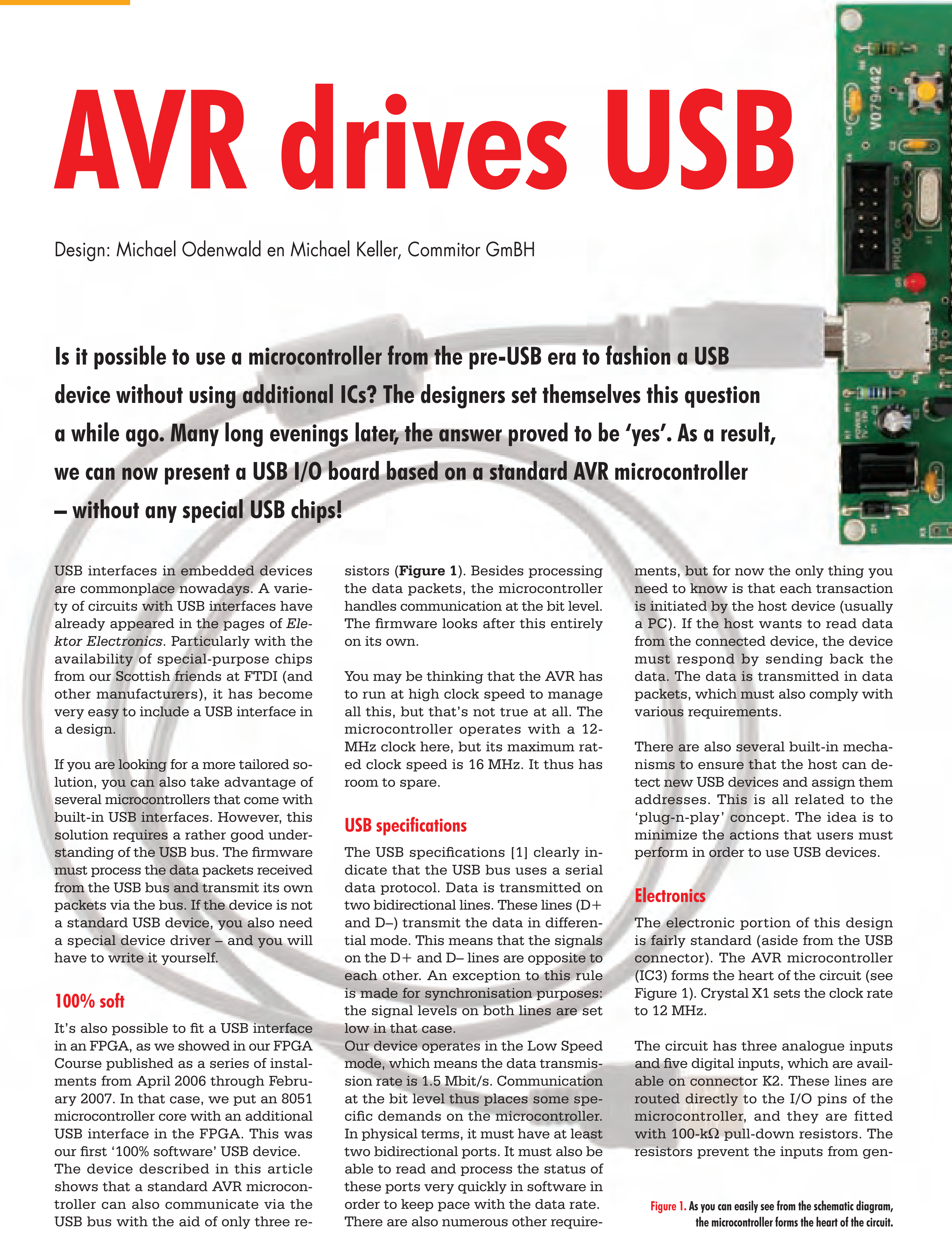


AVR drives USB

Design: Michael Odenwald en Michael Keller, Commitor GmBH

Is it possible to use a microcontroller from the pre-USB era to fashion a USB device without using additional ICs? The designers set themselves this question a while ago. Many long evenings later, the answer proved to be 'yes'. As a result, we can now present a USB I/O board based on a standard AVR microcontroller – without any special USB chips!

USB interfaces in embedded devices are commonplace nowadays. A variety of circuits with USB interfaces have already appeared in the pages of *Elektor Electronics*. Particularly with the availability of special-purpose chips from our Scottish friends at FTDI (and other manufacturers), it has become very easy to include a USB interface in a design.

