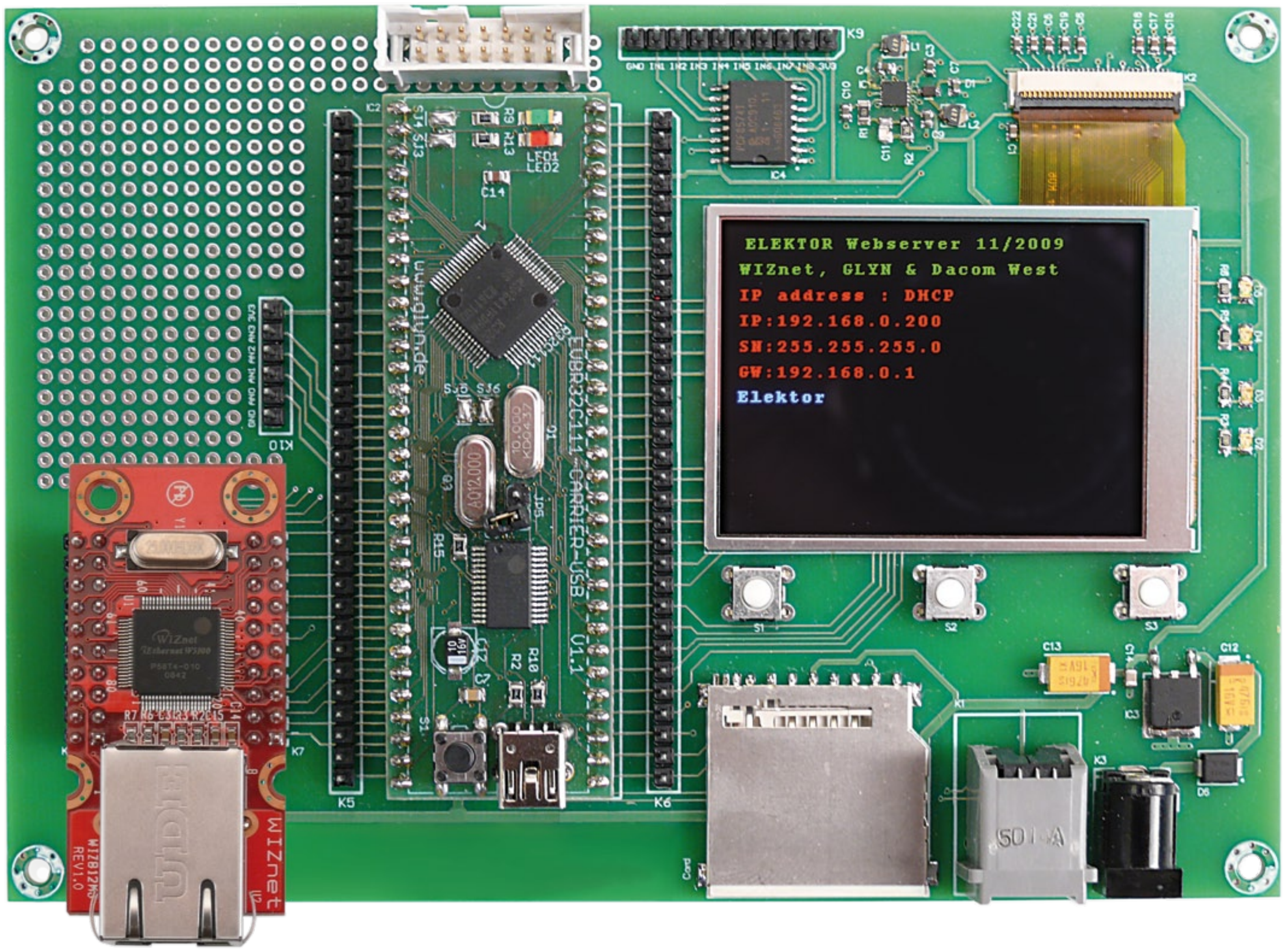
R32C Web Server
Networking module for HTTP,



e-Mail, FTP and much more

by Jinbuhm Kim (South Korea) and Joachim Wülbeck (Germany)

The R32C microcontroller goes Internet! A small add-on module for the application board from our September 2009 issue combines a TCP/IP chip plus Ethernet interface, a network connection with built-in transformer and status LEDs. This handy combination makes it child's play to implement a web server and many other Internet applications without getting involved in complexities such as TCP/IP protocol. Free downloads of an Open Source driver, a short web server program and other sample software complete this attractive proposition.

