Bachelorarbeit

Frequenzmessung mit ATMEL AVR-Mikrocontroller

Messung der Netzfrequenz und Übermittlung der Daten an einen Server über einen Ethernet-Mikrochip

ausgeführt zur Erlangung des akademischen Grades
eines Bachelor of Science unter der Leitung von

O.Univ.Prof. Dipl.-Ing. Dr.techn. Dietmar Dietrich
und
Projektass. Dipl.-Ing.(FH) Thomas Leber

am

Institut für Computertechnik (E384)
der Technischen Universität Wien

durch

Mehmet Ozgan
Matr.Nr. 0526530
Preysinggasse 18/2221
A-1150 Wien

Wien, am 27. Juni 2014 _____

Kurzfassung

Diese Arbeit beschreibt eine Messmethode, die mit Hilfe elektronischer Bauteile die Netzfrequenz messen soll. Dazu wird ein Board entworfen und mit zusätzlichen Modulen wie Optokoppler, Atmel AVR-Mikrocontroller, Netzwerk und Netzadapter ausgestattet. Der Mikrocontroller soll über den Optokoppler die Netzfrequenz messen und diese Frequenzgrößen nach der Abfrage über das Netzwerk-Modul an einen Server weiterleiten.

Um die Netzfrequenz genau zu messen, wird das kontinuierlich-sinusförmige Eingangssignal mit dem Optokoppler digitalisiert und zu dem Eingangspin des ATmega16-Mikrocontrol- lers geleitet. Nachfolgend wird das digitalisierte Signal mittels dem Timer-Treiber aufgezählt, welcher in der Programmierungssprache C geschrieben wird. Anschließend werden diese Frequenzgrößen nach der Abfrage über das eigens konzipierte Netzwerk-Modul in Form von UDP-Paketen an einen Server weitergeleitet. Um die Frequenzgrößen abzufragen, wird ein Dämon für die UNIX basierten Systeme programmiert, welcher mit dem Netzwerk-Modul ein Kommunikationsprotokoll erstellt.

Abstract

This paper describes a measurement method to measure the mains frequency with the help of electronic components. For this purpose, a board is designed and equipped with additional modules such as optocouplers, Atmel AVR microcontrollers, network, and power adapter. The microcontroller is designed to measure the grid frequency via the optocoupler and forward this frequency magnitude according to the query over the network module to a server.

To measure the grid frequency exactly, the sinusoidal input signal is continuously digitized with the help of the optocoupler and passed to the input pin of the ATmega16 microcontroller. Subsequently, the digitized signal is counted by the timer driver which is written in the programming language C. Then this frequency sizes will be redirected to a server with the query via the specially designed network module in the form of UDP packets. To query the frequency magnitudes, a daemon for UNIX-based systems is programmed, which with the network module a communication protocol creates.

Inhaltsverzeichnis

ABKÜRZUNGSVERZEICHNIS

AC	Alternating Current
API	Application Programming Interface
ARP	Address Resolution Protocol
ARPANET	Advanced Research Projects Agency Network
ASCII	American Standard Code for Information Interchange
AVR	8-Bit-Mikrocontroller des Herstellers Atmel
bzw.	beziehungsweise
BSD	Berkeley Software Distribution
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CTC	Clear Timer on Compare Match
DC	Direct Current als Gleichstrom
DNS	Domain Name System
DoD	Department of Defense
EEPROM	Electrically Erasable Programmable Read-Only Memory
FAQ	Frequently Asked Questions
FCS	Frame Check Sequence
FIFO	First In – First Out
FreeBSD	Ein freies und modernes UNIX-basiertes Betriebssystem von Berkeley Software Distribution (BSD)
FTP	File Transfer Protocol
GIF	Graphics Interchange Format
GNU/Linux	UNIX-ähnliche Mehrbenutzer-Betriebssysteme, die auf dem Linux-Kernel und wesentlich auf GNU-Software basieren
HTTP	Hypertext Transfer Protocol
Hz	Hertz (Einheit)
IANA	Internet Assigned Numbers Authority
I^2C	Inter-Integrated Circuit
ICF	Input Capture Flag
ICMP	Internet Control Message Protocol
ICP	Input Capture Pin
IEEE	Institute of Electrical and Electronics Engineers
IGMP	Internet Group Management Protocol
IHL	Internet Header Length

inkl.	inklusive
INT	Interrupt
IP	Internet Protocol
ISO	International Organization for Standardization
JTAG	Joint Test Action Group
KB	Kilobyte (Einheit)
LAN	Local Area Network
Led	Light-emitting Diode
LLC	Logical Link Control
MAC	Media Access Control
MHz	Megahertz (Einheit)
MISO	Master in, Slave out
MOSI	Master out, Slave in
MPEG	Moving Picture Experts Group
NNTP	Usenet News Transfer Protocol
OSI-Modell	Open Systems Interconnection Model
PHY	Physical Layer
POSIX	Portable Operating System Interface
QoS	Quality of Service
RAM	Random-access Memory
RARP	Reverse Address Resolution Protocol
RFC	Request for Comments
RISC	Reduced Instruction Set Computing
RJ	Registered Jack
RTC	Real Time Clock (Echtzeituhr)
RX	Pin for Receive Data
sek.	Sekunde (Einheit)
SCK	SPI Bus Serial Clock
SFD	Start Frame Delimiter
SMB	Server Message Block
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
sog.	so genannt
SPI	Serial Peripheral Interface
SRAM	Static Random-access Memory
$\overline{SS}$	Slave Select
TAP	Test Access Port
TCNT	Timer Counter
TCP	Transmission Control Protocol
Telnet	Telecommunication Network
TIFF	Tagged Image File Format
TX	Pin for Transmit Data
UART	Universal Asynchronous Receiver Transmitter
UDP	User Datagram Protocol
UNIX	Ein Mehrbenutzer-Betriebssystem und wurde im JAhr 1969 von Bell Labs bei AT&T entwickelt
USART	Universal Synchronous/Asynchronous Receiver Transmitter
USB	Universal Serial Bus
V	Volt (Einheit)
Xerox PARC	Xerox Palo Alto Research Center

1 EINFÜHRUNG

1.1 Motivation

Um die elektrotechnischen Geräte in Betrieb nehmen zu können, wird eine Versorgungsspannung von einer Spannungsquelle benötigt, die als Gleich- oder Wechselspannungsquelle bezeichnet wird. Die Gleichspannungsquelle hat zu jedem Zeitpunkt einen konstanten Wert. Im Gegensatz hat die Wechselspannung einen periodisch unterschiedlichen Spannungswert, wie die allgemeine Form in der Gleichung 2.1 beschrieben wird. Die Periode ist eine Eigenschaft eines Vorgangs, der in einer gewissen Dauer beschränkt ist und in der laufenden Zeit wiederholt auftritt. Und die Frequenz ist ein Maß, wie schnell ein periodischer Vorgang regelmäßig aufeinander auftritt. Aus diesem Grund wird die Frequenz der Wechselspannung zur Erklärung herausgegeben. Der Zusammenhang zwischen der Periode T und der Frequenz f ist beschreibbar wie

$$Frequenz[Hz] = \frac{1}{Periodendauer[s]} \tag{1.1}$$

und bedeutet, dass die Frequenz die Anzahl der Schwingungen pro Sekunde ist.

1.2 Problemstellung und Zielsetzung

Bei sinkender Stromstärke spielt die Frequenz eine wichtige Rolle. Aus diesem Grund hat die Netzfrequenz eine große Bedeutung und wird beobachtet. Wenn die Frequenz hoch ist, ist der Spannungswert bzw. der fließende Strom in betriebenen Geräten störend. Wenn der Spannungswert zu niedrig ist, dann fließt Stromstärke zu gering, um Geräte zu betreiben.

Eine Wechselspannung wird mittels einer rotierenden Maschiene erzeugt, die aus einer Spule bzw. einem homogenen Magnetfeld besteht. Die Höhe der Wechselspannung ist von der Drehfrequenz dieses Motors (*Winkelgeschwindigkeit*) abhängig, wie in der folgenden mathematischen Form dargestellt wird:

$$U_{ind}(t) = \frac{U_{max}}{sin(wt)} \tag{1.2}$$

wobei U_{ind} die Induktionsspannung im Verlauf der Zeit und U_{max} die maximale Spannung, welche von der Maschiene erzeugt wird, sind. Hier ist der maximale Spannungswert U_{max} von der magnetischen Spulenfläche und von der Flussdichte abhängig. Umso höher die Drehfrequenz ist, desto schneller ist die Änderung dieser Fläche. Abhängig davon bildet sich der Spannungswert.

In dieser Arbeit soll ein Messgerät für die Netzfrequenz enstehen, das eine möglichst kleine Bauform hat und mit verschwindend kleiner Leistung auskommt. Ein sinusförmiges Signal wird digitalisiert und die Frequenz davon gemessen.

Zusätzlich sollte das Gerät seine Messdaten an einen zentralen Server schicken können, um die Änderungen der gemessenen Netzfrequenz unabhängig vom gemessenen Ort zu beobachten. Außerdem werden die Frequenzwerte an dem Server archiviert. Um diese Messdatensendung zu realisieren, wird ein Kommunikationsprotokoll benötigt und implementiert.

2 Grundlagen

Das nachfolgende Kapitel beschreibt die Grundlagen, welche in dieser Arbeit benötigt werden. Es besteht aus drei Abschnitten. Im ersten Abschnitt wird auf die Darstellung von kontinuierlich-sinusförmige Signale und deren Einheiten sowie die Schreibweise von Gleichungen eingegangen. Der zweite Abschnitt stellt die Kommunikationsprotokolle und deren Schichtenmodelle dar. Die BSD-Sockets werden im letzten Abschnitt erläutert. Außerdem umfasst der letzte Teil auch die Herstellung der Kommunikation mittels BSD-Sockets.

2.1 Elektrotechnische Grundlagen

In der Elektrotechnik wird die elektrische Energie entweder als Gleichstrom oder als Wechselstrom übertragen. Beim Gleichstrom ändern sich weder die Stärke noch die Richtung, aber beim Wechselstrom sind die Stärke und die Richtung im zeitlichen Verlauf veränderlich. Die einfache Erzeugung der Wechselspannung ist mit den elektromagnetischen Motoren möglich, welche rotierende Maschinen sind. Die Richtung der magnetischen Feldlinien ändert sich in einer Spule, dann wird eine Wechselspannung in diese sich drehenden Maschine induziert.

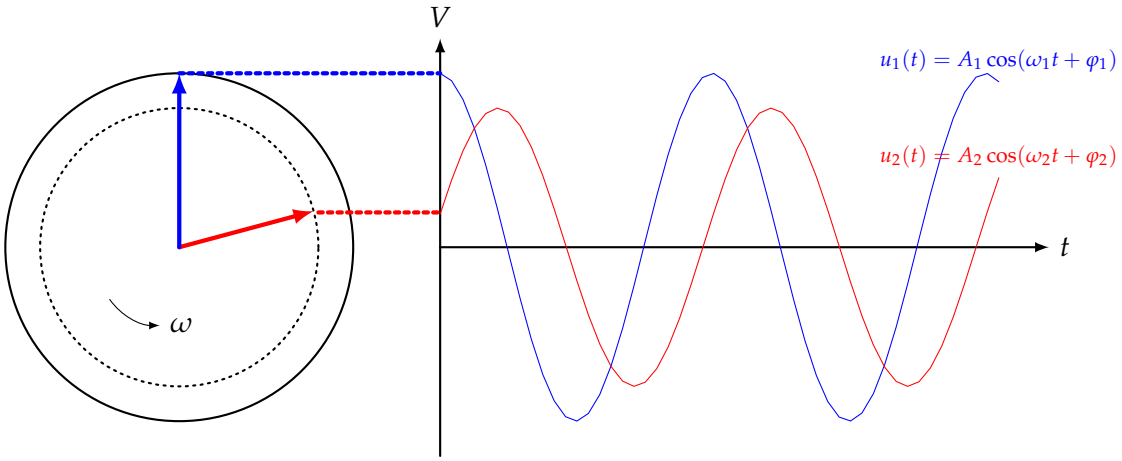


Abbildung 2.1: Kontinuierlich-sinusförmige Signale

Wie in der Abbildung 2.1 zu sehen ist, kann eine Wechselspannung bzw. ein Wechselstrom mit dem winkelförmigen Zeiger erzeugt werden, wobei so ein Signal auch als ein kontinuierlich-sinusförmiges Signal bezeichnet werden. Als allgemeine Form wird eine Wechselspannung bzw. ein kontinuierlich-sinusförmiges Signal mathematisch

$$u(t) = A\cos(\omega t + \varphi) \tag{2.1}$$

mit den folgenden Parametern beschrieben:

- A: Amplitude der Wechselgröße ($Volt$)
- ω: Kreisfrequenz mit $\omega = 2\pi f$ ($Radius/sek.$)
- f: Frequenz (T^{-1} Hz)
- T: Period ($1/f$ sek.)
- φ: Phase ($Radius$)

In der Signalverarbeitung können die Signale mit verschiedenen Operationen verarbeitet werden. Zum Beispiel können zwei oder mehrere Signale addiert bzw. multipliziert werden. Außerdem können auch die Parameter dieser Signale geändert werden.

Es sind zwei verschiedene Lösungswege möglich. Entweder kann die Lösung mittels geometrischer oder mittels algebraischer Methode gefunden werden, wobei die geometrische Methode der schwierigere Ansatz ist. Der algebraische Ansatz wird mit Hilfe der eulerschen Formel[1] analysiert.

Die trigonometrische Funktionen werden als imaginäre Exponentionalfunktionen dargestellt, wie im Folgenden beschrieben wird:

$$\cos(\omega t + \varphi) = \frac{e^{j(\omega t+\varphi)} + e^{-j(\omega t+\varphi)}}{2} \tag{2.2}$$

$$\sin(\omega t + \varphi) = \frac{e^{j(\omega t+\varphi)} - e^{-j(\omega t+\varphi)}}{2j} \tag{2.3}$$

In dieser Arbeit wird ein kontinuierlich-sinusförmiges Eingangssignal mit Hilfe eines Optokopplers als kontinuierlich-rechteckförmiges Ausgangssignal repräsentiert bzw. umgeformt,

[1] besagt, dass eine trigonometrische Funktion als Exponentialfunktion auch gebildet werden kann:
$Ae^{j(\omega t+\varphi)} = A\cos(\omega t + \varphi) + jA\sin(\omega t + \varphi)$

weil die fallenden und/oder steigenden Flanken des Signals aufzählbar sind. Danach wird dieses kontinuierlich-rechteckförmige Signal an den ATMega16-Mikrocontroller geleitet, um die Zeitdifferenzen zwischen zwei aufeinanderfolgenden Flanken zu messen.