If you are looking for a more tailored solution, you can also take advantage of several microcontrollers that come with built-in USB interfaces. However, this solution requires a rather good understanding of the USB bus. The firmware must process the data packets received from the USB bus and transmit its own packets via the bus. If the device is not a standard USB device, you also need a special device driver – and you will have to write it yourself.

100% soft

It's also possible to fit a USB interface in an FPGA, as we showed in our FPGA Course published as a series of instalments from April 2006 through February 2007. In that case, we put an 8051 microcontroller core with an additional USB interface in the FPGA. This was our first '100% software' USB device.
The device described in this article shows that a standard AVR microcontroller can also communicate via the USB bus with the aid of only three resistors (**Figure 1**). Besides processing the data packets, the microcontroller handles communication at the bit level. The firmware looks after this entirely on its own.

You may be thinking that the AVR has to run at high clock speed to manage all this, but that's not true at all. The microcontroller operates with a 12-MHz clock here, but its maximum rated clock speed is 16 MHz. It thus has room to spare.

USB specifications

The USB specifications [1] clearly indicate that the USB bus uses a serial data protocol. Data is transmitted on two bidirectional lines. These lines (D+ and D–) transmit the data in differential mode. This means that the signals on the D+ and D– lines are opposite to each other. An exception to this rule is made for synchronisation purposes: the signal levels on both lines are set low in that case.
Our device operates in the Low Speed mode, which means the data transmission rate is 1.5 Mbit/s. Communication at the bit level thus places some specific demands on the microcontroller. In physical terms, it must have at least two bidirectional ports. It must also be able to read and process the status of these ports very quickly in software in order to keep pace with the data rate. There are also numerous other requirements, but for now the only thing you need to know is that each transaction is initiated by the host device (usually a PC). If the host wants to read data from the connected device, the device must respond by sending back the data. The data is transmitted in data packets, which must also comply with various requirements.

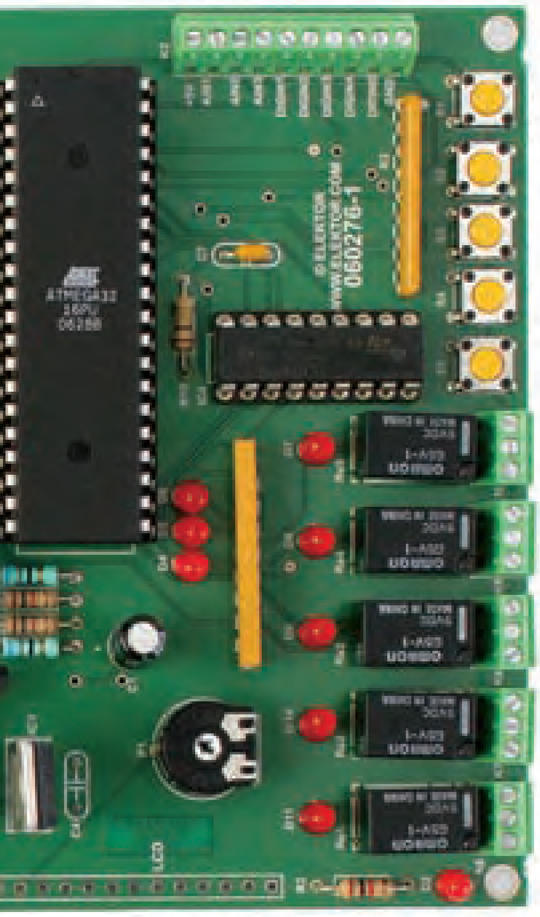
There are also several built-in mechanisms to ensure that the host can detect new USB devices and assign them addresses. This is all related to the 'plug-n-play' concept. The idea is to minimize the actions that users must perform in order to use USB devices.

Electronics

The electronic portion of this design is fairly standard (aside from the USB connector). The AVR microcontroller (IC3) forms the heart of the circuit (see Figure 1). Crystal X1 sets the clock rate to 12 MHz.