The South Korean electronics company WIZnet [1] has its origins at the University of Seoul eleven years ago. Within a short while it had produced the W3100, the first hardwired TCP/IP chip on the market with a 10/100 MBit/s Ethernet interface. The Internet protocol stack in this was created in hardware, so as to relieve the MCU of complicated TCP/IP protocol processing.

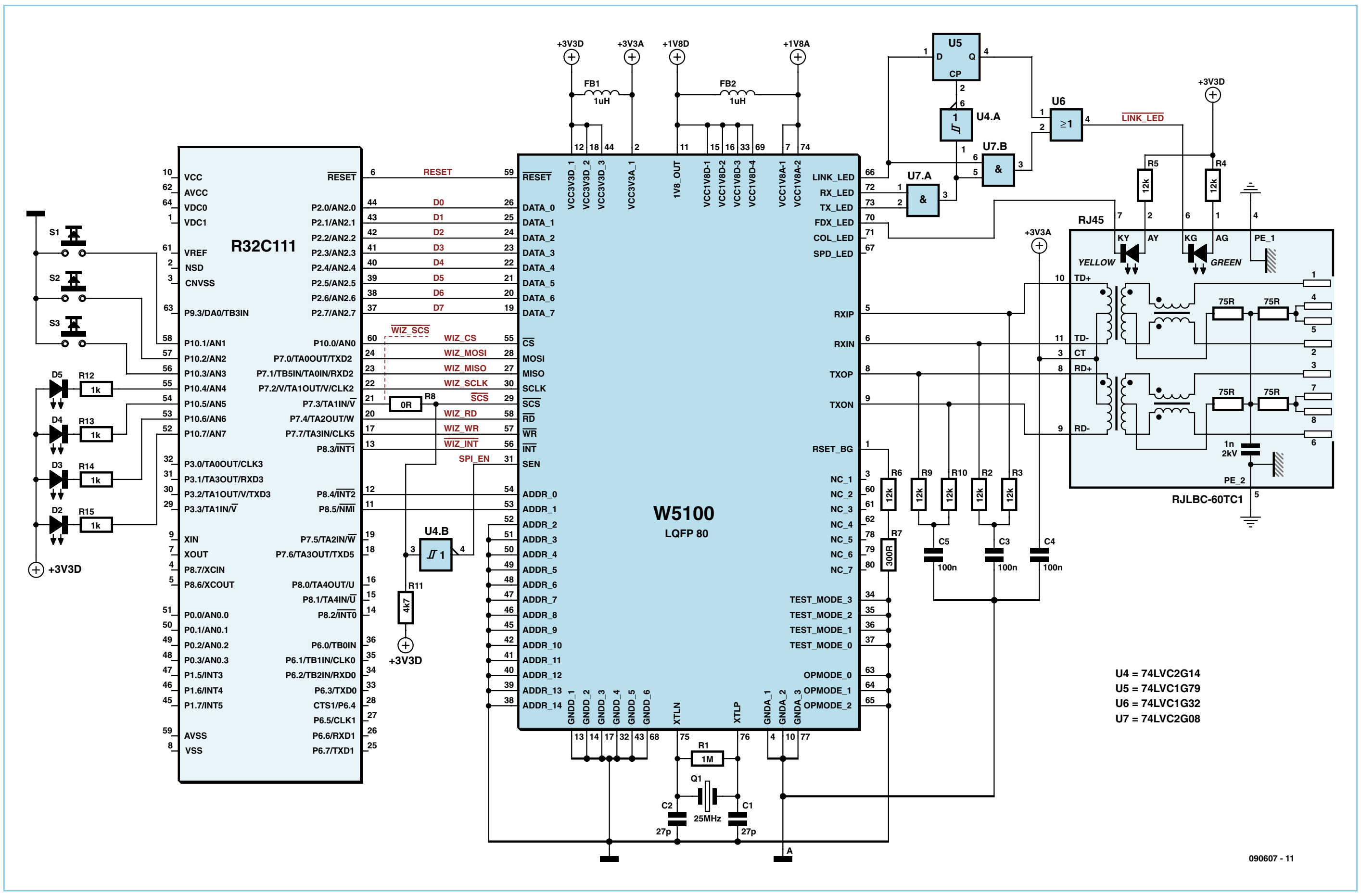


Figure 1. Schematic of the network module and R32C connection (for clarity the application board section shows only the LEDs and press switches, whilst the 3.3 V regulator and some passive components in the network module are not shown either).

This enabled Internet applications such as web server, e-mail, FTP and plenty more to be implemented even on small 8-bit microcontrollers without the need for an operating system. Users need not worry about programming the PHY, MAC, IP and TCP layers but can instead rely simply on the built-in driver functions (see panel for explanation of these abbreviations and [8] for how these layers work).

Family connections

After the W3100 the company developed several more chips, with varying levels of integration. The W5100 that we are using here is a 3-in-1 chip that combines the TCP/IP, MAC and PHY functions in hardware and is controlled either by SPI bus or over an 8-bit parallel interface. Network modules in DIL form, comprising the TCP/IP chip and an RJ-45 connector with transformer, simplify the hook-up further. In this project we employ the WIZ812MJ module, which plugs very conveniently into our R32C application board.

WIZnet's product range also offers higher performance Internet chips, handling data rates up to 80 Mbit/s. These are for demanding applications such as HD video or image recognition. In the near future the 4-in-1 chip W7100 will further enhance the range; as well as the protocol functions already mentioned this also includes an 8051 core, 64 kB of RAM and 64 kB + 128 kB of flash memory.

The hardware

