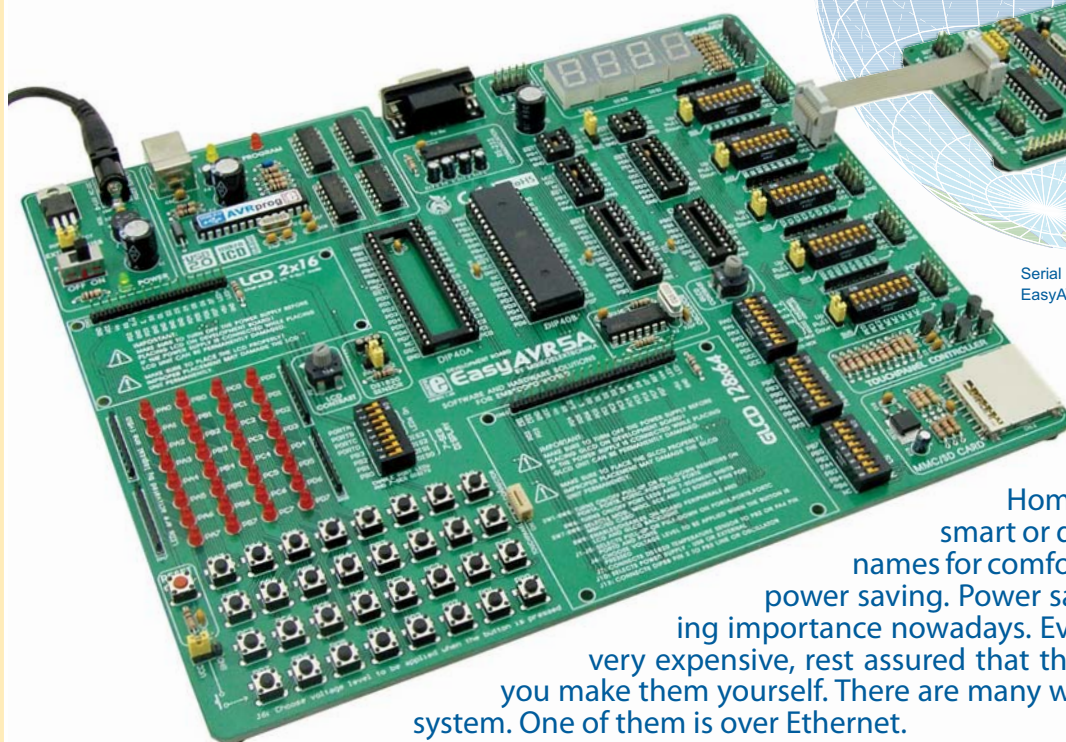


OK. Now you need ... ETHERNET



Serial Ethernet module connected to EasyAVR5A Development System

Home automation, home control, smart or digital home are just different names for comfort, convenience, security and power saving. Power saving systems are of increasing importance nowadays. Even though such systems are very expensive, rest assured that they can also be very cheap if you make them yourself. There are many ways to control a smart home system. One of them is over Ethernet.

By Srdjan Tomic
MikroElektronika - Software Department

All you need is a Atmega 32 microcontroller and an ENC28J60 serial Ethernet chip. This chip is a great solution for other microcontroller families as well such as PIC, dsPIC etc. The CviLux CJCBA8HF1Y0 RJ-45 connector is used for connection to the Ethernet network. An LED connected to the microcontroller PORTB. pin0 simulates a home appliance we want to control.

The *mikroPASCAL for AVR* compiler contains the *SPI_Ethernet* library that will considerably simplify the process of writing a program for the microcontroller. By using a few routines from this library, it is possible to create the program that will enable electrical appliances in your home to be controlled via a web browser.

It is necessary to perform the following operations within the program:

- Step 1.** Create an html page to run the microcontroller through. Import it in the code as a string.
- Step 2.** Set IP, DNS, Gateway addresses and Subnet mask obtained from your internet provider.

For example, our local network parameters are as follows:

IP : 192.168.20.60 (Control System address)
DNS : 192.168.20.1 (Domain Name System address)
GATEWAY : 192.168.20.6 (Gateway address)
SUBNET : 255.255.255.0 (Subnet mask)

- Step 3.** Disable PORTA analogue inputs. The microcontroller pin should be cleared and configured as an output.
- Step 4.** Initialize the SPI module of the Atmega 32 microcontroller.
- Step 5.** Initialize the Serial Ethernet module chip ENC28J60.
- Step 6.** Write the code within the *Spi_Ethernet_userTCP* function that will, after receiving command via web browser, turn on/off the LED connected to the PORTA.0.
- Step 7.** Read received data in an endless loop.