The circuit has three analogue inputs and five digital inputs, which are available on connector K2. These lines are routed directly to the I/O pins of the microcontroller, and they are fitted with 100-kΩ pull-down resistors. The resistors prevent the inputs from gen-

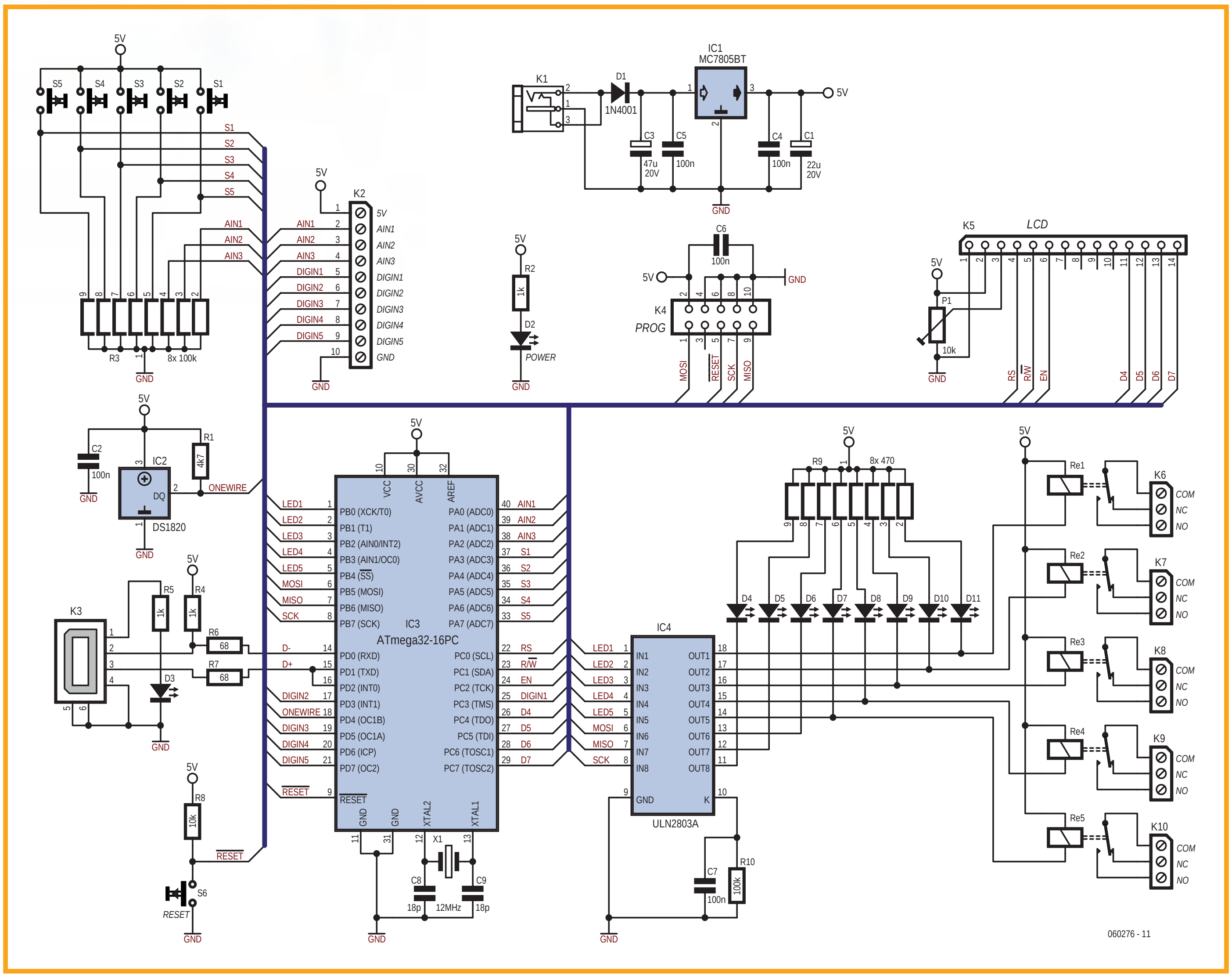
Figure 1. As you can easily see from the schematic diagram, the microcontroller forms the heart of the circuit.