2.2 Kommunikationsprotokolle

Heutzutage kommuniziert ein Großteil der Menschen über Ethernet oder über Wireless-Communication miteinander. Als Netzwerk wird im einfachsten Fall die Verbindung dreier oder mehrerer Computer über ihre Ethernetschnittstellen bezeichnet. Um den Datenaustausch gegenseitig zu gewährleisten oder um die gemeinsamen Resourcen und Dienste zu nutzen, wird eine Zustimmung bzw. ein Netzwerkprotokoll benötigt, zu welchem Zeitpunkt oder in welcher Reihenfolge welcher Vorgang durch wen oder was veranlasst wird. Grundsätzlich ist ein Protokoll eine Vereinbarung zwischen den kommunizierenden Hosts [TW10]. Diese Zustimmung besteht aus einem Satz von Regeln für den Datenaustausch zwischen Sender und Empfänger. Ein Verstoß gegen das Protokoll macht die Kommunikation schwieriger, wenn nicht sogar unmöglich.

Die Regeln des Protokolls deckt die folgenden Eigenschaften:

- **Syntax**: Datenformat und Datenkodierung gültiger Nachricht.
- **Semantik**: Zeichengestaltung der Nachricht und deren Bedeutung.
- **Timing**: Geschwindigkeitsabstimmung und Reihenfolgeplanung.

Einige Teile von mehreren Aufgaben eines Netzwerkprotokolls sind wie folgt beschrieben:

- Sicherer und zuverlässiger Verbindungsaufbau,
- Adressierung,
- Verlässliche Paketzustellung,
- Sicherstellen einer fehlerfreien Übertragung (Prüfsumme),
- Abbau der Nachricht (Kommunikation).

2.2.1 Referenzmodelle

Als ein Referenzmodell wird ein Modellmuster bezeichnet, mit dem andere Modelle verglichen oder davon abgeleitet werden können. Ein Netzwerkmodell stellt eine gemeinsame Struktur oder Protokoll, um die Kommunikation zwischen den Systemen zu gewährleisten. Es gibt zwei Referenzmodelle, die einen Rahmen für die Netzwerkkommunikation bereit stellen:

- OSI-Schichtenmodell (auch OSI/ISO beschreibbar)
- TCP/IP-Schichtenmodell

Netzwerkprotokolle werden üblicherweise in Schichten (*layer*) bzw. in Ebenen entwickelt, wobei jede für eine andere Facette der Kommunikationsschicht verantwortlich ist. Jede Schicht bietet einen bestimmten Dienst, wenn Daten zwischen kooperierenden Anwendungen über ein dazwischenliegendes Netzwerk übertragen werden. Vorteil vom Schichtmodell ist, es wird ein Schichtaufbau ermöglicht, um verschiedene Teile der Struktur getrennt zu entwickeln. Um es vielleicht verständlicher zu erläutern, könnte für das Modell ein Einliniensystem herangezogen werden. Datenpakete werden nur von einer Schicht zur benachbarten Schicht weitergeleitet, bis sie die letzte Schicht erreicht haben. Die Aufgabe jeder Schicht ist es, bestimmte Dienstleistungen an die höheren Schichten anzubieten, während dessen alle anderen Details bzw. Schichten keinen Einfluss auf den Verlauf haben.

2.2.2 OSI-Schichtenmodell

Das OSI-Schichtenmodell wurde erstmals in den frühen 1970er Jahren festgelegt und im Jahr 1984 von der ISO standardisiert. Seit mehr als 20 Jahren wird dieses Modell als eine Referenz für traditionelle oder moderne Netzwerkprotokolle verwendet.

Das OSI-Schichtenmodell besteht aus sieben Schichten, die nur mit den benachbarten Schichten kommunizieren. Jede Schicht bietet spezifische Dienstleistungen an und leitet diese an die benachbarte Schicht weiter. Jede Schicht, die die Nachricht bekommt, verarbeitet die Daten und fügt einen *Header* bzw. Datenrahmen am Anfang hinzu (**Data Encapsulation**). Dies ist zwar für den Sender irrelevant, aber für den Empfänger von großer Bedeutung. Auf der Empfängerseite werden die zugehörigen Header in jeder Schicht subtrahiert (**Data De-encapsulation**). Ein grafisch vereinfacht dargestelltes Szenario ist in der Abbildung 2.2 zu erkennen. Die *vertikale Kommunikation* beschreibt den Vorgang, bei dem die Daten die Schichten des verwendeten Referenzmodells durchlaufen. Bei *horizontaler Kommunikation* verwenden Sender und Empfänger jeweils die gleichen Protokollfunktionen auf den gleichen Schichten. Die Nachricht durchläuft auf der Senderseite alle Schichten von der *obersten* bis zu *untersten* Schicht. Hingegen ist der Durchlauf der Nachricht auf der Empfängerseite im umgekehrter Reihenfolge.

Die Aufgaben der Schichten und deren Beschreibungen werden wie folgt aufgezählt:

Physikalische Schicht (Physical Layer)

Es handelt sich bei dieser Schicht um die **Bitübertragung**. Sie ist die Schicht, die mit der physikalischen Hardware interagiert und definiert die physikalischen Eigenschaften des Netzes, wie Verbindungen, Spannungspegel und Timing, etc. Einige Aufgaben sind

- Definition von Hardware-Spezifikationen
- Kodierung und Signalisierung
- Datenübertragung und Datenempfang

- Topologie und physikalisches Netzwerkdesign

Beispiel: Kupferkabel, Glasfaser, Richtfunk, Signalform und Frequenzen im Medium, Ethernet, Token-Ring etc.

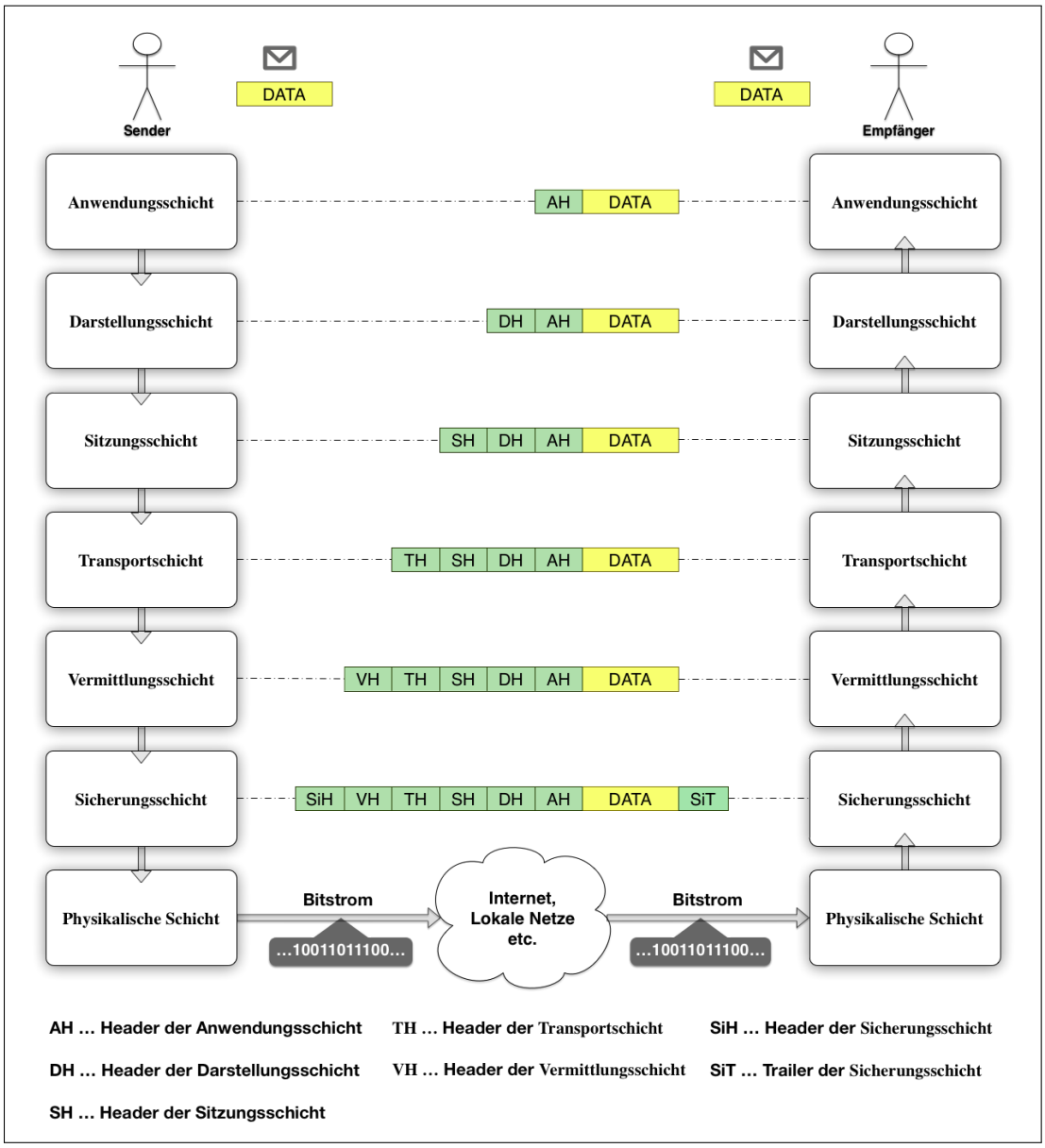


Abbildung 2.2: OSI-Schichtenmodell

Sicherungsschicht (Data Link Layer)

Diese Schicht kann als **Datenverbindungsschicht** bezeichnet werden. Sie ist zuständig für eine zuverlässige Übertragung der Daten über die physische Verbindungen. Ihre Aufgaben sind

- Logical Link Control (LLC)

- Media Access Control (MAC)
- Datengestaltung
- Adressierung
- Fehlererkennung und Fehlerbehebung
- Physikalische Layerstandardisierung

Beispiel: Hardwareadressen, Ethernet, ARP etc.

Vermittlungsschicht (Network Layer)

Die Vermittlungsschicht hat die Aufgabe die **Verbindung aufzubauen**. Diese Schicht bestimmt, wie die Daten an das Empfängergerät gesendet werden sollen. Die Aufgaben sind

- Logische Adressierung
- Routing
- Datagramm-Kapselung
- Fehlerbehandlung und Diagnose

Beispiel: Quality of Service (QoS), Routing, IP-Protokoll, ICMP etc.

Transportschicht (Transport Layer)

Diese Schicht beschreibt die **Sicherungsmechanismen** für einen zuverlässigen Datentransport und garantiert die fehlerfreie Übertragung durch Fehlererkennungs- und Korrekturverfahren. Ihre wichtigsten Aufgaben sind

- Adressierung auf der Prozess-Stufe
- Multiplexen und Demultiplexen
- Segmentierung, Verpackung, und Zusammenbau
- Verbindungsaufbau, Management, und Kündigung
- Anerkennung und Weiterleitung
- Flusskontrolle

Beispiel: TCP- und UDP-Protokoll.

Sitzungsschicht (Session Layer)

Die Sitzungsschicht könnte als **Kommunikationssteuerschicht** bezeichnet werden. Sie beschäftigt sich mit der Verwaltung der Verbindungen zwischen den Anwendungen. Sie stellt ihre Werkzeugsätze bzw. die Befehlssätze als bidirektionale Kommunikation für höhere Protokolle, Anwendungsprogramme oder für die APIs zur Verfügung. Ihre bekanntesten Aufgaben sind

- Datenflusssteuerung
- Dialogkontrolle und Koordination
- Datenzwischenspeicherung

Beispiel: SMB-Protokoll, AppleTalk etc.

Darstellungsschicht (Presentation Layer)

Die Darstellungsschicht übernimmt die Daten von der Anwendungsschicht und wandelt sie in das Standardformat um (**Datenkonvertierungen**), um diese Daten für die darunterliegenden Schichten bereitzustellen. Ihre wichtigsten Aufgaben sind standardisierte

- Übersetzungsverfahren
- Komprimierungsverfahren
- Kodierungsverfahren

Beispiel: Standardisierung wie MPEG, TIFF, GIF und ASCII.

Anwendungsschicht (Application Layer)

Die Anwendungsschicht ist die oberste des OSI-Schichtenmodells. Sie bietet eine Schnittstelle für Benutzer oder für die Betriebssysteme, um Daten zu übertragen bzw. zu empfangen.

Beispiel: Server-Client-Anwendungen, HTTP- und FTP-Client, Mail-Service, Kontroll- und User-Interfaces etc.

2.2.3 TCP/IP-Schichtenmodell

Das zweite wichtige Referenzmodell ist **TCP/IP-Schichtenmodell**, das auf Bedürfnisse der *TCP/IP-Protokollfamilie* zugeschnitten ist. Anders als das OSI- ist das TCP/IP-Schichtenmodell nicht als Standard anerkannt. Dieses Modell wurde ab 1970 vom *Department of Defense (DoD)* in den USA experimentiert. Später wurde das TCP/IP-Referenzmodell im Rahmen des *ARPANET* entwickelt. Im Vergleich zu dem OSI-Schichtenmodell werden bei dem TCP/IP-Schichtenmodell nur vier übereinander liegende Schichten vorgesehen, wie in der Abbildung 2.3 dargestellt wird. Jede Schicht hat mehrere Aufgaben und fügt einige zusätzliche Informationen als *Header* in die

gesendete Nachricht ein (**Einkapselung**). Einige Protokolle (z.B. Ethernet) fügen in der Netzzugangsschicht nicht nur einen Header, sondern auch einen *Trailer* am Ende der gesendeten Nachricht hinzu. TCP/IP ist die heutige Netzwerkprotokollfamilie.