If the WIZ812MJ module is placed in the socket already provided on the application board [2] the R32C/111 can communicate with the W5100 using SPI or an 8-bit parallel connection. In the latter case we must use the so-called indirect mode. In contrast to direct mode the full address bus is not used, which saves a whole load of signal lines. Despite this short cut the indirect mode is just as fast; to read or write a block of data, only the start address needs to be sent (over the data bus). The W5100 now counts up the address independently in auto-increment mode. On the address bus only the two lines A0 and A1 are used. These define whether we have a data byte or the high/low byte of the start address on the data bus.

The schematic in **Figure 1** shows a section of the application board, the connections for the W5100 and the hook-up with the network module. The WIZ812MJ consists of the W5100 plus a crystal, RJ-45 connector and some gates. These gates combine various LED signals from the W5100 in order to produce a steady (non-flashing) link signal. An inverter uses the SPI chip select (/SCS) output to generate the inverted SPI enable (SEN) signal. This means the /SCS connection between controller and Internet chip is all you need to initiate SPI mode.

The internal core voltage of 1.8 V is produced by the W5100 itself. These and all other operating voltages are stabilised by capacitors on the WIZ812MJ module, although for the sake of clarity they (and the external 3.3 V regulator) are omitted from the schematic.

The Physical Layer

A symmetrical transformer is integrated within the RJ-45 connector. This enables the PHY in the W5100 to swap RX and TX signal lines when necessary. This function is known as auto crossover (auto-MDIX for short) and avoids the need for crossover cables. Consequently you can use normal patch cables for the direct connection to the PC.

The auto-negotiation function determines the physical data rate (10 Mbit/s or 100 Mbit/s) automatically, together with other counterpart connection parameters such as full or half duplex. Once complete, point-to-point connection is made and the link is established.

Signals can be transposed within the pairs RX+/RX and TX+/TX. At 10 Mbit/s speeds the polarity of the PHY will be recognised and inverted when required (polarity reversal). A different technique is used at 100 Mbit/s. There is no longer any polarity here, as ones are signalled as a level change and zeros by breaks. To guarantee reliable clock retrieval, the bitstream is transcoded specially so that excessive numbers of ones or zeros are never sent in direct succession.