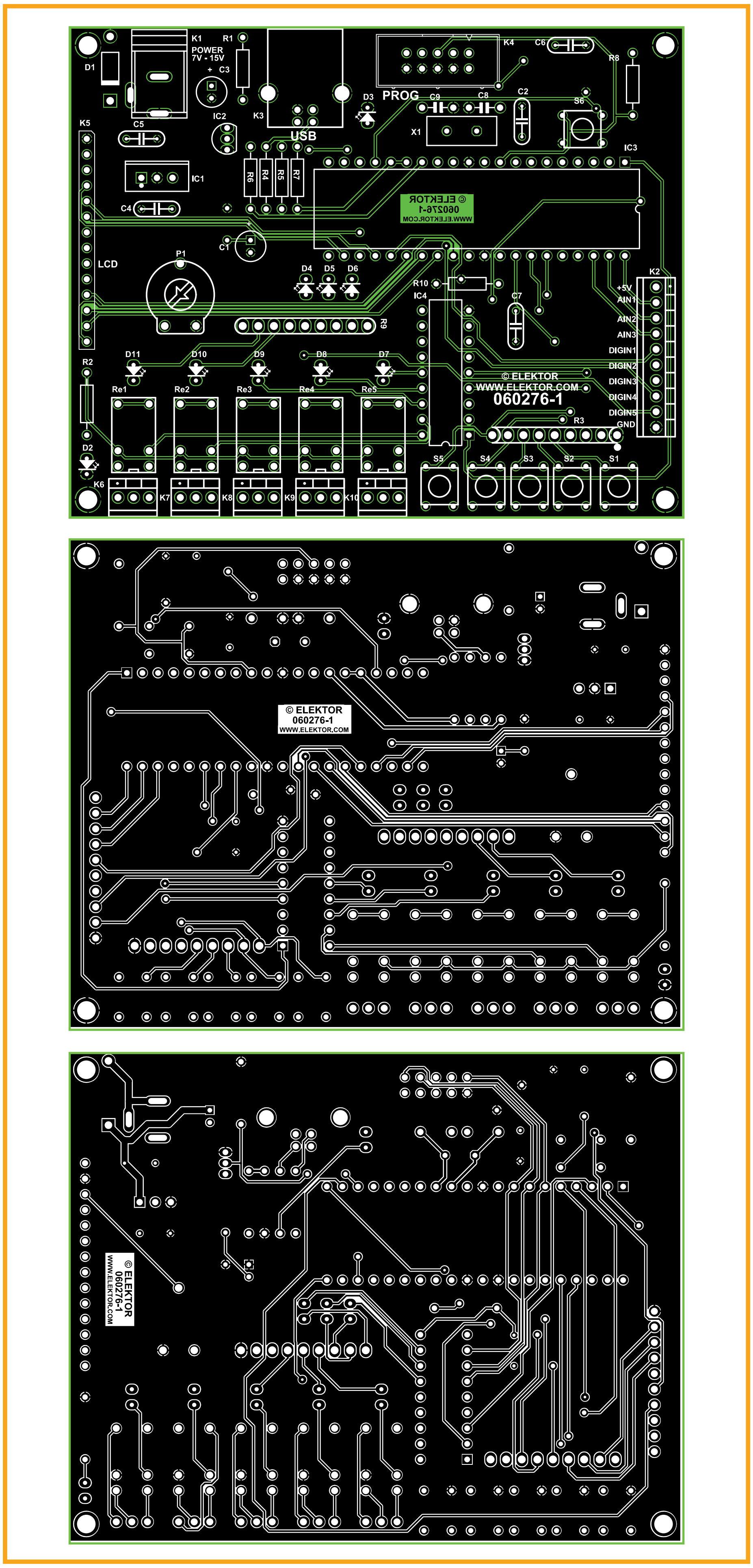


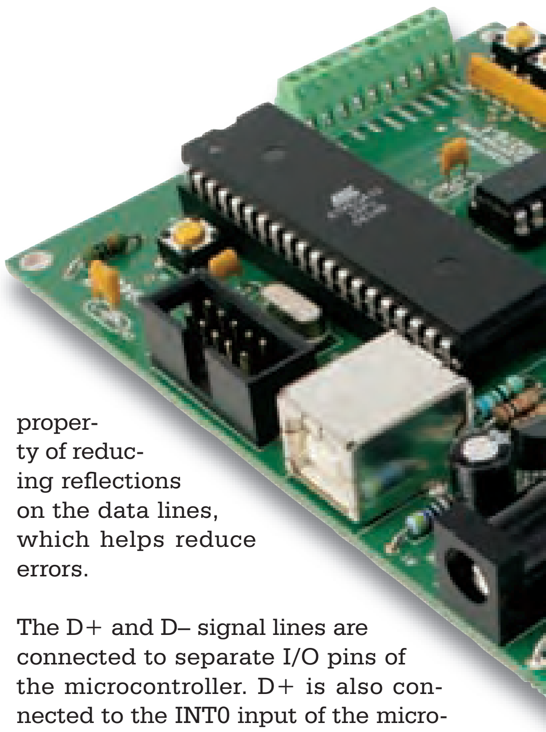
A development board with a software-defined USB interface

erating annoying problems due to static charges if they left open.
The circuit also has a temperature sensor (IC2). This is a '1-wire' chip, which means that only one I/O pin is necessary to use the sensor. The input options are rounded out with five push-button switches.

