ATmega on the Internet (1)

Using Raspberry Pi as a network gateway

By **Dieter Holzhäuser**
(Germany)

Communicating with a microcontroller over the Internet is easy. All you need is a networked PC or smartphone, a local area network (LAN), and another computer connected to the ATmega32 over a serial link. The basis for this series of articles is a system in which a Fritz!Box router provides the LAN, and a Raspberry Pi is used as a network gateway. In the first instalment we present the basic concept and describe some typical hardware.

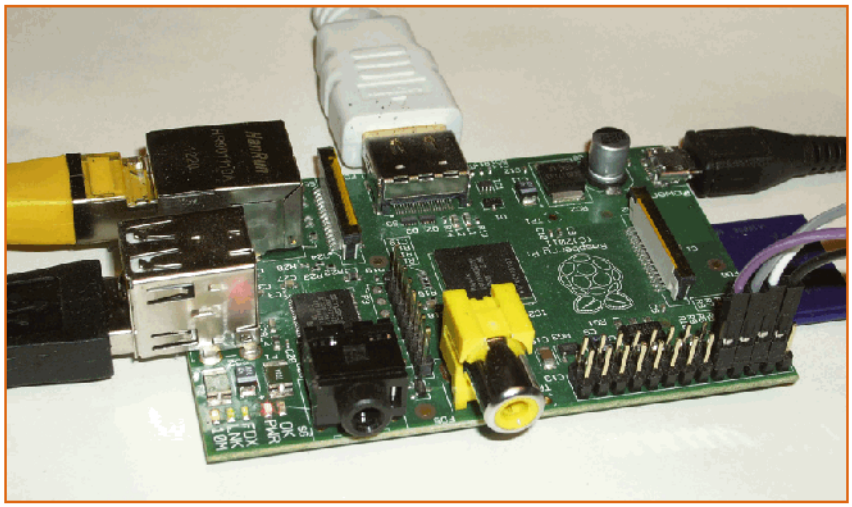

Figure 1.
A Raspberry Pi in practical use.

The Raspberry Pi, which is roughly the size of a credit card, is a simple Linux PC. The "Model B'' version, with 512 MB of RAM and an Ethernet port (see **Figure 1**), costs about 40 dollars. A monitor can be connected to its HDMI output, and its two USB 2.0 ports are used for a keyboard and a mouse. Power is supplied to the Raspberry Pi through a Micro USB connector. With a power consumption of less than 3 watts, it's no power glutton, so it can be left on all the time. After all, there wouldn't be much point in using a computer that is only occasionally powered up as an access point for the Internet.

OS aspects

The current OS, Raspbian wheezy, can be downloaded free of charge from the Internet as a disk image [2]. However, it can also be obtained as a pre-installed version on an 8-GB MicroSD card with SD adapter, with about 6 GB unused.
It's really amazing that a computer this small can support a graphical user interface. However, in this project we use the command line interface instead. The software behind this is called a shell. You can access the command line interface of the shell by launching *LXTerminal* or by exiting the graphical user interface. If you don't want to use the graphical user interface at all, you can alter the configuration settings with the *raspi-config* utility to prevent the GUI from autostarting after a system boot. To run this utility using the command line interface, enter:

```
sudo raspi-config
```

The prefix *sudo* allows regular users to use system calls normally reserved for the root user (Superuser).
The procedure is largely self-explanatory. In the line:

```
boot_behaviour      Start desktop on boot?
```

use the Tab key to set the focus on `<Select>`, press Enter, and then give the appropriate answer to the subsequent question:

```
Should we boot straight to desktop?
<Yes>      <No>
```

If you say `<Yes>`, the graphical user interface will appear every time after booting (without login); if you say `<No>`, the command line interface will appear (with login). If for some reason you want to use the graphical user interface later on, you

can launch it manually with *startx*. Communication between the shell and the user's terminal device, which is also called the console, consists of sending characters or character strings back and forth. It doesn't matter where the terminal device is located or how it is connected to the computer.

The abbreviation commonly used in Linux for a character-oriented terminal device is *tty*, which is short for "teletypewriter"). In Linux this means not only the terminal itself, but also the computer port to which the terminal can be connected.

Serial interface

The local terminal, consisting of the keyboard and monitor connected to the Raspberry Pi, does not need a physically accessible interface. However, the Raspberry Pi does provide a simple serial interface. It is brought out to the two-row pin header and configured such that after booting, a user can log in to a shell using a terminal connected to the serial port.

