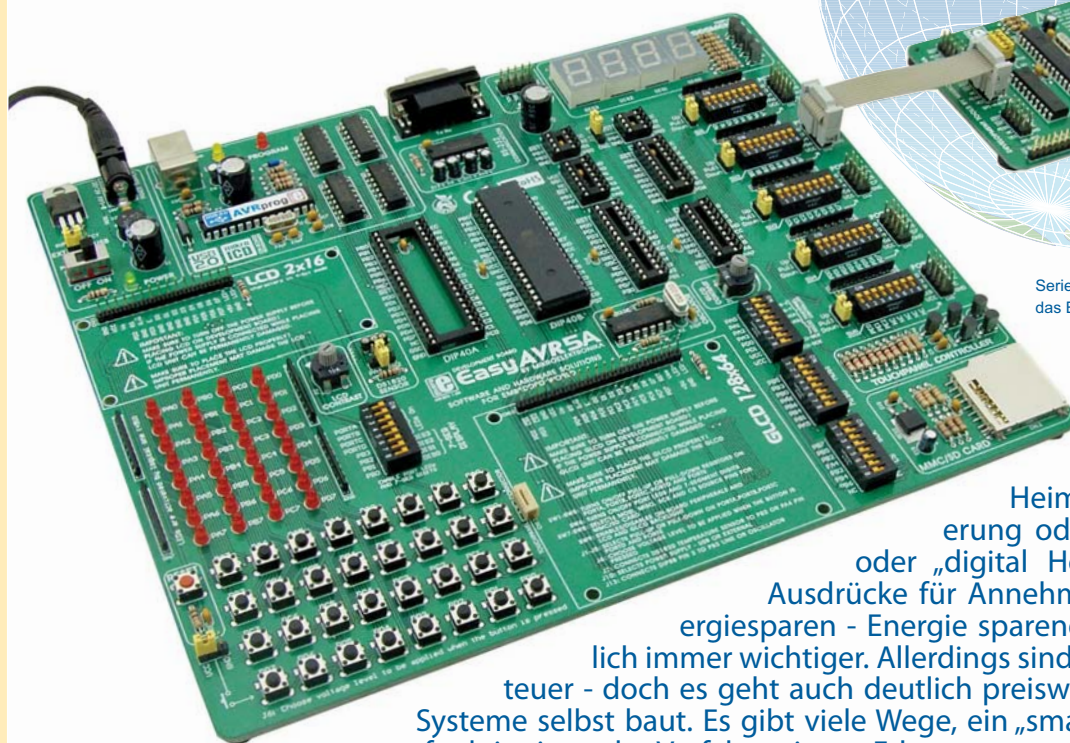


OK. Jetzt brauchen sie... ETHERNET



Serielles Ethernet-Modul mit Anschluss an das Entwicklungs-System EasyAVR5A

Heim-Automatation, Heim-Steuerung oder das so genannte „smart“ oder „digital Home“ sind schlicht andere Ausdrücke für Annehmlichkeit, Sicherheit und Energiesparen - Energie sparende Systeme werden schließlich immer wichtiger. Allerdings sind solche Systeme oft reichlich teuer - doch es geht auch deutlich preiswerter, wenn man sich solche Systeme selbst baut. Es gibt viele Wege, ein „smart Home“ zu realisieren. Ein gut funktionierendes Verfahren ist es, Ethernet zu nutzen.

Von Srdjan Tomic
MikroElektronika - Software-Abteilung

Es wird lediglich ein Atmega 32-Mikrocontroller und ein serieller Ethernet-Chip vom Typ ENC28J60 benötigt. Dieser Chip ist eine gute Ergänzung auch für andere Mikrocontroller-Familien wie dsPIC oder die Controller von Atmel etc. Für den Anschluss an das Ethernet-Netzwerk wird eine Steckverbindung vom Typ CviLux CJCBA8HF1Y0 (RJ-45) verwendet. Eine an den Pin PORTA.0 des Mikrocontrollers angeschlossene LED simuliert das zu steuernde Gerät.

Der Compiler mikroPASCAL for AVR enthält die Library SPI_Ethernet, welche das Schreiben von Software für den Mikrocontroller drastisch vereinfacht. Durch Verwendung nur weniger Routinen dieser Library ist es möglich, ein Programm zu erstellen, das es erlaubt, alle gesteuerten Geräte im Heim auf einfache Weise via Web-Browser fernzusteuern.

Hierzu sind folgende Schritte im Programm erforderlich:

- Schritt 1.** Erstelle eine HTML-Seite für den Mikrocontroller. Diese Seite wird dann als String importiert.
- Schritt 2.** Setzen der IP-, DNS- und Gateway-Adressen sowie der Subnetz-Maske entsprechend der Vorgaben des Internet-Providers.

Die lokalen Netzwerk-Parameter könnten zum Beispiel so aussehen:

IP: 192.168.20.60 (Adresse des Geräts)
DNS: 192.168.20.1 (Adresse des Domain-Name-Systems)
GATEWAY: 192.168.20.6 (Gateway-Adresse)
SUBNET: 255.255.255.0 (Subnetz-Maske)

Schritt 3. Deaktivieren der analogen Eingänge von PORTA. Der jeweilige Pin wird gelöscht und als Ausgang definiert.