The circuit is fitted with an LCD that is driven using four data bits and the usual control signals. The microcontroller can also drive five relays. There an LED for each relay to indicate whether the relay is actuated.
The USB port is extremely simply with regard to the electronic components. Resistor R4 causes the host to recognise that a Low-Speed device is connected to the USB port. Resistors R6 and R7 provide current limiting in case of problems. They also have the pleasant







property of reducing reflections on the data lines, which helps reduce errors.

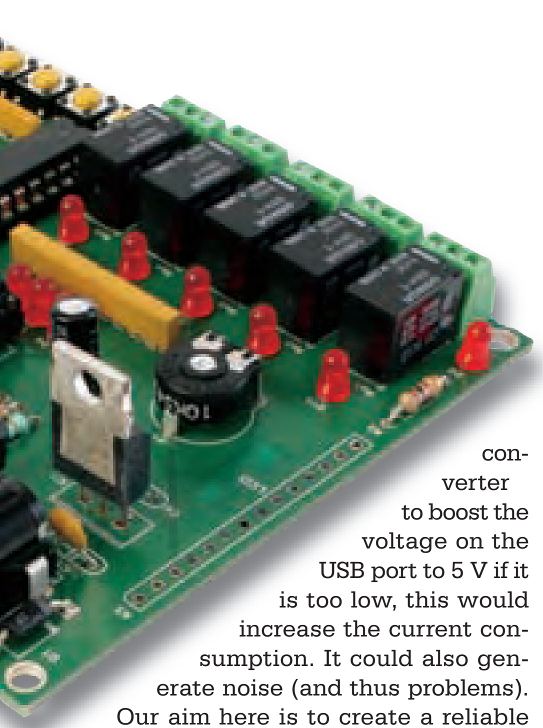
The D+ and D– signal lines are connected to separate I/O pins of the microcontroller. D+ is also connected to the INT0 input of the microcontroller. This makes it possible to generate an interrupt each time the signal level on the D+ line changes. Don't be misled by the fact that the D+ and D– lines are connected to the UART pins of the microcontroller. The software uses these two pins as normal I/O pins. The built-in UART of the microcontroller is not used for the USB interface.
The PCB track and component layouts are shown in **Figure 2**.

Power supply

An AC mains adapter is suitable for powering the entire circuit. In principle, it is also possible to power USB devices from the host, but this option is not suitable here. According to the specifications, the voltage on the USB bus can range from 4.4 V to 5.25 V. However, voltages seen in practice often differ from the specifications. With regard to current consumption, the maximum current a device is allowed to draw before enumeration (see inset) is 100 mA. During the enumeration process, the device states how much current it wishes to draw from the USB port (up to a maximum of 500 mA).

In this case, we need a minimum voltage of 5 V for the microcontroller. Although it would be possible to use a step-up

**Figure 2. The microcontroller also occupies the central position on the circuit board.
As you can see, there is a wealth of I/O options.**



converter to boost the voltage on the USB port to 5 V if it is too low, this would increase the current consumption. It could also generate noise (and thus problems). Our aim here is to create a reliable DIY design that is also reliable in operation, so we chose the simple solution of a standard power supply using an AC mains adapter. Diode D1 protects the voltage regulator (IC1) against reverse-polarity connection of the mains adapter.

Firmware

Naturally, the driving force of this project is the firmware. The firmware consists of several modules, which are predominantly written in C. Assembly-language code is only used for driving the USB lines, since it is faster.

The device descriptor is located in the usb.h file. During the enumeration process, the host uses the data in the device descriptor to determine what sort of device is connected. If you want to adapt the circuit to your own purposes (or just play with it), this is where you can ensure that the host recognises your device correctly. Of course, this requires a certain amount of knowledge of the USB protocol.

The avr-usb.h file contains a list of requests that are supported by the connected device. The host can send a request to the device, which performs the associated action in response. Some examples of typical actions are clearing the LCD, actuating or releasing a relay, and so on. In some cases, the request requires the device to return data to the host. One example of this is reading the temperature.