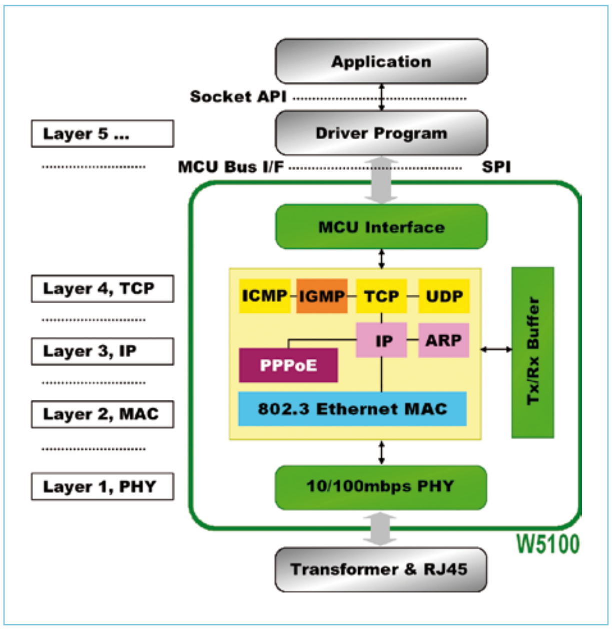


Figure 2. On the W5100 the PHY, MAC and TCP/IP protocols are embedded in hardware to reduce the load on the controller (and on the programmer!).

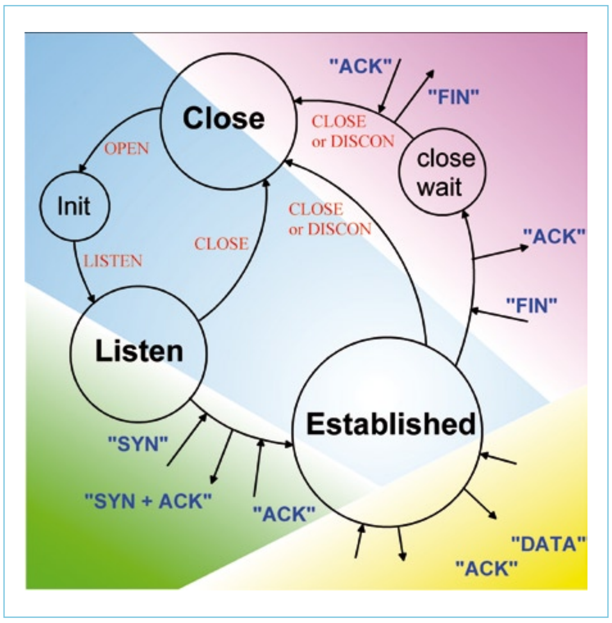


Figure 3. Status diagram of a web server. The W5100 takes charge of the boring details and corresponding driver functions, with the status management function operations clearly visible in the main program (see Listing 2)!

The TCP/IP protocol

Network connections are always regarded as layers that are constructed one above the other, defined in general terms in the so-called ISO/OSI levels model. In this system data and commands from a higher level are passed down to the next one below, with data and replies received back. In contrast to this kind of 'vertical communication' a virtual point-to-point connection is established horizontally between two equal levels on either side of the communication link. This calls for a number of protocols at the corresponding levels.

The inner construction of the W5100 and the protocol levels are shown in **Figure 2**. The W5100 handles all tasks in the first four layers (PHY + MAC + TCP/IP). Application and driver program activities are processed at Layer 5 and control the W5100. The driver needs to be interfaced correctly to the microcontroller employed and for the mode of communication required (SPI or parallel); WIZnet provides source code for various 8, 16 and 32-bit controllers free of charge. The tasks handled by the application program are confined then to status control (see below) and the specific application.

The application program also assigns the IP address. For this purpose the W5100 has its so-called Common Register in the memory range from 0x0000 to 0x0030, which stores all network-specific information (IP, Gateway and MAC addresses, Netmask, etc.). This register is cleared if you call a Reset. For this reason the network information needs to be stored in a non-volatile EEPROM location (integrated here in the R32C). This makes it possible to request an IP address and other networking details from a DHCP server (e.g. a DSL router) dynamically.