However, this is not what we want. Instead, we want to use this interface to allow an ATmega32 to communicate with a terminal. Two things are necessary for this. The first is to eliminate the configuration of the serial interface as the console. The second is to convert the Raspberry Pi into a terminal that uses this interface. This means that all keyboard inputs to the Raspberry Pi are passed on directly over the interface and all received characters are output to the monitor. To get rid of the existing configuration of the serial interface, you have to edit two text files. Open the first file by calling the command line editor *nano*:

```
sudo nano /etc/inittab
```

At the end of the displayed text there is a line with the following (or similar) content:

```
T0:23:respawn:/sbin/getty -L ttyAMA0
115200 vt100
```

Place a hash mark (#) at the start of this line to change it into a comment. After this change, the serial interface (the internal device *ttyAMA0*) will ignore console input and output. Save the edited file and exit the editor.
Even with this change, the boot-up data is still

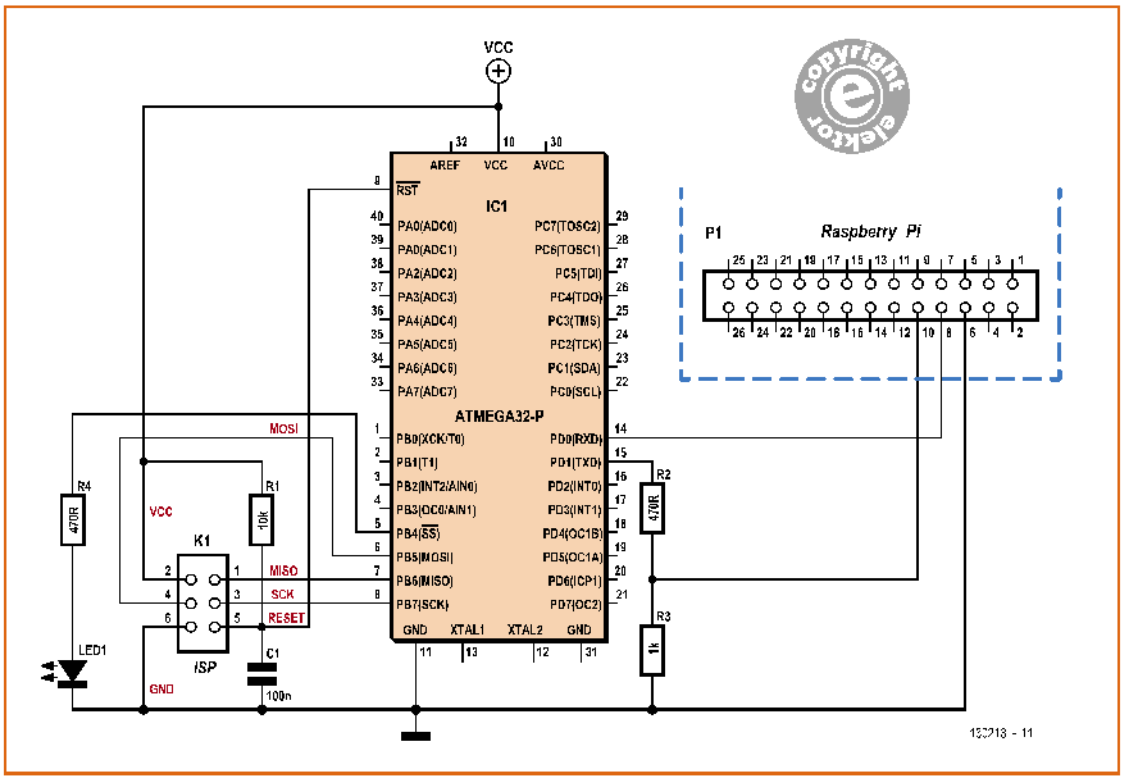

Figure 2.
The serial link between the ATmega32 and the Raspberry Pi.

Listing 1: suidemo1.c

```
1  #include <avr/interrupt.h>
2  #include <avr/io.h>
3
4  #define BAUDCODE      12                      // 4.800 Baud @ 1 MHZ
5  #define INITLED     DDRB = DDRB | 1<<4;
6  #define TOGGLELED   PORTB = PORTB ^ 1<<4;
7  #define ISLEDON     PORTB & 1<<4
8
9  unsigned int cycle = 500;
10 unsigned long clock = 0;
11 unsigned long old = 0;
12 unsigned char chr;
13 unsigned char ontxt []  = "\r1|5  ***  ";
14 unsigned char offtxt [] = "\r1|5       ";
15 unsigned char * ptxt;
16
17 ISR ( TIMER0_COMP_vect ){                     //Interrupt Service Routine Timer
18   if ( !(clock++ % cycle)  ) {
19     TOGGLELED
20     if (ISLEDON) ptxt = ontxt; else ptxt = offtxt;
21     UDR = *ptxt ;
22   }
23 }
24
25 ISR ( USART_TXC_vect ) {                      //Interrupt Service Routine Transmit
26     ptxt++;
27     if (*ptxt != 0)  UDR = *ptxt;
28 }
29
30 ISR ( USART_RXC_vect ) {                      //Interrupt Service Routine Receive
31   chr = UDR;
32   if (clock - old  > 50) {
33     old = clock;
34     if (chr == '5') cycle = 500;
35     else if (chr == '1') cycle = 100;
36   }
37 }
38
39 int main(void) {
40   TCCR0 = 0x08;                    // Init Timer  CTC-Mode
41   TCCR0 = TCCR0 | 0x02;            //Prescaler: 1 MHz: 8 = 125 kHz, 0x02; 16 MHz: 64 = 250 kHz, 0x03
42   OCR0 = 124;                      //Compare  1 MHz: 124; 16 MHz: 249
43   TCNT0 = 0;                       //TimerCounter on 0
44   TIMSK |= 1<<OCIE0;               //enable CTC-Interrupt Timer0
45   UBRRH = (unsigned char) (BAUDCODE >> 8);    //Init USART
46   UBRRL = (unsigned char) BAUDCODE;
47   UCSRB = UCSRB | (1<<RXEN) | (1<<RXCIE) | (1<<TXEN) | (1<<TXCIE)  ;
48   sei();                                      //Enable Interrupts
49   INITLED
50   while (1) ;
51   return 0;
52 }
```