COMPONENTS LIST

Resistors

R1 = 4kΩ7
R2,R4,R5 = 1kΩ
R3 = SIL array 8x 100kΩ
R6,R7 = 68Ω
R8 = 10kΩ
R9 = SIL array 8x 470Ω
R10 = 100Ωk
P1 = 10kΩ preset

Capacitors

C1 = 22µF 20V radial
C2,C4,C5,C6,C7 = 100nF
C3 = 47µF 20V radial
C8,C9 = 18pF

Semiconductors

D1 = 1N4001
D2-D11 = low-current LED, red, lead pitch 2.5mm
IC1 = 7805CP
IC2 = DS1820
IC3 = ATmega32-16PC (programmed, E-SHOP # **060276-41**)
IC4 = ULN2003A

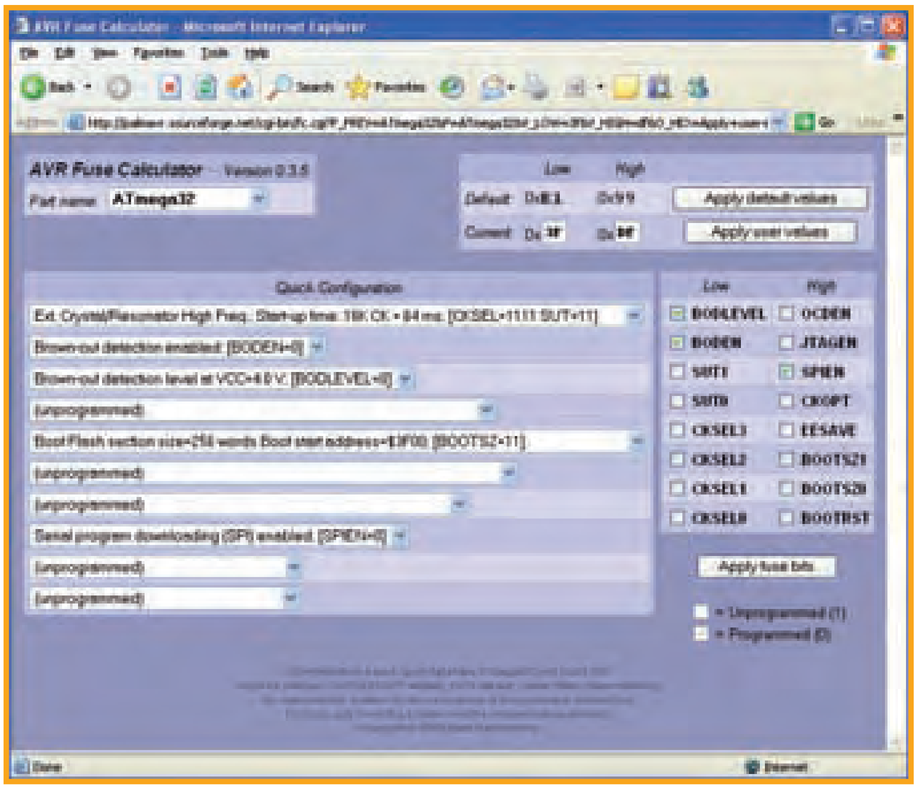
Miscellaneous

K1= 2.5mm mains adaptopr socket
K2= 10-way PCB terminal block, lead pitch 2.54 mm (e.g. Phoenix contact # 1725737)
K3= USB-B connector
K4 = 10-way boxheader
K5= 14-way pinheader
K6-K10= 3-way PCB terminal block, lead pitch 2.54mm (e.g. Phoenix contact # 1725669)
Re1-Re5 = 5V relay (e.g. OMRON G5V-1-DC5)
S1-S6 = pushbutton (e.g. OMRON B3F-1002)
X1 = 12MHz quartz crystal, HC49/U case
LCD module, 2x16 characters
PCB, E-SHOP # **060276-1**

Compiling

The complete firmware, including the assembly-language code, can be compiled using the AVR GCC compiler [2], which is a (free) open-source compiler. There is also a downloadable 'make' file, which makes the compilation process a lot easier (see [3]).