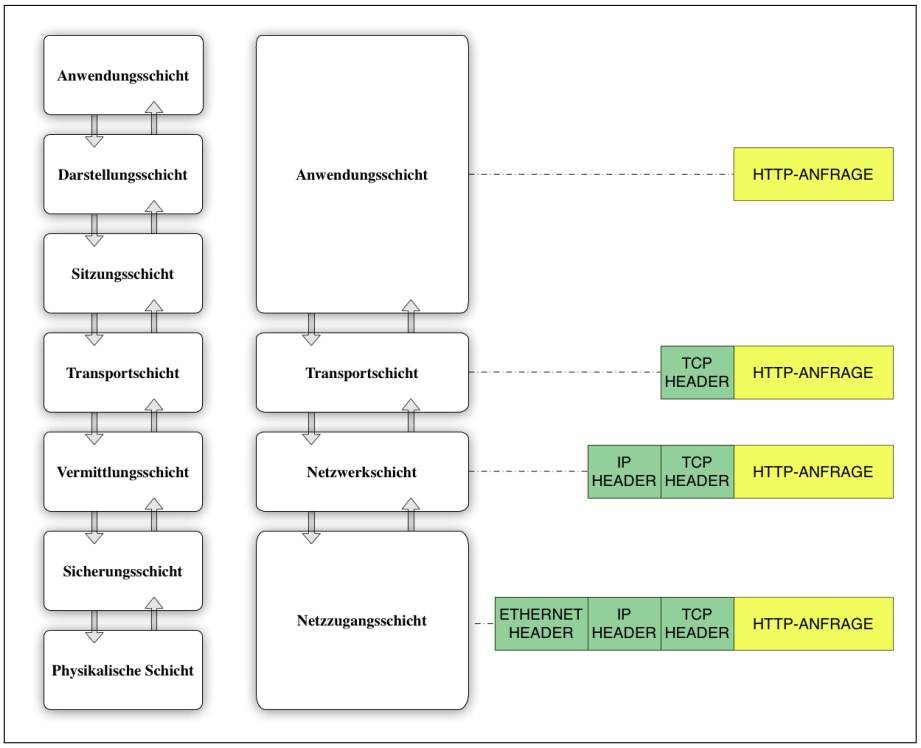


Abbildung 2.3: Vergleich zwischen OSI- und TCP/IP-Schichtenmodell [Jon02] - durch Autor verändert.

Das TCP/IP-Referenzmodell wird in der Literatur als ein 4-Schichtenmodell dargestellt. Andrew S. Tanenbaum hat in seinem Buch "Computer Networks" [TW10] ein *hybrides Referenzmodell* dargestellt, welches nicht als 4- sondern 5-Schichtenmodell präsentiert wird. In diesem Modell wird die Netzzugangsschicht in zwei Schichten aufgeteillt, die als *Sicherungsschicht* und *Bitübertragungsschicht* bezeichnet werden.

Die kurze Beschreibung jeder Schicht und dessen Aufgaben werden im Folgenden beschrieben. Die Abbildung 2.4 zeigt, wie beliebige Musterprotokolle in ihren jeweiligen Schichten definiert werden.

Netzzugangsschicht (Data Link Layer)

Diese Schicht ist die Zusammenlegung der Aufgaben der physikalischen Schicht und der Sicherungsschicht aus dem OSI-Schichtenmodell. Beide Schichten werden in der Netzzugangsschicht vereint. Sie beschäftigt sich mit den Eigenschaften der verschiedenen Übertragungsmedien. Weiters hat sie die Aufgaben von Zugriffsverfahren und zuverlässiger Übertragung der Nachricht über die Übertragungsmedien.

Netzwerkschicht (Network Layer)

Die Aufgaben der Netzwerksschicht sind Aufbau der Verbindung und die Weitervermittlung von Daten in einem logischen Netz. Dies bedeutet, dass das Internet-Protokoll für die Adressierung und Versendung der Datenpakete verantwortlich ist.

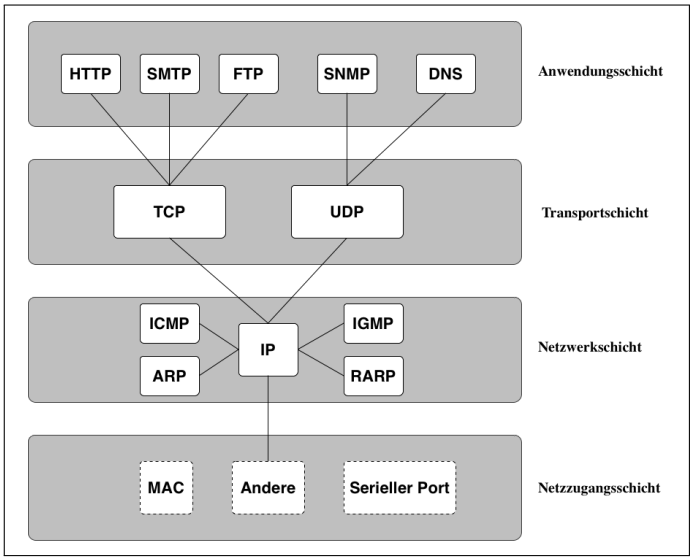


Abbildung 2.4: Musterprotokolle in ihren jeweiligen Schichten [Jon02]
- durch Autor verändert.

Transportschicht (Transport Layer)

Die Transportschicht besteht aus zwei Protokollen, nämlich TCP und UDP. TCP ist ein verbindungsloses Protokoll und stellt Host-zu-Host-Datendienste zur Verfügung.

Anwendungsschicht (Application Layer)

Die Anwendungsschicht enthält alle möglichen Protokolle, welche von Anwendungsprogrammen, von Betriebssystemen oder von Prozessen um auf das Netzwerk zugreifen zu können, verwendet wird.

2.2.4 Internet-Protokolle

Jede Schicht des OSI- bzw. des TCP/IP-Schichtenmodells hat die Aufgabe, interne Dienste für die benachbarten Schichten anzubieten. Diese Aufgaben sind durch Protokolle geregelt. Die Abbildung 2.5 zeigt an, welche Schicht aus welchen Protokollen besteht.

Wie schon angeschnitten wurde, findet die Übertragung über das Ethernet-Kanal statt, welches in der Netzzugangsschich liegt. In der darüber liegenden Schicht (Netzwerkschicht) ist das Internet-Protokoll für den Verbindungsaufbau, Adressierung und Vermittlung zwischen den

kommunizierenden Hosts verantwortlich. Das ARP befindet sich auch in dieser Schicht. Außerdem ist hier auch das ICMP platziert, welches für Statusinformationen des Zielhosts mittels des Befehls `ping` [2] abfragt.

<table>
<tr><td colspan="10">Internet-Protokolle</td></tr>
<tr><td colspan="2">OSI-Schicht</td><td colspan="7">Internet Protokoll Suite</td><td>DOD Schicht</td></tr>
<tr><td>7</td><td>Anwendung</td><td>File Transfer</td><td>Electronic Mail</td><td>Terminal Emulation</td><td>Usenet News</td><td>World Wide Web</td><td>Domain Name Service</td><td>Art der Kommunikation</td></tr>
<tr><td>6</td><td>Darstellung</td><td rowspan="2">File Transfer Protocol (FTP)</td><td rowspan="2">Simple Mail Transfer Protocol (SMTP)</td><td rowspan="2">Telnet Protocol (Telnet)</td><td rowspan="2">Usenet News Transfer Protocol (NNTP)</td><td rowspan="2">Hypertext Transfer Protocol (HTTP)</td><td rowspan="2">Domain Name Service (DNS)</td><td rowspan="2">Applikation</td></tr>
<tr><td>5</td><td>Sitzung</td></tr>
<tr><td>4</td><td>Transport</td><td colspan="5">Transmission Control Protocol (TCP)</td><td>User Datagram Protocol (UDP)</td><td>Host-zu-Host Kommunikation</td></tr>
<tr><td>3</td><td>Vermittlung</td><td>Address Resolution Protocol (ARP)</td><td colspan="4">Internet Protocol (IP)</td><td>Internet Control Messsage Protocol</td><td>Internet</td></tr>
<tr><td>2</td><td>Sicherung</td><td colspan="2">Ethernet</td><td>Token Ring</td><td>DQDB</td><td>FDDI</td><td>ATM</td><td>lokales Netzwerk</td></tr>
<tr><td>1</td><td>Physikalische Übertragung</td><td>Twisted Pair</td><td>Lichtwellenleiter</td><td colspan="2">Coaxkabel</td><td>Funk</td><td>Laser</td><td>Netzzugriff</td></tr>
</table>

Abbildung 2.5: Internet-Protokolle [22]

Das TCP ist in der vierten OSI-Schicht und verantwortlich für einen zuverlässigen Datentransport. Nebenan ist das UDP für eine kurze Netzwerkmeldung zuständig, welches in dieser Arbeit realisiert wurde.

2.2.4.1 Ethernet

Das Ethernet ist in den 1970er Jahren am *Xerox Palo Alto Research Center* (PARC) entworfen worden. Die erste Version des Ethernets wurde ab 1980 vom IEEE in der Arbeitsgruppe 802 weiterentwickelt. Im Jahr 1982 wurde eine neue Version von Ethernet, nämlich **Ethernet-II**, herausgegeben, welche in diser Arbeit verwendet wird. Später wurde es durch IEEE 802.3 standardisiert und weiterentwickelt, um wieder zu TCP/IP kompatibel zu werden. In der Abbildung 2.6 wird der ursprüngliche Ethernet-Entwurf von Robert M. Metcalfe die Netzwerkverbindung als Äther definiert.

Unter dem Standard-Ethernet wurde die spezifizierte Datenübertragungsrate von maximal 10 MBit/s realisiert. Weiters wurde Fast-Ethernet bis zu 100 MBit/s sowie auch das Gigabit-Ethernet bis zu 1000 MBit/s weiterentwickelt.

[2] ist ein klassisches Werkzeug für die Verbindungsprüfung und wurde von Mike Muuss erst im Jahr 1983 in 4.3BSD entwickelt.

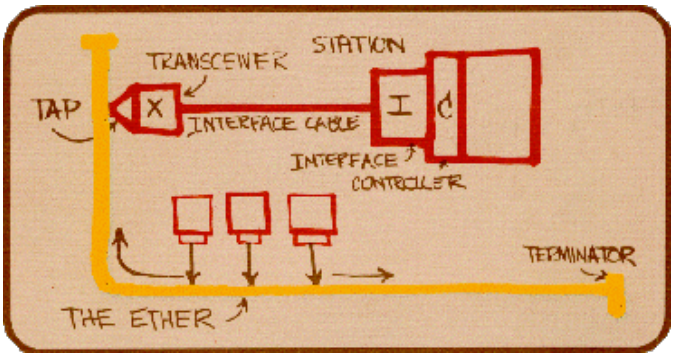


Abbildung 2.6: ursprüngliche Ethernet-Entwurf von Robert M. Metcalfe [8]

Es wird in dem Abschnitt 4.5 erwähnt, dass der ENC28J60-Mikrochip eine Schnittstelle vom Standart-Ethernet hat. Aus diesem Grund findet ein Standard-Ethernet in dieser Arbeit Verwendung. Es muss allerdings beachtet werden, dass ein Kabelsegment beim Standard-Ethernet eine Länge von 100 m nicht überschreiten darf.

Ethernet-II

Die gesendeten Daten bzw. Nachrichten werden immer mit Hilfe der Ethernet-Pakete transportiert. Die Ethernet-Pakete werden spezifisch als **frame** bezeichnet. Der Grundaufbau eines Ethernet-Frames ist bei allen Ethernet-Implementierungen identisch. Das **Typfeld** ist ein charakteristische Merkmal von Ethernet-II, das aus zwei Bytes im Anschluß an die Start- und Zieladressen besteht. Außerdem verfügt es über ein Kennzeichen, welches mittels eindeutiger zwei Byte-Zahl dargestellt wird, um das verwendete Protokoll bekanntzugeben. Es wird lediglich Ethernet-II-Frame in dieser Arbeit eingesetzt. Der Aufbau eines Ethernet-Datenpakets wird in der Abbildung 2.7 dargestellt.

Präambel	SFD	Ethernet-Frame: min. 64 Byte / max. 1518 Byte					Inter Frame Gap 9,6 µs
...11010010111...		Zieladresse	Quelladresse	Typ	Datenfeld	FCS	
8 Bytes		6 Bytes	6 Bytes	2 Bytes	46-1500 Bytes	4 Bytes	

Abbildung 2.7: Frame-Struktur von Ethernet-II [22]
- durch Autor verändert.

Dem Ethernet-Frame wird eine **Präambel** vorangestellt. Sie besteht aus beliebigen Bit-Kombinationen aus Nullen und Einsen in sieben Bytes, die für die Synchronisierung mit den Kommunikationspartnern benötigt werden. Der Präambel folgt **SFD** und dient als Rahmenbegrenzung. SFD führt eine eindeutige Bit-Kombination zur Signalisierung des Anfangs des Datenpakets.

Die **Ziel-** und **Quelladresse** besteht aus ihren MAC-Adressen von jeweils 6 Bytes. Für jede Netzwerkkarte gibt es eine eindeutige MAC-Adresse.

Das **Typfeld** dient als Kennzeichnung für den jeweiligen Protokolltyp. Es kann vorkommen, dass die Kennzeichnung keiner bekannten Type verweist. Genau in diesem Fall wird die Größe des Datenfeldes mittels dieser Zahl bekanntgegeben, da es keine Verpflichtung gibt, ein standardisiertes Protokoll zu verwenden. Einige Kennzeichen für zugehörige Protokolle sind in der folgenden Tabelle 2.1 beschrieben. Eine komplette öffentliche Liste befindet sich auf der Webseite von "IANA" [19].

Kennzeichen	Protokoll
0x0800	IPv4
0x0806	ARP
0x8138	Novell
0x86dd	IPv6
0x809b	Appletalk

Tabelle 2.1: Kennzeichen für einige Protokolle

Das **Datenfeld** besteht aus einigen Informationen, wie beispielsweise der TCP/IP-Header und beinhaltet gesendete Benutzerdaten. Die Länge des Datenfelds zum Datentransport beträgt zwischen 64 Bytes und 1518 Bytes.