The Socket Registers (address range 0x0400 to 0x0800) store all configuration and status data for a maximum of four simultaneous connections ('sockets'). This is also where we define whether a socket is to operate as a client or server. A socket represents a virtual point-to-point connection and is effectively the interface to the outside world. Using this register the main program is also able to check the status of connections and react correspondingly following an interrupt or by regular polling. The send and receive store (16kB), address range 0x4000 to 0x8000, can be applied variably to serve up to four sockets.

Setting up connections

Figure 3 illustrates the states of a socket during a TCP connection. If we wish to implement a web server for example, the first thing to do is to configure a socket to Port 80 and initialise it. We do this using the two commands socket() and init(). The socket now changes from 'closed' status via 'init' into 'listen' state and is now in receive mode. A connection is not established a Client (for instance the browser in a PC) calls up data from this web server. TCP connections are always opened after double confirmation on both sides. These conformation data packets (handshaking) are generated independently by the W5100. Only after this is the horizontal connection created, when data can be transferred.

During the connection setup process the status of the socket changes from 'listen' to 'established'. The main program recognises this status and can read out and analyse the

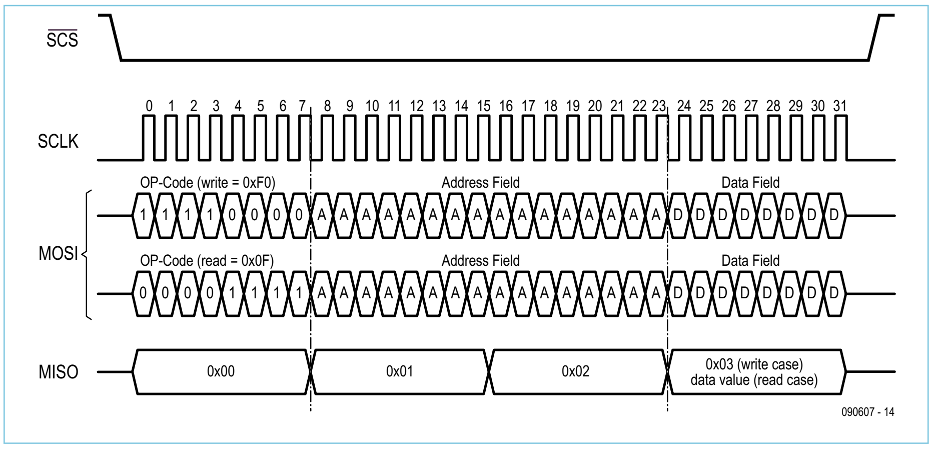


Figure 4. Reading and writing a byte over the SPI bus.

Abbreviations

ISO/OSI: Open System Interconnection (OSI) Reference Model. Defines the various levels of a communication process.

PHY: Physical Layer. Driver for data transmission in a specific medium, e.g. Ethernet. Lowest layer.

MII: Media Independent Interface. Interface between MAC and PHY.

MAC: Media Access Control. Packet relaying via MAC addresses (fixed addresses for relevant hardware, e.g. network cards) at the second lowest layer.

IP (IPv4): Internet Protocol. Address-oriented packet relaying at the third layer. IP addresses can be either fixed or assigned dynamically.

Net Mask: 32-bit word for dividing the IP address into the network address and a device address within the network.

DHCP: Dynamic Host Configuration Protocol. Enables automatic configuration of a network user. For instance DHCP is used to assign IP addresses dynamically to users.

Port: A single IP address can support several applications simultaneously, with each of these applications be assigned a different port. Specific port numbers are assigned by default for many applications.

TCP: Transport Control Protocol. Packet relaying for various services at the fourth layer with confirmation signal (acknowledgement).

UDP: User Datagram Protocol. Communication without acknowledgement, offering advantages of speed, e.g. for wireless Internet.

Socket: TCP/IP service or 'horizontal' connection at the fourth layer for delivering incoming data packets to the appropriate application process or thread. A socket is defined by a combination of IP address and port.

HTTP: Hyper-Text Transport Protocol. TCP service for retrieving web pages.

FTP: File Transfer Protocol. TCP service for transmitting files.