sent to the serial interface, To prevent this, call the editor again with the second file:

```
sudo nano /boot/cmdline.txt
```

In the displayed text

```
dwc_otg.lpm_enable=0 console=ttyAMA0,115200
kgdboc=ttyAMA0,115200 console=tty1 root=/dev/
mmcblk0p2 rootfstype=ext4 elevator=deadline
rootwait
```

delete the two references to *ttyAMA0*. The modified text is:

```
dwc_otg.lpm_enable=0 console=tty1 root=/dev/
mmcblk0p2 rootfstype=ext4 elevator=deadline
rootwait
```

Save the edited file and then restart.

The Raspberry Pi needs a terminal emulator program in order to act as a terminal. One option for this is the program *picocom*. First you have to download and install the program. The command for installing the program is:

```
sudo apt-get install picocom
```

The installation process is automatic. You don't have to do anything else. To launch the program, type:

```
picocom -b 4800 /dev/ttyAMA0
```

Here *-b 4800* sets the baud rate and */dev/ttyAMA0* is the device file for the serial interface. You don't have to type this command again every time you need it. Instead, you can use the Up and Down arrow keys to scroll through previously entered commands and retrieve them to the command line.
For your first test, connect pin 8 (TxD) and pin 10 (RxD) together on pin header P1 (see **Figure 2**). Now the characters you type on the keyboard will appear on the monitor. To exit the terminal emulator program *picocom*, press the key combination Ctrl-A-X.

ATmega32

In order for two devices to communicate over a serial interface, they must have the same transmission parameter settings. The only configurable parameter for the simple serial interface of the Raspberry Pi is the baud rate. The frame format and the mode bits are permanently set to eight data bits, no parity, one stop bit and no handshaking. The *picocom* program and the serial interface of the ATmega32 are preconfigured with these settings. The only parameter you need to set in software is the baud rate, and pins 14 (RxD) and 15 (TxD) of the ATmega32 must be configured for the serial interface.