Von der Zieladresse bis zum Ende des Datenfeldes wird ein **CRC** durchgeführt, die eine 32-Bit-Prüfsumme ist. Die Präambel und der SFD sind nicht in der Prüfsumme enthalten. Sie dient der Fehlererkennung durch die Anwendung eines speziellen Generatorpolynoms. Anschließend wird der CRC-Wert in das **FCS-Feld** des Ethernet-II-Frames gespeichert. Der Empfänger errechnet auch die Prüfsumme aus den empfangenen Daten und vergleicht diese mit den empfangenen FCS-Bytes. Nach dem Senden eines Frames erfolgt eine kurze Pause von 9,6 μs, die als **Inter Frame GAP** bezeichnet wird.

2.2.4.2 MAC-Adresse

Jede Netzwerkkarte hat eine eindeutige **MAC-Adresse** und wird auch als *physikalische Adresse* bezeichnet, beispielsweise "00:23:32:cc:ff:1a". Ihre Aufgabe ist, die miteinander kommunizierenden Hosts zu identifizieren. Die MAC-Adresse besteht aus einer Reihenfolge mit sechs Bytes (also 48 Bits[3]) und wird mit dem hexadezimalen Zahlensystem dargestellt. In UNIX basierten Systemen erscheinen die MAC-Adresse der Netzwerkgeräte sowie Ethernet oder WiFi mit dem Befehl `ifconfig` [4] und können manuell geändert werden. Das Format der MAC-Adresse ist in der Abbildung 2.8 zu sehen.

- **I/G = 0:** Individual-Adresse (Unicast Address), das Ziel ist ein einzelnes Gerät bzw. ein Host

[3] ermöglicht insgesamt 2^{48} MAC-Adressen.

[4] In manchen GNU/Linux-Systemen befindet sich kein Befehl `ifconfig` , sondern es gibt eigene Befehle sowie `ip` bzw. `netstat -ia` oder eigene vorhandene Systemdatei unter dem "/etc"-Verzeichnis, in der die Netzwerkspezifikationen dargestellt werden.

I/G	U/L	OUI	OUA
1 Bit	1 Bit	22 Bits	24 Bits

Abbildung 2.8: Format der MAC-Adress [22]

- **I/G = 1:** Gruppen-Adresse (Multicast Address), das Ziel ist eine Gruppe im LAN (Multicast- oder Broadcast-Adresse)
- **U/L = 0:** universelle eindeutige und veränderbare Adresse bzw. MAC-Adresse
- **U/L = 1:** lokale veränderbare Adresse bzw. logische Adresse
- **OUI (Organizationally Unique Identifier):** Herstellererkennung, die von IEEE vergeben wird
- **OUA (Organizationally Unique Address):** Private Kennzeichnung, die vom Hersteller vergeben wird

2.2.4.3 Adressierung

Um die Daten zwischen zwei kommunizierenden Hosts zu übertragen, muss eine eindeutige Adresse indentifiziert werden, damit die gesendeten Daten an den richtigen Zielhost zugestellt wird. Aus diesem Grund hat jeder Rechner, der an einen TCP/IP-Netzwerk angeschlossen ist, eine bestimmte Adresse, nämlich eine **IP-Adresse**.

Die IP-Adressen bestehen aus einer 32 Bit langen Zahl und werden in Form von vier Dezimalzahlen geschrieben, jeweils zu 8 Bit gruppiert und durch einen Punkt (.) getrennt, beispielsweise "128.130.40.232". Daher können maximal $2^{32} = 4.294.967.296$ Adressen dargestellt werden. Ursprünglich wurden die IP-Adressen in drei Klassen, A, B und C, unterteilt. Jede Klasse besteht aus den Teilen von Netzadresse (Network Identifier) und Hostadresse (Host Identifier). Um die IP-Adresse in die Netzadresse und Hostadresse zu zerlegen, wird eine **Netzmask** verwendet.

Klasse A

In dieser Klasse gibt es 8 Bits, davon 1 Bit für Präfix und 7 freie Bit für Netzadresse. Die restlichen 24 Bit sind für die Hostadresse verwendbar. Das heißt, dass es sich in Klasse-A-Netzen maximal $2^7 = 128$ Netze und jeweils maximal $2^{24} = 16.777.216$ Hostadressen befinden können.

Klasse B

In der Klasse-B gibt es 16 Bits, davon 2 Bit für Präfix und 14 freie Bit für die Netzadresse. Die übrigen 16 Bit werden wieder für die Hostadresse verwendet. Es gibt maximal $2^{14} = 16.384$ Netzadresse mit jeweils höchstens $2^{16} = 65.536$ Hostadressen.

Klasse C

In Klasse-C-Netzen gibt es 24 Bit, davon 3 Bits für Präfix und 21 freie Bit für die Netzadresse. 8 Bits finden wie andere Klassen für die Hostadresse Verwendung. Es gibt maximal $2^{21} = 2.097.152$ Netzadressen mit jeweils höchstens $2^8 = 256$ Hostsadresse.

Klasse	Präfix	Adressbereich
A	0	0.0.0.0 - 127.255.255.255
B	10	128.0.0.0 - 191.255.255.255
C	110	192.0.0.0 - 223.255.255.255

Tabelle 2.2: Präfixe und Adressbereiche der Netzklassen [Bau13]

In der folgenden Tabelle 2.3 ist ein Beispiel zur Klasse-C-Netzen mit der IP-Adresse 192.168.0.3 zu erkennen. Die UND-Verknüpfung der IP-Adresse mit der Netzmask ergibt die Netzadresse. Wiederum die UND-Verknüpfung zwischen der IP-Adresse und der invertierten Netzmask ergibt sich die Hostadresse.

IP-Adresse	192.168.0.3	11000000	10101000	00000000	00000011
Netzmask	255.255.255.0	11111111	11111111	11111111	00000000
inv. Netzmask	0.0.0.255	00000000	00000000	00000000	11111111
Netzadresse	192.168.0.0	11000000	10101000	00000000	00000000
Hostadresse	0.0.0.3	00000000	00000000	00000000	00000011

Tabelle 2.3: Beispiel im Klasse-C

2.2.4.4 ARP

Wenn ein Datenpaket aus den oberen Schichten angekommen ist, muss es an die MAC-Adresse des Zielhosts adressiert werden. Das heißt, dass die MAC-Adresse des Zielhosts bekannt sein muss, damit die gesendeten Datenpakete beim Host zugestellt werden können. Aus diesem Grund hat ARP die Aufgabe, die IP-Adresse der Vermittlungsschicht in Hardware- und in MAC-Adressen der Sicherungsschicht umzusetzen. An dieser Stelle stellt ARP eine Tabelle, in der die logischen und physikalischen Adressen von lokalen Netzwerkgeräten gespeichert werden. Die ARP-Tabelle enthält drei Spalten, die aus IP-, MAC-Adresse und Typ bestehen. In UNIX basierten Systemen können die IP- und MAC-Adressen vom den beteiligten Rechnern im lokalen Netzwerk mit dem Befehl `arp -a` angezeigt werden. Die Abbildung 2.9 zeigt, welche Informationen das ARP-Protokoll beinhaltet.

Um die physikalischen Adressen der lokalen Rechnern zu ermitteln, wird eine ARP-Anforderung (ARP Request Broadcast) mit der Broadcast-Adresse von "**FF:FF:FF:FF:FF:FF**" gesendet. Danach aktualisiert jeder Empfänger seine Tabelle und sendet seine IP- und MAC-Adresse in die Quell-adresse, welche in der Quell-Hardwareadresse der gesendeten ARP-Anforderung verfügbar ist, zurück. Diese Beantwortung der ARP-Anforderung heißt ARP Reply.

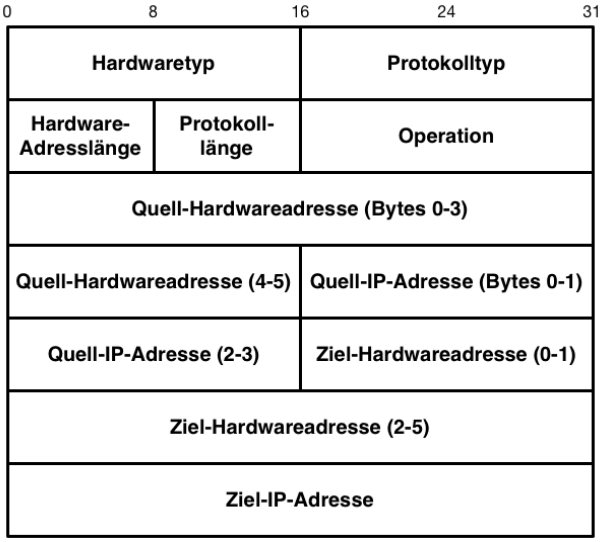


Abbildung 2.9: ARP-Protokoll [22]

2.2.4.5 Internet Protokoll

In der Vermittlungsschicht des OSI-Schichtenmodells und auch in der Netzwerkschicht des TCP/IP-Schichtenmodells befindet sich das Internet-Protokoll (kurz **IP**), um einen grundlegenden Netzdienst zur Verfügung zu stellen. IP hat auch ein eigenes Paketformat, das als **Datagramm** bezeichnet wird, bestehend aus dem Datenpaket und seinem eigenen Header. IP-Header ist der Bereich vom Anfangsbit bis zum Anfang des Datenbereichs. Die Abbildung 2.10 zeigt an, wie das Format von IP-Version-4 (IPv4) beschrieben wird.

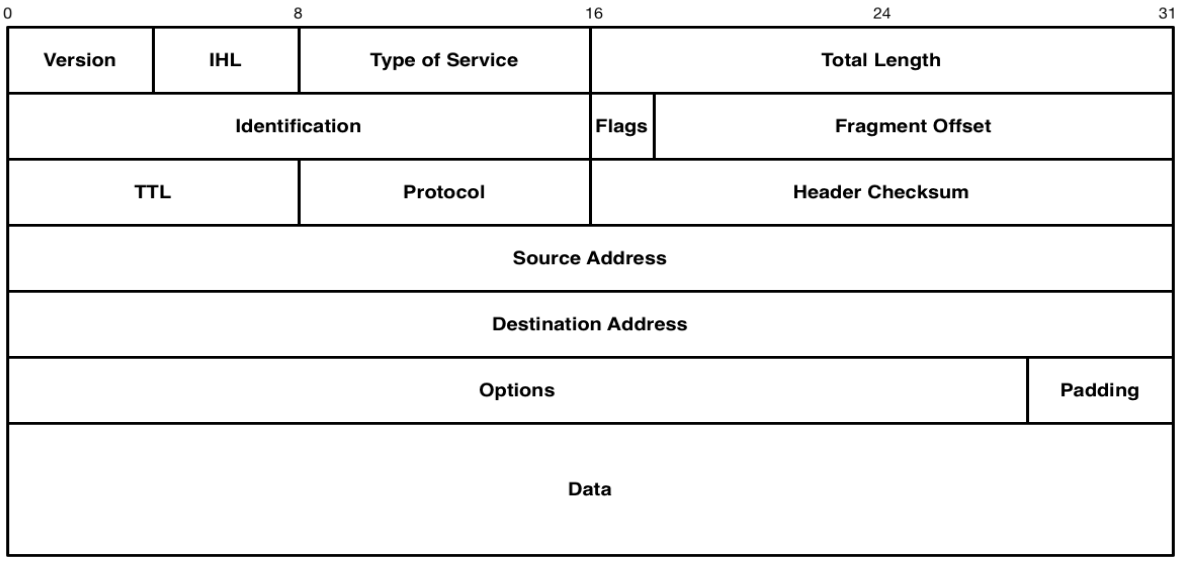


Abbildung 2.10: Internet Header Format [24]

Die Felder des Internet-Protokolls werden im Folgenden beschrieben:

- **Version:** Kennzeichnung von verwendeten Internet-Protokoll-Version[5] (4 Bits)
- **IHL:** Länge des Internet-Protokoll-Headers (4 Bits)
- **Type of Service:** Steuerinformation sowie Priorisierung des Datagramms (8 Bits)[6]
- **Total Length:** Länge des gesammten Datagramms (inkl. Kopfdaten, 16 Bits) ergibt eine maximale Länge von 64 KB[7]
- **Identification:** Eindeutige Kennung eines Datagramms (16 Bits)
- **Flags:** Die Flags haben folgende Bedeutung:
  - **Bit 0:** reserviert (immer 0)
  - **Bit 1:** DF (Don't Fragment), das Paket darf nicht fragmentiert werden, wenn das Bit logisch 1 gesetzt ist
  - **Bit 2:** MF (More Fragments), wenn er logisch 0 ist, ist das Paket die letzte Fragmentierung
- **Fragment Offset:** Anfangsadresse des fragmentierten Datagramms (13 Bits)
- **Time to Live:** Maximale Lebensdauer eines Datagramms (8 Bits)
- **Protocol:** Kennzeichnung des Protokolls (8 Bits)
- **Header Checksum:** Prüfsumme der Kopfdaten (16 Bits)
- **Source Address:** Internet-Adresse des Quellhosts des Datagramms (32 Bits)
- **Destination Address**: Internet-Adresse des Zielhosts des Datagramms (32 Bits)
- **Options:** Optionale Verwendung für weitere Informationen (24 Bits)
- **Padding:** Füllbits, damit unbenutzte Bytes mit Nullen ausgefüllt werden (8 Bits)
- **Data:** Beliebig gesendete bzw. empfangene Nachrichten

[5] Für IPv4 wird 4 verwendet.

[6] Dies Feld wurde in vergangenene Jahren unterschiedlich interpretiert. Es kann zwischen RFC-Nummern von 791, 2474 und 3168 verglichen werden.

[7] 2^{16} Bits = 64 KB.