SMTP: Simple Message Transport Protocol. Used for sending e-mails.

Listing 1: Writing data over the SPI bus

```
uint8 WRITE(uint16 addr,uint8 data)
{
  IINCHIP_ISR_DISABLE();          // Interrupt disable
  SET_SCS(LOW);                   // CS=0, SPI start
  SpiSendData(0xF0);              // OP-Code for write
  SpiSendData((addr & 0xFF00) >> 8);// send address high byte
  SpiSendData(addr & 0x00FF);     // send address low byte
  SpiSendData(data);              // send data
  SET_SCS(HI);                    // CS=1, SPI end
IINCHIP_ISR_ENABLE();
  return 1;
}

void SpiSendData(uint8 Val)
{
#ifdef USE_PORT_IO                // software SPI
  uint8 i;

  for (i = 0; i < 8; i++)
  {
        W5100_P_SCLK = LOW;
        if (0x80 & (val << i)) W5100_P_MOSI = HI;
        else W5100_P_MOSI = LOW;
        W5100_P_SCLK = HI;
  }
  W5100_P_SCLK = LOW;
#else                             // hardware SPI
  uart_begin( val );
  uart_end();
#endif
}

void uart_begin( register char cmd )
{
  te_u2c1 = 1;                    // enable tx
  uart_tx( cmd );                 // send the command
}

void uart_end( void )
{
  while( !ti_u2c1 );              // wait for tx buffer to
empty
  while( !txept_u2c0 );           // wait for tx register to
empty
  re_u2c1 = 0;                    // disable rx
  te_u2c1 = 0;                    // disable tx
}
```

received data (for instance the URL of a website requested) from the receive store of the W5100. If requested, the reply data will be written to the output store and the command send() given. When the Client closes the connection, having received all of the desired data (in multiple packets should the need arise), the socket reverts to 'closed' after the corresponding handshake, The socket will now need to be initialised afresh before a Client can be registered again.

Additional protocol information, checksums, flags and so forth are processed by the W5100 internally during transmission and reception of data, with only the payload itself put into store.

The W5100 can handle a maximum of four sockets simultaneously and independently from one another. This means the server remains reachable when a socket is occupied (established), momentarily shut off (closed) or is just initialising itself afresh (init). Different sockets can also act simultaneously as a server or as differing clients and consequently monitor various ports or connect to servers. A socket can be a DHCP client first and when activated call up the network configuration from a DHCP server. Afterwards this socket can then become a web server. At the same time another socket can be forwarding an e-mail by SMTP. The third port is then (for example) another web server and the fourth an FTP server. Source code for all these applications is available free from the chip manufacturer.
You can check out more on the subject of TCP/IP protocol on the Internet, for example in the 'TCP/IP Guide' [3].

Accessing the outside world

As already mentioned, the R32C application board provides not only SPI control but also parallel connection in indirect mode. Using parallel data transmission the W5100 is in a position to achieve data rates up to 25 Mbit/s. Anybody who feels like it is welcome to implement indirect mode and check out the maximum data rate of the R32C/111, but for this article we have concentrated on SPI mode. **Figure 4** sets out the signal timings of the SPI interface for writing and also for reading a byte. The protocol is extremely simple and consists of four bytes. The preamble (opcode) signalises whether reading or writing is being performed. The middle two bytes contain the 16-bit address and the fourth byte the actual data. To read or write a single byte thus requires four bytes. Maximum clock rate at 14 MHz is so large, however, that a data rate of up to 3.5 Mbit/s can be achieved.

The SPI interface can be realised in software on any desired port pins of a microcontroller. The R32C/111 and many other microcontrollers support SPI with a synchronous serial interface but also in hardware. Renesas calls this method 'clock synchronous serial interface mode'. In the downloadable driver source code files you will find both implementations and can swap between them simply with the compiler directive 'USE_PORT_IO'.

Chris Vossen (Elektor Labs) und Jinbuhm Kim (Head of Application & Software for WIZnet) have written an SPI driver for the Renesas M16C product family, especially for the R32C. **Listing 1** includes extracts from the SPI driver. First the interrupts are switched off, in order to avert further interruptions. Then the SPI chip select (/SCS) is set and the four bytes are sent sequentially. Following this the SPI chip select is removed and interrupts are activated once more. Sending a single byte follows by software or hardware SPI.

