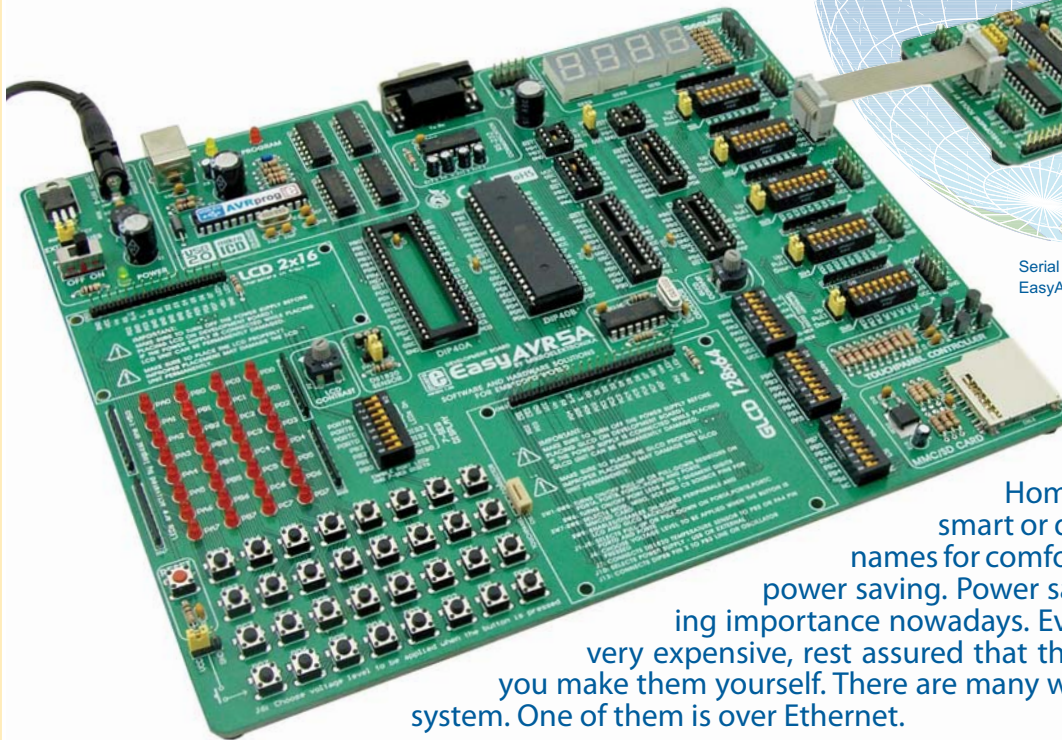


# OK. Now you need ... ETHERNET



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 *mikroC PRO 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.