2.2.4.6 ICMP

Das Internet Control Message Protocol (ICMP) liegt beim Internet-Protokoll in der Vermittlungsschicht und ist ein Bestandteil des IP-Diagramms, wie in der Abbildung 2.11 dargestellt wird. Aus diesem Grund muss ICMP in IP-Pakete bzw. -Datagramme verpackt werden. Die Aufgabe von ICMP ist es, einen Bericht von Fehlermeldungen und Kontrollnachrichten über das Internet-Protokoll zu erstatten.

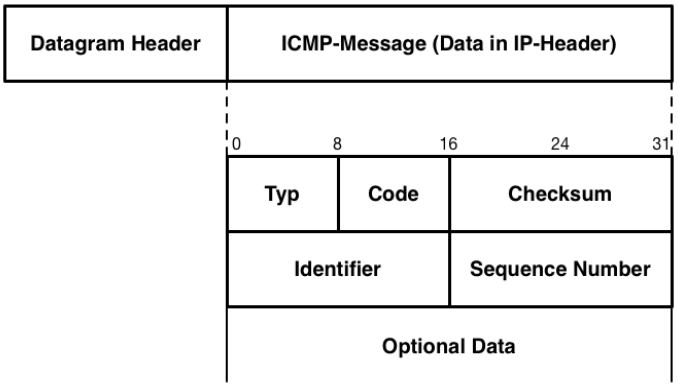


Abbildung 2.11: ICMP-Header [23]
- durch Autor verändert.

Das Typ-Feld gibt an, zu welcher Klasse die ICMP-Nachricht gehört. Anbei spezifiziert das Code-Feld die Art der Nachricht. Checksum ist die Prüfsumme des kompletten Datagramms inklusive Header. Identifier ist ein Wert, um die Datagramme zuzuordnen. Die Sequenznummer unterscheidet die Datagramme. Die folgende Tabelle 2.4 gibt eine genauere Beschreibung von Typ-Code-Kombinationen an.

Typ	Typname	Code	Bedeutung
0	Echo-Antwort	0	Echo-Antwort (Anwort auf `ping` -Anfrage)
3	Ziel nicht erreichbar	0	Netzwerk nicht erreichbar
		1	Host nicht erreichbar
		2	Protokoll nicht erreichbar
		3	Port nicht erreichbar
		4	Fragmentierung benötigt, aber nicht erlaubt (Don't Fragment gesetzt)
		5	Source-Route fehlgeschlagen
		6	Zielnetzwerk unbekannt
		7	Ziel-Host unbekannt
8	`ping` (Echo-Anfrage)	0	Echo-Anfrage
10	Router Selection	0	(Router-Auswahl)
11	Zeitlimit überschritten	0	TTL (Time To Live) abgelaufen
30	Traceroute	0	Weg zum Ziel ermitteln

Tabelle 2.4: Typ-Code-Kombinationen von ICMP

2.2.4.7 TCP

Das Transmission Control Protocol (TCP) wird in der Transportsschicht des OSI-Schichtenmodells lokalisiert und bietet eine verbindungsorientierte, sichere Übertragung zwischen zwei kommunizierenden Hosts, im Prinzip eine Host-zu-Host-Verbindung. Die verbindungsorientierte Übertragung bedeutet, dass eine Kontrollfunktion während der Datenübertragung zwischen den Hosts ausgeführt wird, wodurch es sehr zuverlässig ist, weil die verlorengegangenen Daten erneut versendet werden können; das heißt, dass keine Daten verloren gehen.

Das TCP-Paket wird auch als **Segment** bezeichnet und enthält keine IP-Adresse, sondern Port-Nummer des Senders oder Empfängers, weil die IP-Adressen von kommunizierenden Hosts bereits im IP-Header angegeben werden (siehe Abbildung 2.10). Daher findet die Kommunikation über Ports statt. Die folgende Abbildung 2.12 zeigt die Struktur eines Segments an.

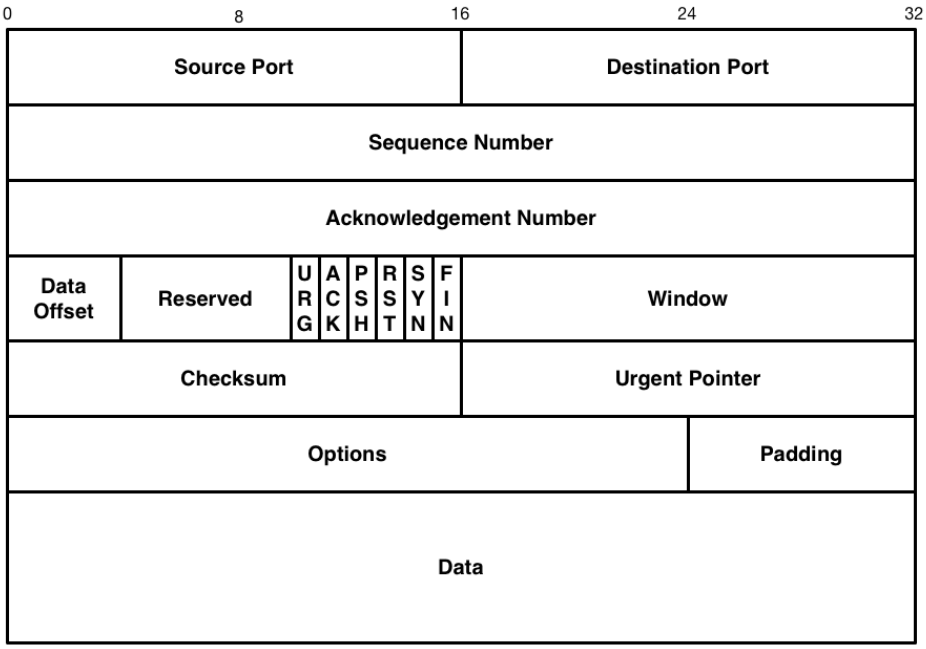


Abbildung 2.12: TCP Header Format [25]

Die Felder des TCP-Headers werden im Folgenden beschrieben:

- **Source Port:** Die Portnummer des Senders besitzt eine Größe von 16 Bits und es stehen $2^{16} = 65536$ Ports zur Verfügung.
- **Destination Port:** Die Portnummer des Empfängers hat die Größe von 16 Bits und ermöglicht also 65536 Ports.
- **Sequence Number:** Die Nummer des aktuellen Segments innerhalb des Datenstromes.
- **Acknowledgement Number :** Die Sequenznummer des nächsten erwarteten Segments.

- **Data Offset:** Ist die Länge des Headers und enthält die Länge des TCP-Headers in 32-Bit Blöcken, damit der Empfänger weiß, wo die Nutzdaten im TCP-Segment anfangen.
- **Reserved:** Dieses Feld ist für zukünftige Entwicklungen reserviert und muss Null sein.
- **Control Bits:** Die folgenden sechs je 1 Bit große Felder werden für den Verbindungsaufbau, Datenaustausch und Verbindungsabbau benötigt. Jedes Bit erfüllt zwei unterschiedliche Funktionen, je nach dem ob es gesetzt ist oder nicht.
  - **URG (Urgent):** Es wird gesetzt, wenn Urgent Pointer einen gültigen Wert enthält.
  - **ACK (Acknowledge):** ACK bestätigt die Gültigkeit der Bestätigungsnummer (ACK Number).
  - **PSH (Push):** PSH weist darauf hin, dass das Segment sofort in den Empfangspuffer übermittelt und direkt zur Anwendung gelangen kann.
  - **RST (Reset):** Die Verbindung wird abgebrochen, wenn der Reset-Bit gesetzt ist.
  - **SYN (Synchronize):** Der SYN-Bit ist gesetzt, während eine Verbindung aufgebaut wird.
  - **FIN (Finish):** Der Sender setzt das FIN-Bit, wenn die Übermittlung beendet ist.
- **Window:** Das Window-Feld hat eine Anzahl der Bytes, die der Sender übermitteln darf, damit ein zuverlässiger Datentransport ohne Überlaufen des Empfangspuffers erledigt werden kann.
- **Checksum:** Die Prüfsumme des Datagramms (inkl. TCP-Header und Data) ist für die Fehlererkennung zuständig.
- **Urgent Pointer:** Wenn das URG-Bit des Control-Bits gesetzt ist, dann bekommt der Urgent-Pointer eine Bedeutung und wird interpretiert. Er gibt die Position des letzten Bytes der Urgent-Daten im Datenstrom an.
- **Options:** Das Option-Feld enthält Zusatzinformationen. TCP kennt drei Optionen, die NOP, End of Option List und die Festlegung der maximalen Segmentgröße.
- **Padding:** Padding wird verwendet, um den Anfang und das Ende des 32-Bit TCP-Headers sicherzustellen, wenn es nötig ist. Das Padding besteht aus Nullen.
- **Data:** Die Nutzdaten, welche für Anwendungen gedacht sind.

2.2.4.8 UDP

Das User Datagram Protocol (UDP) ist ein minimales Protokoll in der Transportschicht des OSI-Schichtenmodells. Im Gegensatz zu TCP ist UDP kein zuverlässiger, verbindungsloser Dienst, weil hier keine Flußkontrolle zur Verfügung gestellt wird. Es gibt keine Garantie, dass ein

gesendetes Datanpaket an der Empfängerseite ankommt. Die versendeten Daten gehen auf gut Glück auf die Reise ins Netz. Der Vorteil bei UDP ist, dass die Größe des UDP-Segments ziemlich klein und die Geschwindigkeit des Datentransports sehr hoch sind. Aus diesem Grund wird in dieser Arbeit ein UDP-Datagramm verwendet, weil die abgefragete Information nur aus einer kommastelligen Zahl besteht. Der Aufbau eines UDP-Datagramms ist ganz einfach, wie in der folgenden Abbildung 2.13 zu sehen ist.

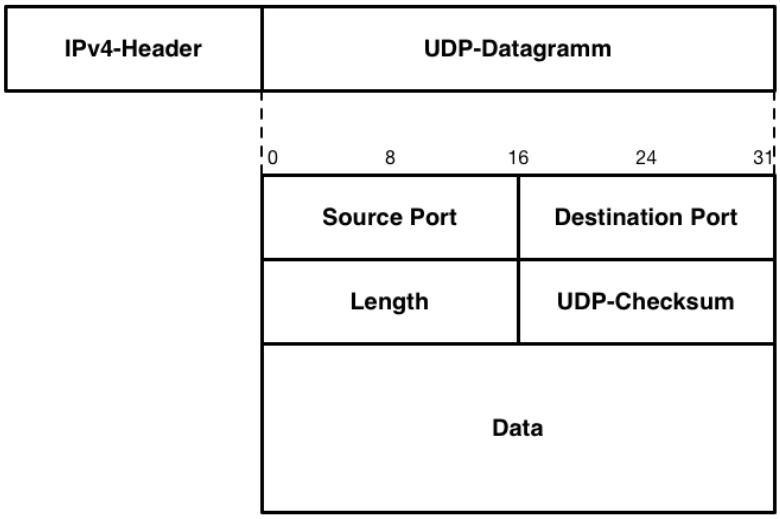


Abbildung 2.13: User Datagram Header Format [SF12] - durch Autor verändert.

Die Felder des UDP-Headers werden im Folgenden beschrieben:

- **Source Port:** Die Portnummer des Senders, welche Werte von 0 bis 65536 annehmen kann.
- **Destination Port:** Die Portnummer des Empfänger.
- **Length:** Die Länge des UDP-Datagramms (inkl. UDP-Header).
- **Checksum:** Prüffsumme wird optional verwendet. Falls es nicht verwendet wird, wird der Inhalt mit Nullen dargestellt.
- **Data:** Benutzerdaten.

2.2.5 RFC-Dokumente

In der Tabelle 2.5 werden einige wichtigste und nützlichste RFC-Dokumente mit ihrer RFC-Nummer dargestellt. Für nützliche Informationen werden zwei unterschiedliche Webseiten angeboten:

- Die ganze Liste der RFC-Dokumente befinden sich auf der Webseite **rfc-editor** [10]
- Häufig gestellte Fragen ist auf der Webseite **FAQs** [7] aufrufbar

RFC-Nummer	Beschreibung
RFC 768	User Datagram Protocol (UDP)
RFC 791	Internet Protocol (IP)
RFC 792	Internet Control Message Protocol (ICMP)
RFC 793	Transmission Control Protocol (TCP)
RFC 821	Simple Mail Transfer Protocol (SMTP)
RFC 826	Ethernet Address Resolution Protocol (ARP)
RFC 854	Telnet Protocol (Telnet)
RFC 959	File Transfer Protocol (FTP)
RFC 977	Usenet News Transfer Protocol (NNTP)
RFC 1034	Domain Name Service (DNS)
RFC 1055	Serial Line Internet Protocol (SLIP)
RFC 1533	DHCP Options
RFC 1541	Dynamic Host Configuration Protocol (DHCP)

Tabelle 2.5: RFC-Dokumente und ihre Beschreibungen

2.3 BSD-Sockets

Der Aufbau eines Netzwerkens wurde im Abschnitt 2.2 theoretisch beschrieben. Dieses Kapitel klärt auf, wie ein Netzwekaufbau mittels Betriebssystemfunktionen, sowie Socket-APIs erstellt werden kann.

Wie schon oben erwähnt wurde, muss ein Kanal zwischen kommunizierenden Hosts erstellt werden, wenn sie Nachrichten gegenseitig austauschen wollen. Dieser Kanal wird als **Socket**, sowie in UNIX-Systemen als **BSD-Socket**[8] bezeichnet. Die Socket-APIs liegen in der Sitzungsschicht des OSI-Schichtenmodells, sowie in der Anwendungsschicht des TCP/IP-Modells. Diese stellt eine Schnittstellt zwischen Anwendungsprogramme und der darunterliegenden Protokolle her. Die Abbildung 2.14 zeigt an, wie die Szenarien für TCP- und UDP-Verbindungen mittels BSD-APIs sind.

2.3.1 Anlegen eines Sockets

Beim Anlegen eines Sockets gibt es die **socket(2)**-Funktion, die im Erfolgsfall einen speziellen Datei- bzw. Socketdeskriptor zurückliefert, oder im Fehlerfall -1. Wenn die socket()-Funktion einen Socket erstellt, dann können die Nachrichten mit Hilfe des Socketdeskriptors ins Socket geschrieben oder aus dem Socket gelesen werden.

```
#include <sys/types.h>
#include <sys/socket.h>

int
socket(int domain, int type, int protocol);
```

[8] wurde ertmal im Jahr 1983 für 4.2BSD auf der Berkeley Universität entwickelt.