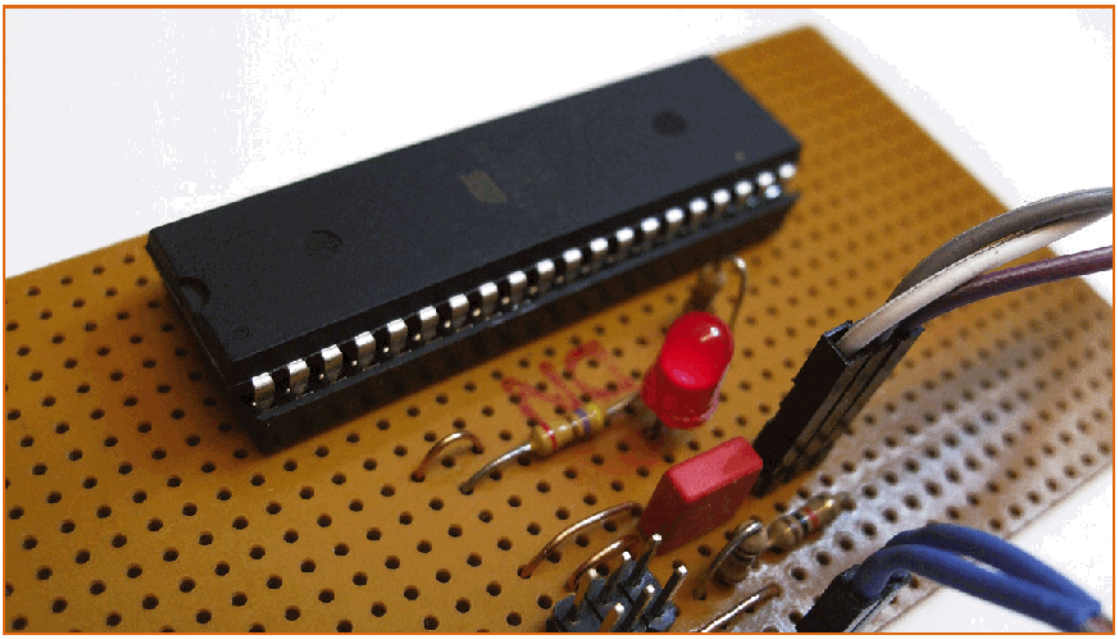

Figure 3.
The ATmega32 on a piece of prototyping board.

For experimenting, it's helpful to mount the ATmega32 on a piece of prototyping board (see **Figure 3**). It's ready to run after an external circuit or signal is connected to the reset input. The device is factory-configured to operate from its internal clock source, which runs at 1 MHz.
A programming device, such as the AVRISP mkII, can be connected to the 6-pin pin header K1. Information about setting up a development environment can be found on the Internet [3]. Connect R1 and C1 to the Reset input.

Connect the TX and RX pins of the microcontroller to the corresponding pins of the 26-pin pin header P1 on the Raspberry Pi board. The voltage divider formed by R2 and R3 ensures compliance with the 3.3-V signal level requirement of the Raspberry Pi board. Check carefully to ensure that you have made the right connections, since the Raspberry Pi can be permanently damaged by incorrect connections.
Connect an LED to pin 5 (PB4) of the ATmega32 to allow program activity to be observed (e.g. the activity of the simple C program described below).

Demo program *suidemo1.c*

The *suidemo1.c* program shown in **Listing 1** is intended to be used as a demo and for experimenting. It consists of the *main* function, which contains the main loop, and three interrupt service routines (ISRs).
The program causes the LED to blink at two different rates. A simple user interface is implemented in the form of characters that are output and displayed on the Raspberry Pi monitor. They show the state of the LED. You can change the blink rate by pressing a key on the keyboard.
In lines 40 to 44*Timer0* is initialized to cause the ISR*TIMER0_COMP_vect* to be called at a frequency of 1 kHz. The ISR increments the variable *clock*, which acts as an internal clock. In lines 18 and 19, the variable *cycle* is used to derive the LED blink rate from the value of *clock*. On each state change the pointer *ptxt* is set to the text

Web Links

[1] Author's website: www.system-maker.de (in German)
[2] Raspian wheezy download: www.raspberrypi.org/downloads
[3] Information about IDEs and programmers: www.system-maker.de/avr.html (in German)
[4] ATmega32 data sheets: www.atmel.com/devices/atmega32.aspx
[5] Elektor web page for this project: www.elektor-magazine.com/130213

string to be transmitted over the serial interface (line 20) in order to display the new state of the LED on the monitor. Transmission is initiated by writing the first character of the text string to the register *UDR* (line 21).

The ISR *USART_TXC_vect* is responsible for sending the rest of the characters. It is called each time the transmit shift register becomes empty. That is why this ISR can only transmit the second and subsequent characters. The ISR stops transmitting when the null character marking the end of the string is detected (line 27).

The ISR *USART_RXC_vect* is called each time a character from the terminal keyboard has been received completely. First the register *UDR* must be read out (line 31). Keystrokes that send more than one character can be recognized from the arrival rate of the characters. However, character sequences of this sort are not used in this program. For this reason, if the time since the last ISR call when a new character arrives is distinctly less than the usual time between keystrokes, the character is discarded (line 32). Only keys 1 and 5 are significant. They change the LED blink rate by altering the value of the variable *cycle* in lines 34 and 35. The serial interface must be initialized before it can be used; this is done in lines 46 to 48. The baud rate is set by writing a number that depends on the clock frequency (see the ATmega32 data sheet [4]) to the registers *UBRRH* and *UBRRL*. The maximum supported baud rate with a clock frequency of 1 MHz is 4,800. The interrupt enable bits for the transmit and receive functions must also be set. The bits of the Status and Control register *UCSRC* are initialized automatically after a reset to correspond to the frame format.

The endless loop in line 50 keeps the program active. The program is controlled exclusively by the *Timer0* events and the terminal keyboard.

The serial user interface only responds to keys 1 and 5. It also shows the current LED state in a single line on the terminal monitor. When the LED is lit, the line reads:

```
1|5    ***
```

Now we have effectively implemented a remote control mechanism that allows the ATmega32 to be controlled over a serial interface.

In the next instalment of this mini-series, we show you how to use this combination of a Raspberry Pi computer and an ATmega32 in a local area network and how it can be controlled over the Internet from anywhere in the world.

(130215-I)