Schritt 4. Initialisieren des SPI-Moduls des Atmega 32-Mikrocontrollers.

Schritt 5. Initialisieren des seriellen Ethernet-Modul-Chips ENC28J60.

Schritt 6. Schreiben des Codes innerhalb der Spi_Ethernet_userTCP-Funktion, welcher nach Empfang eines Befehls via Web-Browser die an PORTA.0 angeschlossene LED ein bzw. ausschaltet.

Schritt 7. Lesen der zu empfangenden Daten in einer Endlos-Schleife.

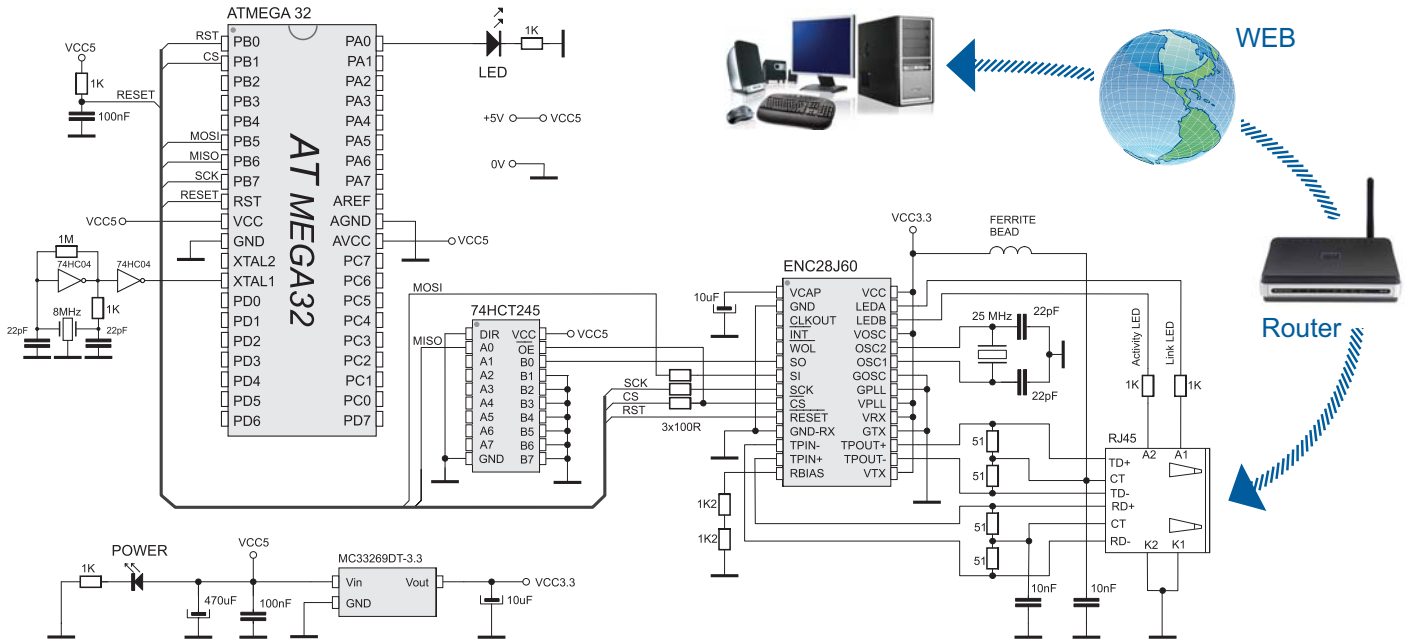
Der wichtigste Teil des Programms ist die Spi_Ethernet_userTCP-Funktion, die empfangene Befehle ausführt.

Nachdem eine „GET“-Anfrage des Web-Browsers empfangen wurde, die Ihr PC an die IP-Adresse des zu steuernden Geräts geschickt hat, wird der Mikrocontroller mit einer Web-Seite antworten, die in seinem Speicher abgelegt ist. Diese Web-Seite wird dann automatisch im Browser des PCs angezeigt werden. Wenn ein „ON“-Befehl empfangen wurde, wird die an PORTA.0 angeschlossene LED leuchten. Entsprechend wird ein empfangener „OFF“-Befehl die LED wieder verlöschen lassen. Ist ein Relais anstelle einer LED angeschlossen, kann das Gerät diverse andere Geräte wie Lampen, Alarmanlagen, Heizungen etc. steuern.