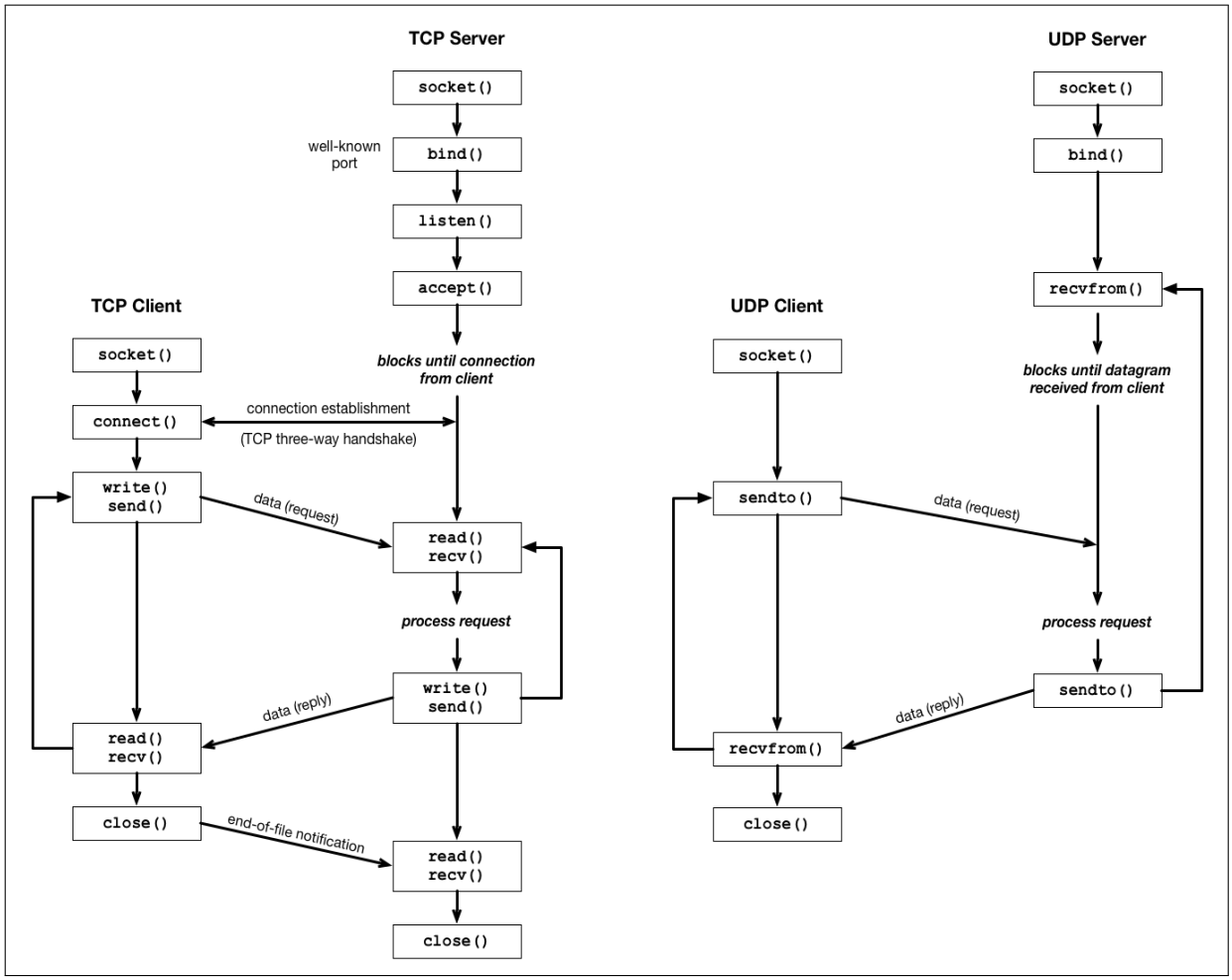


Abbildung 2.14: Szenarien von TCP- und UDP-Kommunikationen [SFR04]

Der Parameter *domain* beschreibt die Art der Kommunikation sowie die gewünschte Adress- oder Protokollfamilie, mit der der neu angelegte Socket assoziiert wird. Diese Familien sind in der <*sys/socket.h*> definiert, welche von POSIX.1 bekannt gegeben werden, wie einige Domains in der Tabelle 2.6 beschrieben werden. **PF_** steht hier für "Protocol Family" und **AF_** für "Address Family". **PF_** und **AF_** haben dieselbe Konstante, weil sie aus historischen Gründen dieselben Implementierungen haben.

Domain	Beschreibung
AF_INET	IPv4 Internet-Domain
AF_UNIX	UNIX-Domain
PF_LOCAL	Interne Host-Protokoll
AF_UNSPEC	nicht spezifiert (unspecified)

Tabelle 2.6: Domäne der socket()-Funktion

Der zweite Parameter der socket()-Funktion *type* setzt das Sockettyp ein, um die Semantik der Kommunikation im Rahmen der zuvor definierten Domains zu bestimmen. Die von POSIX.1 definierten Sockettypen werden in der Tabelle 2.7 dargestellt.

Sockettyp	Beschreibung
SOCK_STREAM	Byte-Stream Socket
SOCK_DGRAM	Datagram Socket
SOCK_RAW	Raw Protocol Interface
SOCK_SEQPACKET	Sequenced-Packet Socket

Tabelle 2.7: Soykettypen der socket()-Funktion

Der letzte Parameter der socket()-Funktion *protocol* bestimmt, welches Protokoll zwischen kommunizierenden Hosts eingestzt werden soll. Es wird oft *NULL* gesetzt, um das Standardprotokoll für das angegebene Domain und Sockettyp zu verwenden.

Beispielsweise die Standardprotokolle: Es wird ein Sockettyp von SOCK_STREAM und ein Domain von AF_INET für ein Kommunikationsprotoll als TCP bestimmt. Oder es wird ein Sockettyp von SOCK_DGRAM und ein Domain von AF_INET für ein Kommunikationsprotokoll als UDP verwendet. Die verschiedenen definierten Möglichkeiten des *protocol*-Parameters werden in der folgenden Tabelle 2.8 beschrieben.

Protokoll	Beschreibung
IPPROTO_IP	IPv4 Internet-Protokoll
IPPROTO_ICMP	Internet Control Message Protokoll
IPPROTO_RAW	Raw IP-Pakete Protokoll
IPPROTO_TCP	Transmission Control Protokoll
IPPROTO_UDP	User Datagram Protokoll

Tabelle 2.8: Protokolle der socket()-Funktion

2.3.2 Bindung eines Sockets

Nach dem Anlegen eines Sockets wird es als *unbenanntes Socket* bezeichnet, da es keine Protokoll-Adresse besitzt. Um die Nachrichten über einen Port mit Hilfe eines bestimmten Protokolls[9] zu versenden, muss das Socket mit einer lokalen Protokoll-Adresse und Portnummer verbunden werden. Um die IP-Adresse und Portnummer an das Socket zuzuweisen, wird die **bind(2)**-Funktion verwendet. Wie die anderen Systemaufrufe, liefert die bind()-Funktion die Zahl 0, wenn sie erfolgreich ist. Ansonsten wird im Fehlerfall -1 geliefert.

[9] Die einzigen von IANA bekanntegegebenen Portnummern für TCP und UDP mit den jeweiligen Beschreibungen werden in der Systemdatei "/etc/services" zugeordnet.

```
#include <sys/types.h>
#include <sys/socket.h>

int
bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

Der erste Parameter *sockfd* ist der Socketdeskriptor, welcher mit der socket()-Funktion erzeugt wurde. Der zweite Parameter *addr* ist ein Zeiger auf eine Socket-Adressstruktur *struct sockaddr* (siehe Datenstrukturen von Socket). *addrlen* ist die Länge der verwendeten Socket-Adressstruktur.

2.3.3 Auflösung von Hostnamen

Um die IP-Adresse von den Hostnamen aufzulösen, wird die **gethostbyname(3)**-Funktion verwendet. Sie bekommt einen Hostnamen und liefert einen Zeiger auf die Struktur *hostent* zurück, welche die IP-Adresse und einige Informationen beinhaltet, wie im Datenstrukturen von Socket näher beschrieben wird.

```
#include <netdb.h>

struct
hostent *gethostbyname(const char *name);
```

2.3.4 Verbindungsaufbau

Bis jetzt wurde ein Socket erstellt und mit der lokalen Protokoll-Adresse und Portnummer verbunden. Nun wird die Kommunikation zweiseitig, also erfolgt die Server- und die Client-Seite getrennt.

2.3.4.1 Verbindunganforderung

Um eine Verbindungsanforderung aus der Client-Seite zu senden, steht **connect(2)**-Funktion zur Verfügung. Sie liefert im Erfolgsfall eine 0, im Fehlerfall eine -1.

```
#include <sys/types.h>
#include <sys/socket.h>

int
connect(int sockfd, const struct sockaddr *addr, socklen_t len);
```

Der erste Parameter ist *sockfd*, der mittels *socket()*-Funktion erzeugt wurde. Der zweite Parameter ist ein Zeiger auf der *sockaddr*-Struktur, in der sich die Adresse des Servers befindet. *len* ist die Länge der Struktur, die übergeben wird.

2.3.4.2 Horchen

Auf der Server-Seite wird eine am erzeugten Socket einkommende Verbindunganforderung mit der **listen(2)**-Funktion abgehorcht. Im Erfolgsfall liefert die listen()-Funktion den Wert 0, ansonsten -1.

```
#include <sys/types.h>
#include <sys/socket.h>

int
listen(int sockfd, int backlog);
```

Das erste Argument ist das horchende Socket, und das zweite bietet einen Hinweis auf die Anzahl der Verbindungsanforderungen. Dafür bildet das Betriebssystem eine Warteschlange. In der Header-Datei <*sys/socket.h*> wurde *SOMAXCONN* für die maximale Warteschlange als 128 vordefiniert.

2.3.4.3 Verbindungsannahme

Nachdem eine Verbindungsanforderung an das erzeugte Socket mit der voreingestellten Portnummer angekommen ist, muss an der Server-Seite eine Akzeptierung dieser Anforderung mit Hilfe der **accept(2)**-Funktion angenommen werden. Diese Funktion nimmt eine Verbindungsanforderung aus der Warteschlange der listen()-Funktion, dann wird ein neues Socket mit dem gleichen Sockettyp, dem gleichen Protokoll und der gleichen Adressfamilie erstellt. Anschließend legt sie den neuen Socketdeskriptor für das neue Socket an.

```
#include <sys/types.h>
#include <sys/socket.h>

int
accept(int sockfd, struct sockaddr *restrict addr,
       socklen_t *restrict addrlen);
```

Das erste Argument ist der Socketdeskriptor aus der Warteschlange der listen()-Funktion. Der zweite Parameter ist der Zeiger auf eine *sockaddr*-Struktur, in der die Socket-Informationen von *sockfd*-Deskriptor für das neu angelegte Socket kopiert werden. *addrlen* ist die Länge der kopierten Socket-Information. Im Erfolgsfall liefert die accept()-Funktion den Socketdeskriptor des akzeptierten Sockets, ansonsten -1.

2.3.5 Datenübertragung

Wenn eine Verbindung zwischen dem Server und dem Client aufgebaut wurde, dann erfolgt die Datenübertragung bzw. Nachrichtenaustausch zwischen kommunizierenden Hosts.

2.3.5.1 Datensendung

Um die Daten zu versenden, stehten drei Systemaufrufe zur Verfügung, also **send(2)**, **sendto(2)** und **sendmsg(2)**. Die *send()*-Funktion darf nur verwendet werden, wenn das Socket in einem verbundenen Zustand ist. Jedoch die *sendto()*- und *sendmsg()*-Funktionen können jeder Zeit eine Nachricht versenden. Aus diesem Grund werden die *sendto()*- und *sendmsg()*-Funktionen üblicherweise für UDP-Sockets eingesetzt. Am Ende liefern diese drei Systemaufrufe die Anzahl der erfolgreich gesendeten Bytes, oder -1 im Fehlerfall.

```
#include <sys/types.h>
#include <sys/socket.h>

ssize_t
send(int sockfd, const void *buf, size_t nbytes, int flags);

ssize_t
sendto(int sockfd, const void *buf, size_t nbytes, int flags,
       const struct sockaddr *destaddr, socklen_t destlen);

ssize_t
sendmsg(int sockfd, const struct msghdr *msg, int flags);
```

Der erste Parameter der *send()*-Funktion ist der Socketdeskriptor, über den die Nachricht versendet wird. *buf* ist ein Zeiger auf einen Puffer, der die zu versendenden Daten enthält. Die Länge dieser Daten wird mit *len* angegeben. *flags*[10] spezifiziert die Art der Datenübertragung, wie im folgenden Code-Feld beschrieben wird.

```
#define MSG_OOB         0x1
#define MSG_PEEK        0x2
#define MSG_DONTROUTE   0x4
#define MSG_EOR         0x8
...
#define MSG_WAITALL     0x40
...
#define MSG_EOF         0x100
...
...
```

Die ersten vier Parameter der *sendto()*-Funktion sind gleich wie die Parameter der *send()*-Funktion. Der fünfte Parameter *destaddr* ist der Zeiger auf die Zieladresse des Datagramms, während *destlen* die Länge der Struktur der Zieladresse festlegt.

Die *sendmsg()*-Funktion sendet eine spezifische Nachrichtenstruktur mit den oben beschriebenen Flag-Codes über *flags* an den Socketdeskriptor *sockfd*. Die Datenstruktur *msghdr* wird im folgenden Code-Feld dargestellt.

[10] werden in der Header-Datei "<sys/socket.h>" definiert.