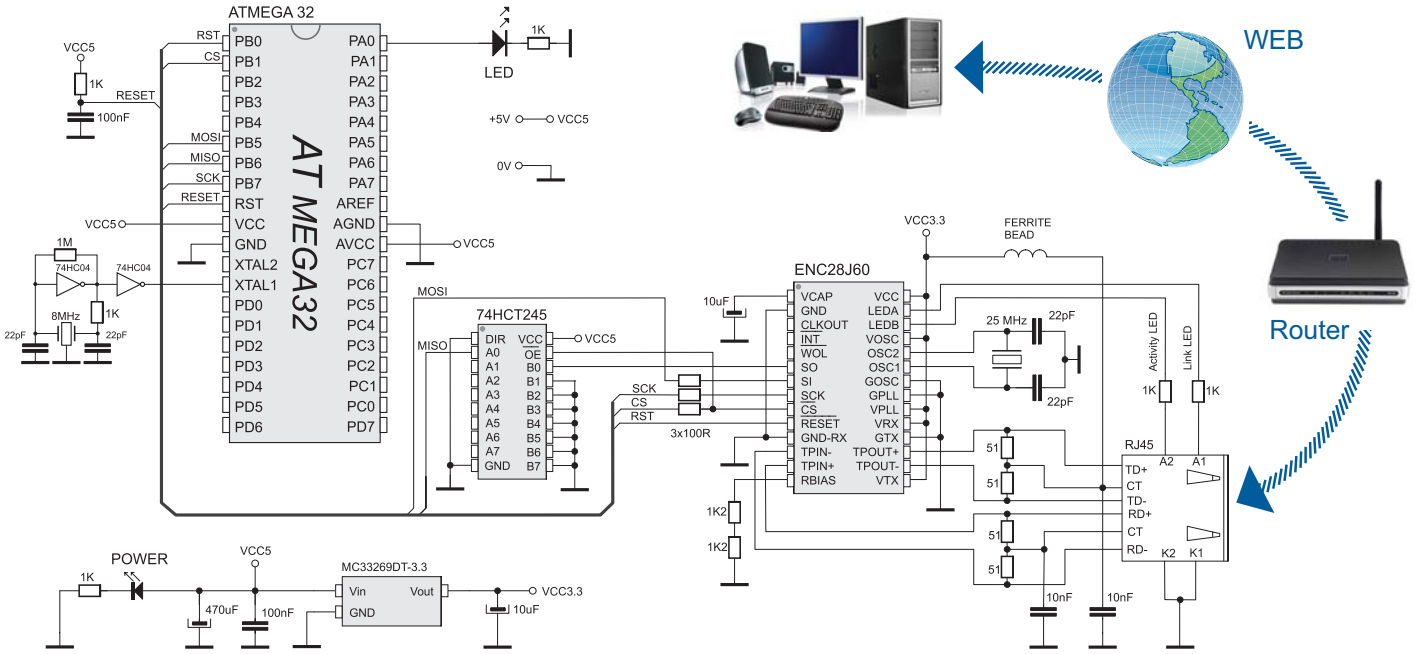
Likewise, when the OFF command is received the LED will be turned off. If you have a relay instead of LED, it is possible to control any appliance such as lighting, security system, heating system etc.

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

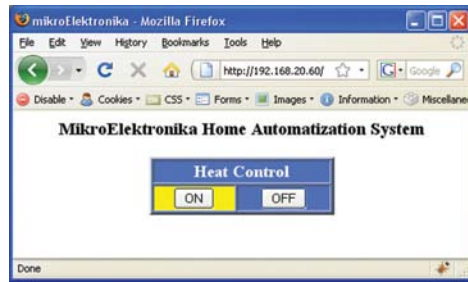


Figure 1. MikroElektronika's Serial Ethernet module with ENC28J60 chip



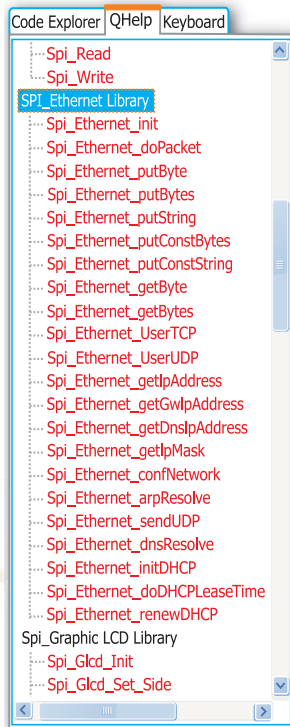
Schematic 1. Connecting the Serial Ethernet module to a Atmega 32

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.

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



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

Other mikroC PRO for AVR functions used in program:

- Spi\_Init()** Initialize microcontroller SPI module
- memcpy()** Copy microcontroller RAM memory locations
- memcmp()** Compare microcontroller RAM memory locations