The circuit has a programming interface, so you can program new firmware into the microcontroller while it is fitted to the board ('in system'). However, you must set the proper fuse bits for this, since otherwise the entire process won't work. The palmaver site [4] describes a convenient way to determine the proper configuration bytes. You can determine the right settings for this circuit by entering '0x3fDf' in the box at the upper right (see **Figure 3**). This corresponds to programming the BOD-



AVR Fuse Calculator – Version 0.3.5
Part name: ATmega32

	Low	High
Default	0xE1	0x99
Current	0x 3F	0x DF

Quick Configuration
Ext. Crystal/Resonator High Freq.; Start-up time: 16K CK + 64 ms; [CKSEL=1111 SUT=11]
Brown-out detection enabled; [BODEN=0]
Brown-out detection level at VCC=4.0 V; [BODLEVEL=0]
(unprogrammed)
Boot Flash section size=256 words Boot start address=$3F00; [BOOTSZ=11]
(unprogrammed)
(unprogrammed)
Serial program downloading (SPI) enabled; [SPIEN=0]
(unprogrammed)
(unprogrammed)

Low	High
✓ BODLEVEL	☐ OCDEN
✓ BODEN	☐ JTAGEN
☐ SUT1	✓ SPIEN
☐ SUT0	☐ CKOPT
☐ CKSEL3	☐ EESAVE
☐ CKSEL2	☐ BOOTSZ1
☐ CKSEL1	☐ BOOTSZ0
☐ CKSEL0	☐ BOOTRST

Figure 3. It's easy to determine the right fuse settings with this handy online tool [4].

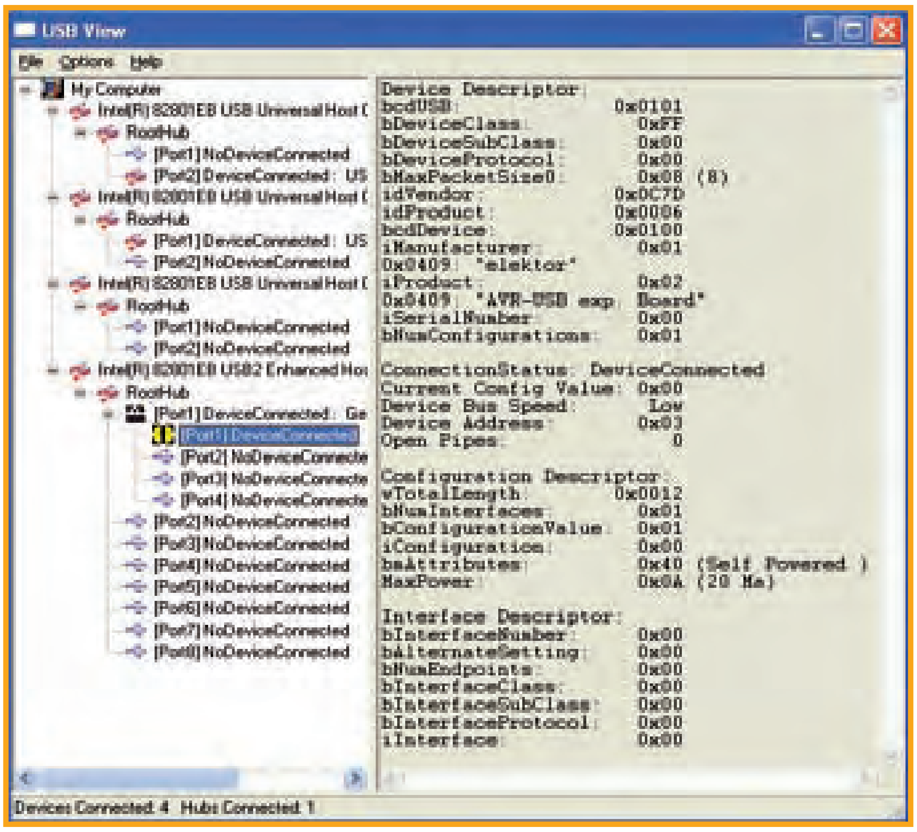
Enumeration

The host device must perform a process called 'enumeration' before it can use a connected USB device. The first step in the process takes place in he USB device, where a pull-up resistor in the device signals its presence on the USB bus.

In the case of a Low-Speed device (1.5 Mbit/s), the pull-up resistor must pull the D– line to +3.3 V. In the case of Full-Speed (12 Mbit/s) and High-Speed (480 MB/s) devices, the D+ line must be pulled to +3.3 V.

In response to the change in the signal level on the data line, the host uses a predefined protocol to try to determine what sort of device is connected to the USB port. Besides the expected data (such as the VID and PID), the device must also report which class it belongs to, along with other information such as its version number, name, and so on. A USB address is also assigned to the device. The host uses the addresses to distinguish the different USB devices.