```
struct msghdr {
  void         *msg_name;       /* optinale Adresse                  */
  socklen_t     msg_namelen;    /* Laenge der Adresse                */
  struct iovec *msg_iov;        /* Datenfeld von Ein-/Ausgang-Puffer */
  int           msg_iovlen;     /* Anzahl der Elementen im Datenfeld */
  void         *msg_control;    /* nebenbetriebene Data              */
  socklen_t     msg_controllen; /* Anzahl von Data                   */
  int           msg_flags;      /* Flags fuer Empfangsdaten          */
};
```

2.3.5.2 Datenempfang

Um die ankommenden Daten von einem verbundenen Socket einzulesen, stehen drei Systemaufrufe zur Verfügung, nämlich **recv(2)**, **recvfrom(2)** und **recvmsg(2)**. Die *recv()*-Funktion arbeitet nur mit den verbundenen Sockets, jedoch die *recvfrom()*- und *recvmsg()*-Funktionen sind für nicht-verbundene Sockets geeignet. Die drei Funktionen liefern eine Anzahl der erfolgreich eingelesenen Bytes, ansonsten -1 im Fehlerfall.

```
#include <sys/types.h>
#include <sys/socket.h>

ssize_t
recv(int sockfd, void *buf, size_t nbytes, int flags);

ssize_t
recvfrom(int sockfd, void *buf, size_t len, int flags,
         struct sockaddr *restrict addr, socklen_t *restrict addrlen);

ssize_t
recvmsg(int sockfd, struct msghdr *msg, int flags);
```

Die *recv()*-Funktion liest mit der gegebenen Länge von Bytes *nbytes* aus dem Socketdeskriptor *sockfd* ein und speichert sie in den Puffer *buf*. Der Parameter *flags* spezifiziert die Art der Datenübertragung, wie im obigen Code-Feld beschrieben wurde.

Die ersten vier Parameter der *recvfrom()*-Funktion sind gleich wie die *recv()*-Funktion. Der Parameter *addr* enthält die Quelladresse des Datagramms, während *addrlen* die Länge der Struktur darstellt.

Die *recvmsg()*-Funktion empfängt eine spezifische Nachrichtenstruktur mit den oben beschriebenen Flag-Codes *flags* aus dem Socketdeskriptor *sockfd*. Die Datenstruktur *msghdr* wurde oben beschrieben.

2.3.6 Abschluss der Kommunikation

Wenn die Datenübertragung erfolgreich ist, dann sollten die geöffnete Sockets geschlossen werden. In diesem Erfolgsfall wird die **close(2)**-Funktion verwendet, um den File- bzw. Socketdesckriptor

zu löschen. Um ihn zu entfernen, wird dafür der Parameter *fildes* verwendet. Im Erfolgsfall liefert die *close()*-Funktion eine 0 zurück, ansonsten -1 im Fehlerfall.

```
#include <unistd.h>

int
close(int fildes);
```

Wenn ein Kommunikationsfehler bei der Datenübertragung auftritt, bleiben die Sockets immer noch offen. In diesem Fehlerfall wird die **shutdown(2)**-Funktion vom Betriebssystem verwendet. Sie dient dazu, den Socket *sockfd* mit dem Parameter *how* zu deaktivieren.

```
#include <sys/types.h>
#include <sys/socket.h>

int
shutdown(int sockfd, int how);
```

Der Parameter *how* sagt, wie der Socket deaktiviert wurde. Die möglichen Werte werden in der folgenden Tabelle 2.9 beschrieben.

how-Wert	Beschreibung
SHUT_RD	Weitere Empfang nicht anerkannt
SHUT_WR	Weitere Sendung nicht anerkannt
SHUT_RDWR	Weitere Empfang und Sendung nicht erkannt, impliziert SHUT_WR

Tabelle 2.9: how-Werte der shutdown()-Funktion

2.3.7 Datenstrukturen von Socket

Die oben beschriebenen BSD-Socket-Funktionen bekommen einige Zeiger auf die Datenstruktur **sockaddr**, welche die Adress-Familien sowie verwendete Protokoll-Adresse enthält, um ein Verbindungsaufbau auf den richtigen Rechner herzustellen. Für die Internet-Protokolle wurde auch eine Datenstruktur **sockaddr_in** weiter entwickelt, welche zusätzlich die IP-Adresse und Port-Nummer beinhaltet.

```
#include <sys/socket.h>

struct sockaddr {
        unsigned char   sa_len;
        sa_family_t     sa_family;
        char            sa_data[14];
};
```

```
#include <netinet/in.h>

struct sockaddr_in {
        uint8_t         sin_len;
        sa_family_t     sin_family;
        in_port_t       sin_port;
        struct in_addr  sin_addr;
        char            sin_zero[8];
};
```

Die Struktur *sockaddr* besteht aus einer verwendeten Adress-Familie *sa_family* sowie einem Adress-Wert *sa_data* und der gesamten Länge der Struktur *sa_len*. Die möglichen Werte der Adress-Familie wurden in der Tabelle 2.6 dargestellt.

```
#include <netdb.h>

struct hostent {
        char            *h_name;
        char            **h_aliases;
        int             h_addrtype;
        int             h_length;
        char            **h_addr_list;
#define h_addr          h_addr_list[0]
};
```

Die Socket-Adressstruktur **sockaddr_in** für IPv4 enthält die Adress-Familie *sin_family*, eine Portnummer *sin_port* sowie die IP-Adresse des Sockets *sin_addr* und die gesamte Länge der Struktur *sin_len*. Zusätzlich spielt das Element *sin_zero* eine Rolle, um die Größe der Struktur *sockaddr_in* zu der Größe der Struktur *sockaddr* zu entsprechen. Es ist eine gute Idee, die gesamte Struktur mit der **memset(3)**-Funktion zu entleeren bzw. mit 0 zu überschreiben, damit die Probleme von *sin_len* zu vermeiden.

Wie oben erwähnt, liefert die *gethostbyname()*-Funktion einen Zeiger auf die Struktur **hostent** als Netzwerkdatenbankbibliothek zurück, wenn **h_addr**[11] gültig ist. In manchen UNIX basierten Systemen wird die Element *h_addr* in der Struktur *hostent* nicht definiert. Aus diesem Grund gibt der C-Compiler immer Fehlermeldungen zurück. Um dies zu vermeiden, kann in einer verwendeten Header-Datei die folgende Zeile geschrieben werden.