Example 1: Program to demonstrate control over Ethernet

```
// duplex config flags
#define Spi_Ethernet_HALFDUPLEX 0x00 // half duplex
#define Spi_Ethernet_FULLDUPLEX 0x01 // full duplex

// mE ethernet NIC pinout
sfr sbit SPI_Ethernet_Rst at PORTB.B0; // reset pin
sfr sbit SPI_Ethernet_CS at PORTB.B1; // chip select pin
sfr sbit SPI_Ethernet_Rst_Direction at DDRB.B0; // reset pin direction
sfr sbit SPI_Ethernet_CS_Direction at DDRB.B1; // chip select pin direction
// end ethernet NIC definitions

const char httpHeader[] = "HTTP/1.1 200 OK\nContent-type: "; // HTTP header
const char httpMimeTypeHTML[] = "text/html\n\n"; // HTML MIME type
const char httpMimeTypeScript[] = "text/plain\n\n"; // TEXT MIME type

// default html page
char indexPage[] =
"<html><head><title>mikroElektronika</title></head><body>\n
<h3 align=center>MikroElektronika Home Automatization System</h3>\n
<form name='input'\ action='/' method='get'\n
<table align=center width=200 bgcolor=#4974E2 border=2><tr>\n
<td align=center colspan=2><font size=4 color=white><b>Heat Control</b></font>\n
</td></tr><tr><td align=center bgcolor=#4974E2><input name='tst1'\ width=60\n
type='submit' value='ON'\></td><td align=center bgcolor=#4974E2>\n
<input name='tst2'\ type='submit' value='OFF'\></td></tr></table>\n
</form></body></html>";

// network parameters
char myMacAddr[6] = {0x00, 0x14, 0xA5, 0x76, 0x19, 0x3f}; // my MAC address
char myIpAddr[4] = {192, 168, 20, 60}; // my IP address
char gwIpAddr[4] = {192, 168, 20, 6}; // gateway IP address
char dnsIpAddr[4] = {192, 168, 20, 1}; // dns IP address
char ipMask[4] = {255, 255, 255, 0}; // subnet mask
// end network parameters

unsigned char getRequest[20]; // HTTP request buffer

unsigned int SPI_Ethernet_UserTCP( char *remoteHost, unsigned int remotePort,
unsigned int localPort, unsigned int reqLength)
{
  unsigned int len; // my reply length
  if(localPort != 80) return(0); // I listen only to web request on port 80

  // get 10 first bytes only of the request, the rest does not matter here
  for(len = 0; len < 15; len++) getRequest[len] = SPI_Ethernet_getByte();
  getrequest[len] = 0;

  if(memcmp(getRequest, "GET /", 5)) return(0); // only GET method

  if(memcmp(getRequest+11, "ON", 2)) // do we have ON command
    PORTA.F0 = 1; // set PORTA bit0
  else // do we have OFF command
    PORTA.F0 = 0; // clear PORTA bit0

  if(PORTA.F0)
  {
    memcpy(indexPage+340, "#FFFF00", 6); // highlight (yellow) ON
    memcpy(indexPage+431, "#4974E2", 6); // clear OFF
  }
  else
  {
    memcpy(indexPage+340, "#4974E2", 6); // clear ON
    memcpy(indexPage+431, "#FFFF00", 6); // highlight (yellow) OFF
  }

  len = SPI_Ethernet_putConstString(httpHeader); // HTTP header
  len += SPI_Ethernet_putConstString(httpMimeTypeHTML); // with HTML MIME type
  len += SPI_Ethernet_putString(indexPage); // HTML page first part
  return len; // return to the library with the number of bytes to transmit
}

unsigned int SPI_Ethernet_UserUDP( char *remoteHost, unsigned int remotePort,
unsigned int destPort, unsigned int reqLength)
{
  return 0; // back to the library with the length of the UDP reply
}

void main()
{
  // set PORTA as output
  PORTA0_bit = 0; // clear PORTA.B0
  DDRA.F0 = 1; // set PORTA.B0 as output (rele control pin)

  // starts ENC28J60 with: reset bit on PORTB.F0, CS bit on PORTB.F1,
  // my MAC & IP address, full duplex
  SPI1_Init_Advanced(SPI_MASTER, SPI_FCY_DIV4, SPI_CLK_LO_LEADING);
  SPI_Rd_Ptr = SPI1_Read; // pass SPI Read function of used SPI module
  // full duplex, CRC + MAC Unicast + MAC Broadcast filtering
  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(1) {
    SPI_Ethernet_doPacket(); // do forever
  }
}
```

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/](http://www.mikroe.com/en/article/)