The most important part of the program is the *Spi_Ethernet_userTCP* function, processing all received commands

After the web browser "GET" request is received, sent from your computer to the control system IP address, the microcontroller will respond with a web page stored in its memory. This page will then be automatically displayed on the computer screen by the browser.

When the ON command is received, the LED connected to the PORTA.0 will be turned on.

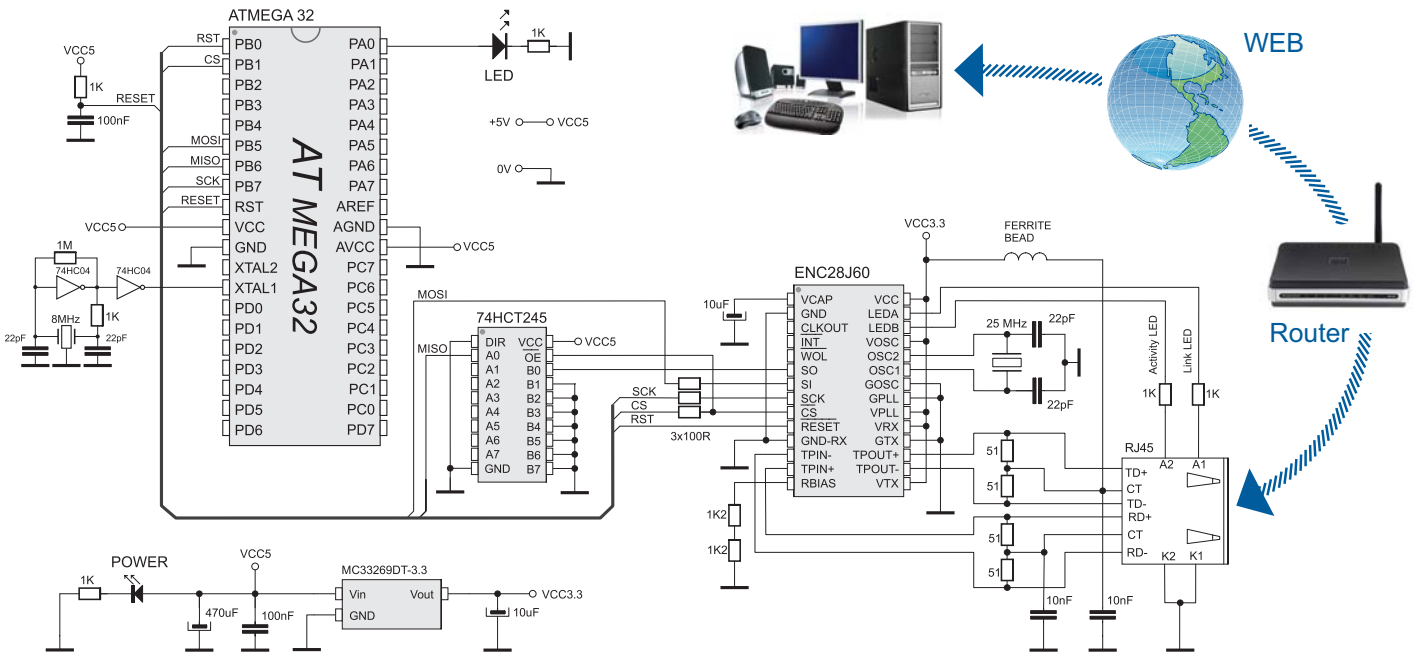
Likewise, when the OFF command is received the LED will be turned off. If you have a relay instead of LED, it is possible to control any appliance such as lighting, security system, heating system etc.

The control of any home appliance consists of entering control system IP address in the web browser and specifying the desired commands.

Of course, it is possible to control more than

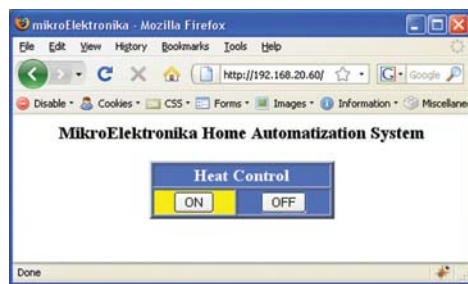


Figure 1. MikroElektronika's Serial Ethernet module with ENC28J60 chip



Schematic 1. Connecting the Serial Ethernet module to a Atmega 32

one microcontroller pin, which enables you to govern a large number of appliances or complete home automation system as well.