The information described above enables the host to determine which device driver it needs in order to use the device.


USB View
File Options Help
My Computer
Intel(R) 82801EB USB Universal Host (
RootHub
[Port1] NoDeviceConnected
[Port2] DeviceConnected : US
Intel(R) 82801EB USB Universal Host (
RootHub
[Port1] DeviceConnected : US
[Port2] NoDeviceConnected
Intel(R) 82801EB USB Universal Host (
RootHub
[Port1] NoDeviceConnected
[Port2] NoDeviceConnected
Intel(R) 82801EB USB2 Enhanced Hos
RootHub
[Port1] DeviceConnected : Ge
[Port1] DeviceConnected
[Port2] NoDeviceConnecte
[Port3] NoDeviceConnecte
[Port4] NoDeviceConnecte
[Port2] NoDeviceConnected
[Port3] NoDeviceConnected
[Port4] NoDeviceConnected
[Port5] NoDeviceConnected
[Port6] NoDeviceConnected
[Port7] NoDeviceConnected
[Port8] NoDeviceConnected
Devices Connected: 4 Hubs Connected: 1

```
Device Descriptor:
bcdUSB:                 0x0101
bDeviceClass:             0xFF
bDeviceSubClass:          0x00
bDeviceProtocol:          0x00
bMaxPacketSize0:          0x08 (8)
idVendor:               0x0C7D
idProduct:              0x0006
bcdDevice:              0x0100
iManufacturer:            0x01
0x0409: "elektor"
iProduct:                 0x02
0x0409: "AVR-USB exp. Board"
iSerialNumber:            0x00
bNumConfigurations:       0x01

ConnectionStatus: DeviceConnected
Current Config Value:     0x00
Device Bus Speed:          Low
Device Address:           0x03
Open Pipes:                  0

Configuration Descriptor:
wTotalLength:           0x0012
bNumInterfaces:           0x01
bConfigurationValue:      0x01
iConfiguration:           0x00
bmAttributes:             0x40 (Self Powered )
MaxPower:                 0x0A (20 Ma)

Interface Descriptor:
bInterfaceNumber:         0x00
bAlternateSetting:        0x00
bNumEndpoints:            0x00
bInterfaceClass:          0x00
bInterfaceSubClass:       0x00
bInterfaceProtocol:       0x00
iInterface:               0x00
```

Figure 4. USBview gives you a bird's-eye view of everything connected to the USB.

LEVEL, BODEN and SPIEN bits (setting them to logic 0) and leaving the other bits unprogrammed (logic 1).

Assembly and testing

The circuit should be easy to assemble if you use the accompanying PCB design. The circuit does not include any difficult SMD components, and everything is readily available. The PCB is also available from our E-SHOP [3].

After soldering all the components in place, inspect the results carefully to ensure that all the solder joints are good and you haven't created any shorts. It's also a good idea to double-check the values of the resistors and capacitors and verify that all the ICs are oriented correctly in their sockets.

Once you have completed the inspection, you can start testing the board – cautiously of course! First connect an AC mains adaptor with a DC output voltage of 9–12 V. If everything is as it should be, LED D2 will light up. Next, connect the circuit to a PC via a USB cable. LED D3 should also light up now. Windows (assuming you are running Windows) will then start the enumeration process.

You can skip installation of the driver for now, since you don't yet have a specific driver for this circuit. The driver (which is also open-source software) is described in the 'Universal USB Driver' article in this issue.

USBview

You don't necessarily need a driver to test USB communications. Instead, you can use Microsoft's USBview utility. This program is included in the Microsoft Driver Development Kit (DDK) [5]. You can use it to view the data from the device descriptor of the USB device (see **Figure 4**).

This program works without a device driver. You can test USB communications by viewing the data from your device in the USBview window. All the data you see there comes from the device and is sent to the PC via the USB bus.

If all the test results are positive, the hardware is ready and you can start working on the device driver. Refer to the 'Universal USB Driver' article elsewhere in this issue for the details.

(060276-1)

Web links

[1] www.usb.org/developers/docs.html

[2] winavr.sourceforge.net/

[3] www.elektor-electronics.co.uk

[4] palmavr.sourceforge.net/cgi-bin/fc.cgi

[5] www.microsoft.com/whdc/devtools/ddk/default.mspx