Sample web server

A small, very simple sample application is shown in **Listing 2**. The Elektor server monitors Port 5000 and responds to any request that includes the string 'elektor'. The main loop splits into a switch case statement according to the status of the connection in the subroutines SOCK_ESTABLISHED, SOCK_CLOSE_WAIT and SOCK_CLOSED. If the socket is closed the socket is reactivated with the commands socket() and init(). If data is being received the server responds with („elektor").
A step from this 'text server' to a proper web server is taken by initialising the socket on port 80 and performing an analysis rou-

Architecture and source code files

The web server application consists of a number of source code files that handle the functions listed above. In detail these are:

Main.c: The actual web server and status management controller. Can be customised for implementing an individual server.

Web server.c: Fundamental web server functions that can be used without alteration for users' own applications.

HTTPD.c: HTTP functions that can be used without alteration for users' own applications.

SOCKET.c: Functions for socket handling that can be used without alteration for users' own applications.

W5100.c: Driver (here in a version for hardware/software-SPI and the R32C controller). Requires rewriting if another form of communication (parallel) is employed.

hwsetup.c: R32C-specific hardware set-up.

OLED28.c: Driver for the OLED.

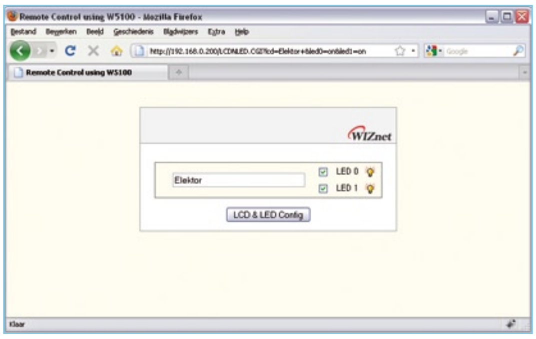
If a different controller is used then W5100.c and hwsetup.c will need to be adapted correspondingly (the chip manufacturer has, however, already produced drivers for some controllers [1]).

Listing 2: Main loop 'Elektor Server'

```
socket(i, Sn_MR_TCP, 5000, 0x20);  // open Socket, Port 5000
listen(i);        // go into listen mode = Server
while(1)
{
  sock_status = getSn_SR(i);   // get socket status
  switch(sock_status)
  {
    case SOCK_ESTABLISHED:
      len = getSn_RX_RSR(i);    // get size of buffer
      if (len > 0)
      {
        if (len > MAX_BUF_SIZE) len = MAX_BUF_SIZE;
        len = recv(i, sock_buf, len);   // return received size
        send(i,"elektor\r\n", 9);  // send „elektor"
      }
      break;
    case SOCK_CLOSE_WAIT:
      disconnect(0);     // send „FIN, ACK"
      break;
    case SOCK_CLOSED:
      close(i);       // close Socket
      socket(i, Sn_MR_TCP, 5000, 0x20);  // reopen Socket
      listen(i);       // to listen mode
      break;
  }
}
```

tine for data received from the client (e.g. URLs from websites you have requested, etc.).

A small, proper web server is waiting for you to download at [4]. **Figure 5** shows what you see in the browser. The application first sends a request by DHCP automatically as client to obtain configuration from the router.