The screendump illustrates the web page displayed by the web browser after entering the control system IP address. In our example, ON and OFF button clicks cause the LED to be turned on and off, thus simulating the heating control system.

Example 1: Program to demonstrate control over Ethernet

```

program enc_ethernet;
var myMacAddr : array[6] of byte ; // my MAC address
    myIpAddr : array[4] of byte ; // my IP address
    gwIpAddr : array[4] of byte ; // gateway (router) IP address
    ipMask : array[4] of byte ; // network mask (for example : 255.255.255.0)
    dnsIpAddr : array[4] of byte ; // DNS server IP address
    indexPage : string[523] ; // default html page
    getReq : array[20] of byte ; // HTTP request buffer

// mEhternet NIC pinout
SPI_Ethernet_Rst : sbit at PORTB.0;
SPI_Ethernet_CS : sbit at PORTB.1;
SPI_Ethernet_Rst_Direction : sbit at DDRB.0;
SPI_Ethernet_CS_Direction : sbit at DDRB.1;
// end ethernet NIC definitions

const httpHeader : string[30] = "HTTP/1.1 200 OK"+#10+"Content-type: "; // HTTP header
const httpMimeTypeHTML : string[13] = "text/html"+#10+#10; // HTML MIME type
const httpMimeTypeScript : string[14] = "text/plain"+#10+#10; // TEXT MIME type

function putConstString(const s : ^byte) : word;
begin
    result := 0;
    while(s^ <> 0) do
        Spi_Ethernet_putByte(s^); s := s + 1; result := result + 1;
    end;
end;

function putString(s : ^byte) : word;
begin
    result := 0;
    while(s^ <> 0) do
        Spi_Ethernet_putByte(s^); s := s + 1; result := result + 1;
    end;
end;

function SPI_Ethernet_UserTCP(var remoteHost : array[4] of byte;
    remotePort, localPort, reqLength : word) : word;
var len : word ; // my reply length
begin
    result := string(10);
    if(localPort < 80) then // listen only to web request on port 80
        result := 0; exit;
    end;
    // get 10 first bytes only of the request, the rest does not matter here
    for len := 0 to 14 do getReq[len] := SPI_Ethernet_getByte();
    getReq[len] := 0;

    tmp := 'GET';
    if(memcmp(getReq, @tmp, 5) < 0) then // only GET method
        result := 0; exit;
    end;

    tmp := 'ON';
    if(memcmp(getReq+11, @tmp, 2) = 0) then // do we have ON command
        PORTA.B0 := 1 // set PORTA bit0
    else
        tmp := 'OFF';
        if(memcmp(getReq+11, @tmp, 3) = 0) then // do we have OFF command
            PORTA.B0 := 0; // clear PORTA bit0
        end;
    end;

    if(PORTA.B0) then
        tmp := "#FFFF00"; memcpy(@indexPage+340, @tmp, 6); // highlight (yellow) ON
        tmp := "#974E2"; memcpy(@indexPage+431, @tmp, 6); // clear OFF
    end
    else
        begin
            tmp := "#974E2"; memcpy(@indexPage+340, @tmp, 6); // clear ON
            tmp := "#FFFF00"; memcpy(@indexPage+431, @tmp, 6); // highlight (yellow) OFF
        end;
    len := putConstString(@httpHeader); // HTTP header
    len := len + putConstString(@httpMimeTypeHTML); // with HTML MIME type
    len := len + putString(@indexPage); // HTML page first part
    result := len; // return to the library with the number of bytes to transmit
end;

function SPI_Ethernet_UserUDP(var remoteHost : array[4] of byte;
    remotePort, destPort, reqLength : word) : word;
begin
    result := 0; // back to the library with the length of the UDP reply
end;

begin
    PORTA.B0 bit = 0; // set PORTA as output
    DDRA.B0 := 1; // set PORTA.B0 as output (rel control pin)

    indexPage =
    '<html><head><title>mikroElektronika</title></head><body>'+
    '<h3 align=center>MikroElektronika Home Automatization System</h3>'+
    '<form name="input" action="/" method="get">'+
    '<table align=center width=200 bgcolor=#974E2 border=2><tr>'+
    '<td align=center colspan=2><font size=4 color=white><b>Heat Control</b></td>'+
    '<td align=center colspan=2><input type="button" value="ON" width=60'+
    '<type="submit" value="ON"></td><td align=center bgcolor=#FFFF00>'+
    '<input type="button" value="OFF" width=60'+
    '<type="submit" value="OFF"></td></tr></table>'+
    '</form></body></html>';

    myMacAddr[0] := 0x00; myMacAddr[1] := 0x14; myMacAddr[2] := 0xA5;
    myMacAddr[3] := 0x76; myMacAddr[4] := 0x19; myMacAddr[5] := 0x3F;
    ipMask[0] := 255; ipMask[1] := 255; ipMask[2] := 255; ipMask[3] := 0;
    myIpAddr[0] := 192; myIpAddr[1] := 168; myIpAddr[2] := 20; myIpAddr[3] := 60;
    gwIpAddr[0] := 192; gwIpAddr[1] := 168; gwIpAddr[2] := 20; gwIpAddr[3] := 6;
    dnsIpAddr[0] := 192; dnsIpAddr[1] := 168; dnsIpAddr[2] := 20; dnsIpAddr[3] := 1;

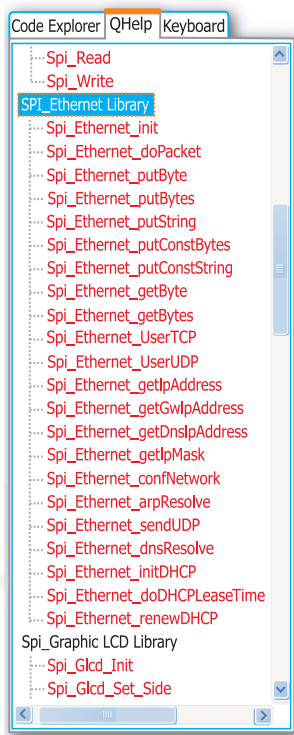
    // starts ENC28J60 with: reset bit on PORTB.F0, CS bit on PORTB.F1,
    SPI_Init_Advance(SPI_MASTER, SPI_FCY_DIV4, SPI_CLK_LO_LEADING);
    SPI_Rd_Prt := @SPI_Read;
    SPI_Ethernet_UserTCP_Prt := @SPI_Ethernet_UserTCP;
    SPI_Ethernet_UserUDP_Prt := @SPI_Ethernet_UserUDP;
    SPI_Ethernet_Init(myMacAddr, myIpAddr, SPI_Ethernet_FULLDUPLEX);

    // dhcp will not be used here, so use preconfigured addresses
    SPI_Ethernet_confNetwork(ipMask, gwIpAddr, dnsIpAddr);

    while true do
        SPI_Ethernet_doPacket(); // do forever
    end;
end;