Die Steuerung von Geräten erfolgt prinzipiell, indem die IP-Adresse des Geräts in die Adresszeile des Web-Browsers eingegeben wird und die gewünschten Befehle spezifiziert werden. Selbstverständlich kann mehr als nur ein einziger Pin eines Mikrocontrollers gesteuert werden. Auf diese Weise können viele unter-

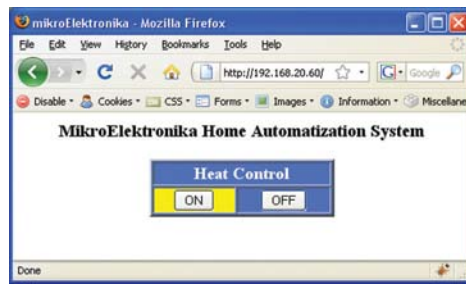


Figure 1. MikroElektronika's Serielles Ethernet-Modul mit ENC28J60 chip



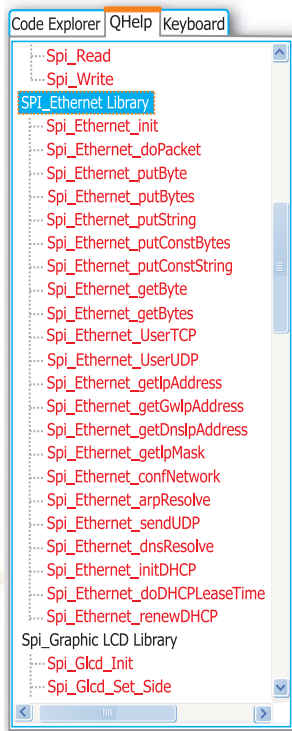
Schaltung 1. Anschluss des Serial -Ethernet-Module an einen Atmega 32

schiedliche Geräte gesteuert oder auch ein komplettes Home-Automation-System realisiert werden.



Das Bildschirmsfoto zeigt die Web-Seite, die der Web-Browser anzeigt, nachdem man die IP-Adresse eines steuerbaren Geräts eingegeben hat. In unserem Beispiel bewirken Klicks auf die Schaltflächen „ON“ und „OFF“, dass eine angeschlossene LED ein- oder ausgeschaltet wird, womit man die korrekte Funktion z.B. einer Heizung simulieren könnte.

Nachfolgend die Liste vorgefertigter Befehle, wie sie in der Library „SPI Ethernet“ enthalten sind. Diese Library ist im Compiler mikroPASCAL for AVR integriert.



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

Andere im Programm verwendete Funktionen von mikroPASCAL for AVR:

Spi_Init() Initialisiere das Mikrocontroller-SPI-Modul
memcpy() Kopiere den RAM-Inhalt des Mikrocontrollers
memcmp() Vergleiche die RAM-Inhalte des Mikrocontrollers

Beispiel 1: Demo-Programm zur Steuerung via 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)
    dnsAddr : array[4] of byte; // DNS server IP address
    indexPage : string[523]; // default html page
    getReqest : array[20] of byte; // HTTP request buffer

// mE Ethernet 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
    begin
        Spi_Ethernet_putByte(s); s := s + 1; result := result + 1;
    end;
end;

function putString(s : byte) : word;
begin
    result := 0;
    while(s <> 0) do
    begin
        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
    len := string(10);
    if(localPort < 80) then // I listen only to web request on port 80
    begin
        result := 0; exit;
    end;
    // get 10 first bytes only of the request, the rest does not matter here
    for len := 0 to 14 do getReqest[len] := SPI_Ethernet_getByte();
    getReqest[len] := 0;
    tmp := 'GET';
    if(memcmp(getReqest, @tmp, 5) < 0) then // only GET method
    begin
        result := 0; exit;
    end;
    tmp := 'ON';
    if(memcmp(getReqest+11, @tmp, 2) = 0) then // do we have ON command
        PORTA.B0 := 1 // set PORTA bit 0
    else
    begin
        tmp := 'OFF';
        if(memcmp(getReqest+11, @tmp, 3) = 0) then // do we have OFF command
            PORTA.B0 := 0; // clear PORTA bit 0
        end;
    end;
    if(PORTA.B0) then
    begin
        tmp := "#FFFF00"; memcpy(@indexPage+340, @tmp, 6); // highlight (yellow) ON
        tmp := "#4974E2"; memcpy(@indexPage+431, @tmp, 6); // clear OFF
    end
    else
    begin
        tmp := "#4974E2"; 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;

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

indexPage =
<html><head><title>mikroElektronika</title></head><body>+
<h3 align=center>MikroElektronika Home Automation System</h3>+
<form name="input" action="/" method="get">
<table align=center width=200 bgcolor=#4974E2 border=2><tr>+
<td align=center colspan=2><font size=4 color=white><b>Heat Control</b></td></tr>+
<tr>+
<td align=center colspan=2><input type="button" value="ON" width=60 +
" type="submit" value="ON"></td><td align=center bgcolor=#FFFF00; +
" 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,
// my MAC & IP address, full duplex
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 not be used here, so use predefined addresses
SPI_Ethernet_confNetwork(ipMask, gwIpAddr, dnsIpAddr);

while true do // do forever
SPI_Ethernet_doPacket(); // process incoming Ethernet packets
end.

```

NOTE: Beispiel-Code für AVR®-Mikrocontroller in C, Basic und Pascal sowie andere Programme für dsPIC®- und PIC®-Mikrocontroller finden sich auf unserer Web-Seite: www.mikroe.com/en/article/