Remote Control using W5100 - Mozilla Firefox

http://192.168.0.200/LCDNLED.CGI?lcd=Elektor+&led0=on&led1=on

Elektor

[✓] LED 0

[✓] LED 1

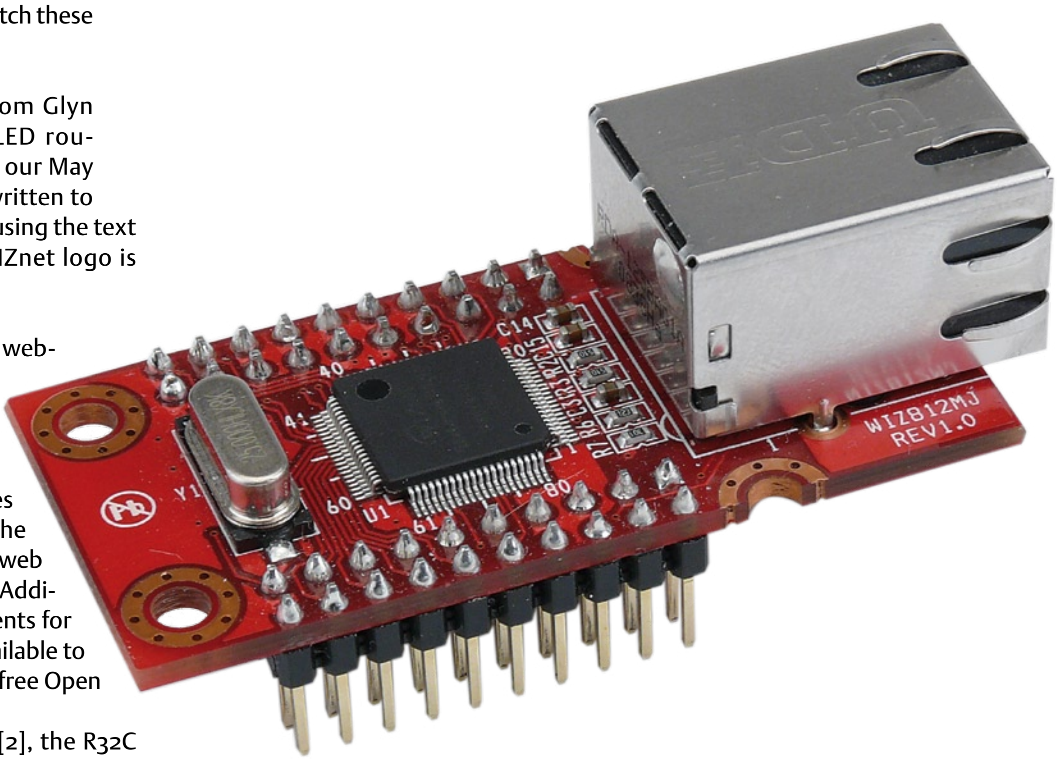
Bestand Bewerken Beeld Geschiedenis Bladwijzers Extra Help
Remote Control using W5100
WIZnet
LCD & LED Config
Klaar

Figure 5. The sample web server software lets you control the LEDs on the application board across the Internet. Words that you type into the text box appear on the OLED!

You now have a web server that will respond to requests as a simple website. This example is a web form with two check boxes for indicating the status of two LEDs on the application board. Naturally the user can also switch these LEDs on and off.

Marc Oliver Reinschmidt from Glyn has also integrated the OLED routines that were described in our May issue [5]. Messages can be written to the OLED display line by line using the text input box. Of course the WIZnet logo is included as well.

All the images shown on the website are compiled statically in our sample program. Anyone who wishes to can also implement an FTP server and bring on board image files from the external SD card. The source code files of the small web server (are well commented. Additional sample servers and clients for a variety of controllers are available to download from WIZnet [1] as free Open Source software.



The R32C application board [2], the R32C starter kit including the controller board [6] and the WIZ812MJ network card are all available from Elektor [4]. This versatile and comprehensive range of hardware has in fact so many possibilities that it won't take long for Elektor readers to come up with their own ideas. Tips and suggestions to the editorial address will be most welcome!

(090607-I)

Internet Links

[1] www.wiznet.co.kr/en/

[2] www.elektor.com/090209

[3] www.tcpipguide.com/free

[4] www.elektor.com/090607

[5] www.elektor.com/081029

[6] www.elektor.com/080928

[7] www.dacomwest.de/e_index.htm

[8] http://en.wikipedia.org/wiki/OSI_model

The authors

Jinbuhm Kim manages the applications department at WIZnet [1] and is head of the support team for their global distribution company.

Joachim Wülbeck is a field application engineer with the distributor Dacom West [7], with responsibility for interfaces and sensor technology.

The authors can be contacted via their organisations' home pages.