EtherCard is a driver for the ENC28J60 chip, compatible with Arduino IDE.

Adapted and extended from code written by Guido Socher and Pascal Stang.
The home page for this library is at http://jeelabs.net/projects/ethercard/wiki.
License: GPL2

PIN Connections (Using Arduino UNO):

EtherCard	Arduino UNO
VCC	3.3V
GND	GND
SCK	Pin 13
SO	Pin 12
SI	Pin 11
CS	Pin 8

Introduction

EtherCard is a library that performs low-level interfacing with network interfaces based on the MicroChip (C) **ENC28J60**. High-level routines are provided to allow a variety of purposes including simple data transfer through to HTTP handling. **EtherCard** is a driver for the **ENC28J60** chip, compatible with Arduino IDE.

Adapted and extended from code written by Guido Socher and Pascal Stang.

The documentation for this library is at http://jeelabs.net/pub/docs/ethercard/

The code is available on GitHub, at https://github.com/jcw/ethercard - to install:

Download the ZIP file from https://github.com/jcw/ethercard/archive/master.zip From the Arduino IDE: Sketch -> Import Library... -> Add Library... Restart the Arduino IDE to see the new "EtherCard" library with examples See the comments in the example sketches for details about how to try them out.

For questions and help, see the forums. For bugs and feature requests, see the issue tracker.

Examples

Several **example scripts** are provided with the library which demonstrate various features. Below are descriptions on how to use the library.

Note: **ether** is defined globally and may be used to access the library thus: ether.*member*.

To initiate the library call EtherCard::begin

```
uint8_t Ethernet::buffer[700]; // configure buffer size to 700 octets
static uint8_t mymac[] = { 0x74,0x69,0x69,0x2D,0x30,0x31 }; // define (unique on LAN) hardware (MAC) address

uint8_type nFirmwareVersion ether.begin(sizeof Ethernet::buffer, mymac);
if(0 == nFirmwareVersion)
{
    // handle failure to initiate network interface
}
```

To configure IP address via DHCP use EtherCard::dhcpSetup

```
if(!ether.dhcpSetup())
{
// handle failure to obtain IP address via DHCP
}
ether.printIp("IP:   ", ether.myip); // output IP address to Serial
ether.printIp("GW:   ", ether.gwip); // output gateway address to Serial
ether.printIp("Mask: ", ether.mymask); // output netmask to Serial
ether.printIp("DHCP server: ", ether.dhcpip); // output IP address of the DHCP server
```

To configure a static IP address use EtherCard::staticSetup

```
const static uint8_t ip[] = {192,168,0,100};
```

```
const static uint8_t gw[] = {192,168,0,254};
const static uint8_t dns[] = {192,168,0,1};

if(!ether.staticSetup(ip, gw, dns);
{
    // handle failure to configure static IP address (current implementation always returns true!)
}
```

Send UDP packet

To send a UDP packet use **EtherCard::sendUdp**

```
char payload[] = "My UDP message";
uint8_t nSourcePort = 1234;
uint8_t nDestinationPort = 5678;
uint8_t ipDestinationAddress[4];
ether.parseIp(ipDestinationAddress, "192.168.0.200");

ether.sendUdp(payload, sizeof(payload), nSourcePort, ipDestinationAddress, nDestinationPort);
```

DNS Lookup

To perform a DNS lookup use **EtherCard::dnsLookup**

```
if(!ether.dnsLookup("google.com"))
{
    // handle failure of DNS lookup
}
ether.printIp("Server: ", ether.hispip); // Result of DNS lookup is placed in the hisip member of EtherCard.
```