```

Below is a list of ready to use functions contained in the SPI Ethernet Library. This library is integrated in mikroPASCAL for AVR compiler.



Spi_Ethernet_Init()*	Init ENC28J60 controller
Spi_Ethernet_Enable()	Enable network traffic
Spi_Ethernet_Disable()	Disable network traffic
Spi_Ethernet_doPacket()*	Process received packet
Spi_Ethernet_putByte()	Store a byte
Spi_Ethernet_putBytes()	Store bytes
Spi_Ethernet_putConstBytes()	Store const bytes
Spi_Ethernet_putString()*	Store string
Spi_Ethernet_putConstString()*	Store const string
Spi_Ethernet_getByte()*	Fetch a byte
Spi_Ethernet_getBytes()	Fetch bytes
Spi_Ethernet_UserTCP()*	TCP handling code
Spi_Ethernet_UserUDP()	UDP handling code
Spi_Ethernet_getIpAddress()	Get IP address
Spi_Ethernet_getGwIpAddress()	Get Gateway address
Spi_Ethernet_getDnsIpAddress()	Get DNS address
Spi_Ethernet_getIpMask()	Get IP mask
Spi_Ethernet_confNetwork()*	Set network parameters
Spi_Ethernet_arpResolve()	Send an ARP request
Spi_Ethernet_sendUDP()	Send an UDP packet
Spi_Ethernet_dnsResolve()	Send an DNS request
Spi_Ethernet_initDHCP()	Send an DHCP request
Spi_Ethernet_doDHCPLeaseTime()	Process lease time
Spi_Ethernet_renewDHCP()	DHCP renewal request
* SPI Ethernet Library functions used in program	

Other mikroPASCAL for AVR functions used in program:

- Spi_Init() Initialize microcontroller SPI module
- memcpy() Copy microcontroller RAM memory locations
- memcmp() Compare microcontroller RAM memory locations

NOTE: Code for this example written for AVR® microcontrollers in C, Basic and Pascal as well as the programs written for dsPIC® and PIC® microcontrollers can be found on our web site: www.mikroe.com/en/article/