```
#ifndef h_addr
#define h_addr h_addr_list[0]
#endif
```

[11] ist der erste Element des Arrays **h_addr_list**.

3 Überblick

Die Gesamtaufgabe wurde in mehrere Subaufgaben geteilt, damit jeder Schritt einzeln bearbeitet und analysiert werden kann. Somit wird gewährleistet, dass die Übersichtlichkeit auch bei größeren Schaltungen nicht verloren geht. Sollte jedoch trotz der Aufteilung in kleinere Teilschritte ein Problem auftreten, kann dank der Überschaubarkeit schnell und effizient der Fehler gefunden und behoben werden.

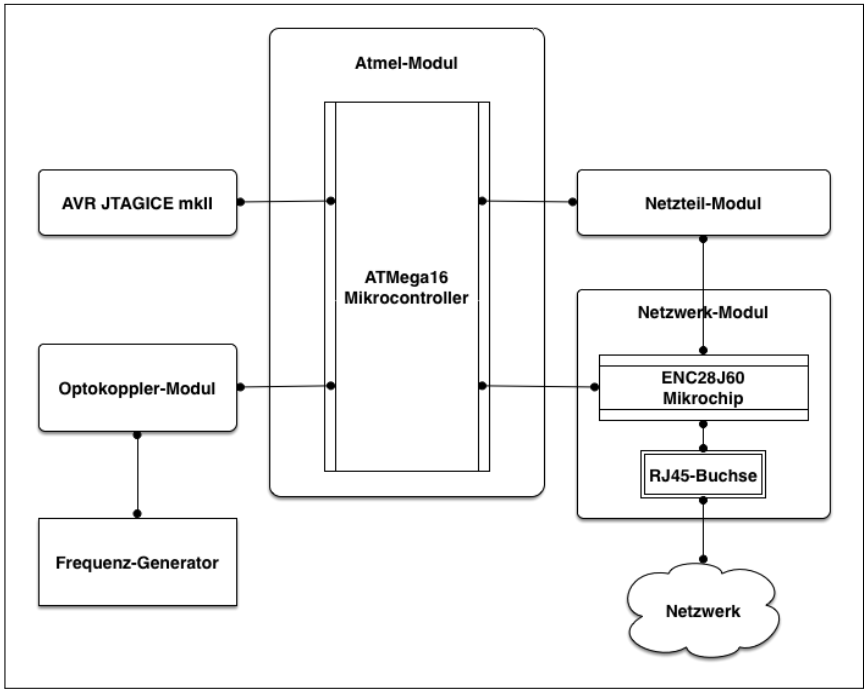


Abbildung 3.1: Zusammenbau von allen Modulen

Die Abbildung 3.1 zeigt, wie die einzelnen Module im Gesamtsystem miteinander verbunden sind. Danach wurde der Schaltplan laut dem obigen Gesamtsystem im Eagle[1]-Schaltplaneditor [4] erstellt. Die Gesamtschaltung ist im Anhang A zu sehen.

[1] ist ein Program der Firma CadSoft zur Erstellung von Leiterplatte zur Entwurfsautomatisierung elektronischer Systeme.

Es wird eine Schnittstelle, sog. **AVR JTAGICE mkII**, zwischen dem Atmel-Modul und dem verwendeten Rechner angeschlossen. Dieses Modul ermöglicht die Programmierung des ATmega16-Mikrocontrollers.

Das **Optokoppler-Modul** liest ein kontinuierlich-sinusförmiges Signal aus dem Frequenzgenerator aus und digitalisiert dieses Signal. Anschließend wird das kontinuierlich-digitalisierte Signal an den Eingangspin des Atmel-Moduls weitergeleitet.

Das **Atmel-Modul** liest das Eingangssignal aus dem Pin und sogleich beschäftigt es sich mit der Abarbeitung der Frequenzmessung des Signals mittels eines internen Timers. Nachfolgend werden die Messungswerte dem Netzwerk-Modul abgegeben, wenn sie abgefragt werden.

Das **Netzwerk-Modul** ist eine Kommunikationsschnittstelle für das UDP-Protokoll zwischen dem Atmel-Modul und einem beliebigen UNIX-Server. Die Frequenzwerten werden nach Anforderung des Servers über das Netzwerk-Modul weitergegeben. Für die Abfrage wird ein Dämon zum Laufen auf UNIX basierten Systemen programmiert.

Jedes Modul des Gesamtsystems wird im nächsten Kapitel näher beschrieben. Im Abschnitt 2.2 wurde das Kommunikationsprotokoll näher unter die Lupe genommen. Außerdem ist im Anhang B auch das programmierte Kommunikationsprotokoll zu lesen.

4 ENTWURF

Dieses Kapitel beschäftigt sich mit allen Modulen des Projektes. Unter anderem wird sowohl der Aufbau als auch die Zusammenschaltung mehrerer Module aus der elektrotechnischen Sicht beschrieben, welche in der Abbildung 3.1 grafisch dargestellt wurde.

4.1 Optokoppler-Modul

Eine einfache Methode zur Messung der Frequenz eines kontinuierlich-sinusförmigen Signals mit Hilfe eines Mikrocontrollers ist die Digitalisierung dieses Signals, damit die steigenden bzw. fallenden Flanken aufgezählt werden kann. Um dieses sinusförmige Signal zu digitalisieren, wird hier ein **Optokoppler** benutzt, der ein (Halbleiter-)Bauelement und vor allem in der Nachrichten-Übertragungstechnik sehr nutzbar ist.

Der Optokoppler dient der Signalübertragung zwischen zwei galvanisch vollständig getrennten Stromkreisen in einem gemeinsamen Gehäuse [20]. In einem Optokoppler bzw. Gabelkoppler befinden sich ein Lichtsender (*Led*) und ein Lichtempfänger (*Fototransistor*). Die Funktionsweise des Optokopplers ist die Übersetzung von einer sinusförmigen Eingangsspannung zu einer rechteckförmigen Ausgangsspannung [TS02]. Das heißt, dass der Strom am Ausgang über den Fototransistor nur dann fließt, wenn die positive Halbwelle den Lichtsender erreicht. Zusätzlich zu dieser Diode für die positive Halbwelle wird eine Diode für die negative Halbwelle im Eingangsbereich benötigt. In dieser Arbeit findet eine Spannungsquelle im Eingangsbereich mit 10 V (AC) und eine Spannungsquelle im Ausgangsbereich mit 5 V (DC), wie in der Abbildung 4.1 zu sehen ist.

4.2 Netzteil-Modul

Sowohl für das Atmel-Modul als auch für das Netzwerk-Modul wird ein Netzteil benötigt, damit diese an das Netz angeschlossen werden können. Jedes Modul hat eine spezifische Versorgungsspannung. Die Versorgungsspannung beim Atmel-Modul liegt bei 5 Volt ($\pm$10%) und beim Netzwerk-Modul bei 3.3 Volt ($\pm$10%).

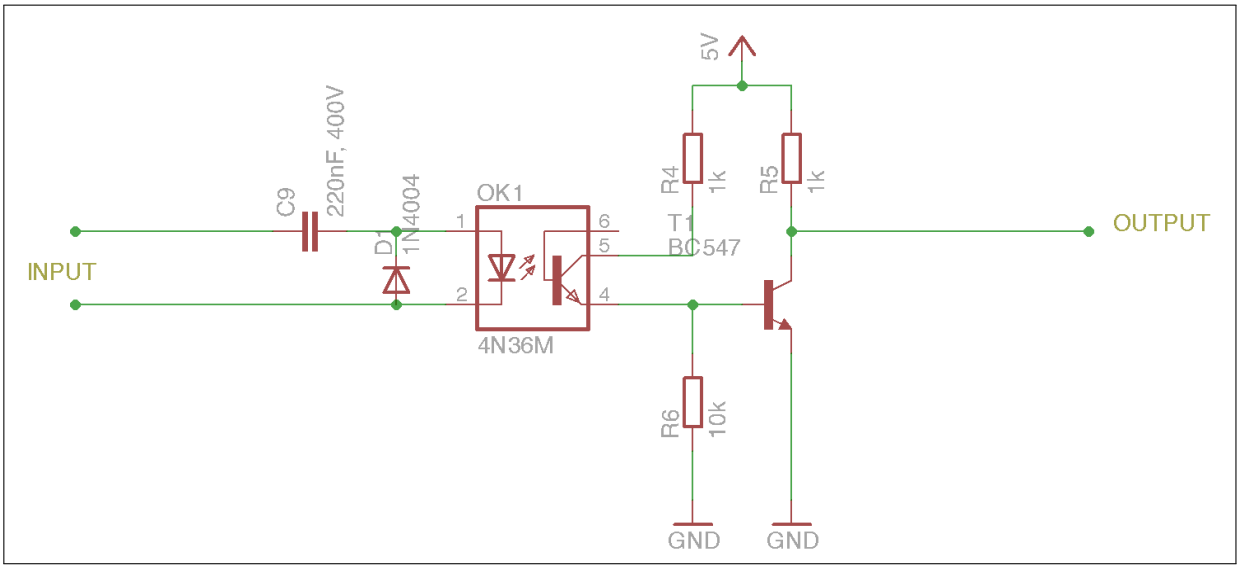


Abbildung 4.1: Optokoppler-Modul

Wie in der Abbildung 4.2 zu sehen ist, wird das Atmel-Modul mit 5 Volt Spannung und das Netzwerk-Modul mit 3.3 Volt Spannung verbunden.

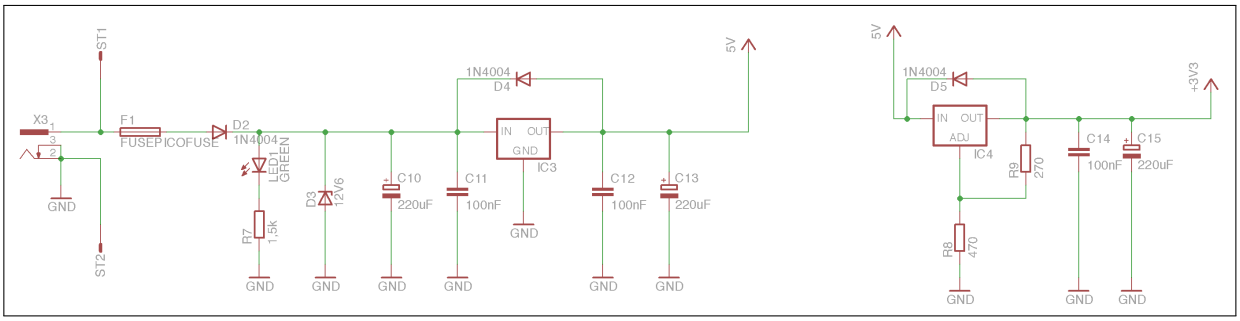


Abbildung 4.2: Netzteil-Modul

4.3 AVR JTAGICE mkII

Um einen funktionierenden Mikrocontroller zu erzielen, muss dieser Mikrocontroller mit einem Entwicklungswerkzeug bzw. mit einem Programmer programmiert werden. Es wurde für diese Arbeit ein Entwicklungswerkzeug vom Firma *ATMEL Corporation* ausgewählt, nämlich **AVR JTAGICE mkII** [14] (siehe Abbildung 4.3). Dieses Entwicklungswerkzeug bietet die Möglichkeit für das "On-Chip-Debugging[1]" und die Programmierung über JTAG für alle AVR 8-bit RISC-Mikrocontrollern. Die JTAG-Schnittstelle wurde für ein Test-Access-Port (*TAP*) entwickelt und als *IEEE 1149.1* im Jahr 1990 standardisiert.

[1]On-Chip-Debugging dient die Möglichkeit, mögliche Fehler im Programmcode direkt auf dem Chip zur Laufzeit zu debuggen bzw. zu finden.

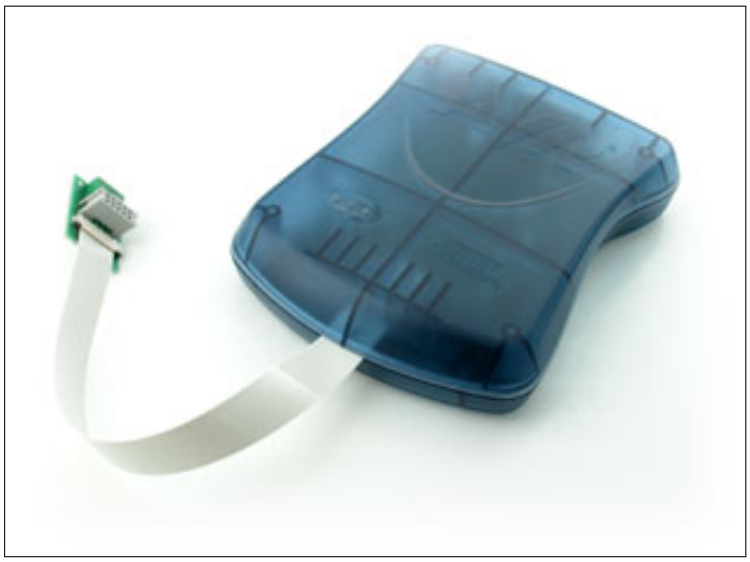


Abbildung 4.3: AVR JTAGICE mkII [15]

Der JTAGICE-Programmer wird zwischen Entwicklungs- und Zielplattform geschaltet. Er übermittelt somit die Kommunikation zwischen dem Rechner und dem AVR-Mikrocontroller. An den Rechner wird der JTAG-Programmer über ein USB-Kabel angeschlossen und verbraucht keine zusätzliche Versorgungsspannung. An den Mikrocontroller wird der JTAG-Programmer direkt zu den AVR-Pins angeschlossen. Die Pinbelegungen und ihre Beschreibungen sind in der Tabelle 4.1, die Verdrahtung zwischen dem AVR JTAGICE mkII und dem ATMEL-Mikrocontroller in der Tabelle 4.2 beschrieben.

Pin	Signal	I/O	Beschreibung
1	TCK	Output	Test-Clock, Taktsignal aus JTAGICE mkII zu JTAG-PORT des Ziel-Controllers
2	GND	-	Erde
3	TDO	Input	Test-Data-Output, Data-Signal aus JTAG-Port des Ziel-Controllers zu JTAGICE mkII
4	V_{ref}	Input	Referenzspannung
5	TMS	Output	Auswahl des Testmodes aus JTAGICE mkII zu JTAG-PORT des Ziel-Controllers
6	nSRST	OUT-/Input	Zur Steuerung und zur Überwachung über RESET-Pin des Ziel-Controllers
7	-	-	nicht verbunden
8	nTRST	NC (Output)	nicht verbunden
9	TDI	Output	Test-Data-Input, Data-Signal aus JTAGICE mkII zu JTAG-PORT des Ziel-Controllers
10	GND	-	Erde

Tabelle 4.1: PIN-Belegungen von JTAG-Schnittstelle und ihre Beschreibungen

JTAGICE mkII		ATMega16	
TCK	PIN 1	PC2 (PIN 24)	TCK
GND	PIN 2	GND (PIN 11)	GND
TDO	PIN 3	PC4 (PIN 26)	TDO
V_{ref}	PIN 4	VCC (PIN 10)	V_{ref}
TMS	PIN 5	PC3 (PIN 25)	TMS
nSRST	PIN 6	RESET (PIN 9)	nSRST
nicht verbunden	PIN 7	nicht verbunden	nicht verbunden
nTRST	PIN 8	nicht verbunden	nicht verbunden
TDI	PIN 9	PC5 (PIN 27)	TDI
GND	PIN 10	GND (PIN 11)	GND

Tabelle 4.2: Verdrahtung zwischen JTAGICE mkII und ATMega16-Mikrocontroller

4.4 Atmel-Modul

Bei der AVR-Mikrocontroller-Familie von Atmel [1] handelt es sich um 8-Bit Mikrocontroller. Damit die maximale Leistung und die Parallelität erreicht werden kann, verwenden AVR-Mikrocontroller eine Harvard-Architektur[2], in welcher der Befehls- und der Datenspeicher *logisch* und *physisch* von einander getrennt sind (siehe Abbildung 4.4). Der Vorteil der Harvard-Architektur liegt darin, dass Befehle und Daten mit einem einzigen Taktzyklus geladen bzw. geschrieben werden. Daher wird diese Architektur in den RISC-Kernen verwendet. Die logische und arithmetische Befehle kommen beim RISC-Kern ein *Dreiadressbefehlsformat*[3] vor.

Es wird in dieser Arbeit der AVR-Mikrocontroller-Type **ATMega16** verwendet, welcher ausreichend genug Leistung für eine einfache Laboranwendung bereitstellt. Der ATMega16-Mikrocontroller verfügt über einen 16 KB großen Flashspeicher, in dem die Programme abgelegt werden, und über einen 512 Byte großen EEPROM, das sich in einem separaten Datenbereich befindet, sowie über einen 1 KB großen SRAM-Speicher. Die CPU-Taktfrequenz ist bis zu 16 MHz begrenzt. Es gibt einen 8-Bit bereiten Bus für Daten und einen 16-Bit bereiten Bus für Befehle. Weiterhin bietet der ATMega16 Ein-/Ausgangsschnittstellen, Analog/Digital-Umwandler und 8- bzw. 16-Bit Timers. Zur Kommunikation mit der Außenwelt befinden sich SPI-, U(S)ART-, I^2C und JTAG-Schnittstelle.

Zum Konfigurieren eines AVR-Mikrocontrollers werden *Fuse-Bits* benutzt. Es gibt zwei Bytes (*low* und *high*) zum Konfigurieren von Fuse-Bits. Diese werden beim System-Start des Mikrocontrollers und während dem Betrieb verwendet. Bei einem neuen AVR-Mikrocontroller werden die Fuse-Bits nach der Auslieferung vorkonfiguriert. Die vorkonfigurierten Fuse-Bits werden werksseitig so eingestellt, dass sie den internen RC-Oszillator mit einem 1 MHZ verwenden. Die Fuse-Bits sollen eigentlich nur einmal oder für die notwendigen Anwendungen konfiguriert werden. Die Tabelle 4.3 und die Tabelle 4.4 zeigen, welche Aufgaben die jeweiligen Bits von Low- bzw. High-

[2]bezeichnet ein Architekturprinzip zur Realisierung besonders schneller CPUs und Signalprozessoren und erst im Jahr 1959 von *Harvard Mark I* an der Harvard-Universität Cabridge (USA) gestellt.

[3]Bei Dreiadressbefehlsformat gibt es den Operationscode, die Quelladresse und anschließlich eine Zieladresse.

Byte haben. Die Spalte *Konfiguration* zeigt, ob für diese Arbeit die jeweilige Bit programmiert wird oder nicht. Die wichtige Anmerkungen zu den Fuse-Bits sind:

- "unprogrammiert" bedeutet, dass der Wert 1 auf den Fuse-Bit geschrieben ist.
- "programmiert" bedeutet, dass der Wert 0 auf den Fuse-Bit geschrieben ist.

Bit	Fuse Low Byte	Beschreibung	Konfiguration
7	BODLEVEL	Trigger für Brown-Out-Detector[4]. Wenn der Bit gesetzt ist, ist der minimale Schwellenwert 4.0 Volt, sonst 2.7 Volt.	unprogrammiert
6	BODEN	Brown-Out Detector aktivieren. Falls BODEN gesetzt ist, wird der AVR-Mikrocontroller mit dem Trigger-Signal neugestartet, wenn V_{cc} unter den Schwellenwert fällt.	unprogrammiert
5	SUT1	Start-Up-Zeit. Regeln das Bootverhalten des AVR-Mikrocontrollers.	unprogrammiert
4	SUT0	Start-Up-Zeit. Regeln das Bootverhalten des AVR-Mikrocontrollers.	unprogrammiert
3	CKSEL3	Die Taktquelle des Controllers bestimmen.	unprogrammiert
2	CKSEL2	Die Taktquelle des Controllers bestimmen.	unprogrammiert
1	CKSEL1	Die Taktquelle des Controllers bestimmen.	unprogrammiert
0	CKSEL0	Die Taktquelle des Controllers bestimmen.	unprogrammiert

Tabelle 4.3: Fuse-Bits Low Byte

Bit	Fuse High Byte	Beschreibung	Konfiguration
7	OCDEN	On-Chip-Debugging. Bietet die Möglichekeit ein Echtzeit-Debugging des Mikrocontrollers bei der Aüsführung im Zielsystem.	programmiert
6	JTAGEN	JTAG-Schnittstelle für Debugging.	programmiert
5	SPIEN	Serielle Programmierung.	programmiert
4	CKOPT	Oszillator-Verstärkung.	programmiert
3	EESAVE	Löschen des EEPROM-Speichers beim Programmieren.	unprogrammiert
2	BOOTSZ1	Setzen der Bootloader-Größe.	programmiert
1	BOOTSZ0	Setzen der Bootloader-Größe.	programmiert
0	BOOTRST	Falls BOOTRST gesetzt ist, wird das Programm nach dem RESET auf erste Adress des Bootloaders gesprungen.	unprogrammiert

Tabelle 4.4: Fuse-Bits High Byte

Die Webseite *engbedded* [18] bietet die Möglichkeit auf schnellsten Wege die Konfiguration von Fuse-Bits zu erstellen bzw. zu berechnen. Durch Auswahl des ATMega16-Mikrocontrollers und

[4] überwacht die V_{cc} Spannung.

der benötigten Einstellungen auf der Webseite, gibt sie das High- und das Low-Byte zurück. Für diese Arbeit ist das High-Byte **0x09** und das Low-Byte **0xFF**. Aus diesen zwei Bytes ergibt sich die Spalte "Konfiguration" in Tabelle 4.3 und Tabelle 4.4. Nun können die zwei Bytes in den ATMega16-Mikrocontroller geschrieben werden. Um diese Konfiguration zu schreiben, wird das Programm *avrdude* [2] mittels der folgenden Kommandozeile im UNIX-Terminal verwendet:

```
avrdude -c PROGRAMMER -P PORT -p PART -U lfuse:w:0xff:m -U hfuse:w:0x09:m
```

Für eine ausführliche Erläuterung kann das Benutzerhandbuch [16] des Programms *avrdude* gelesen werden.

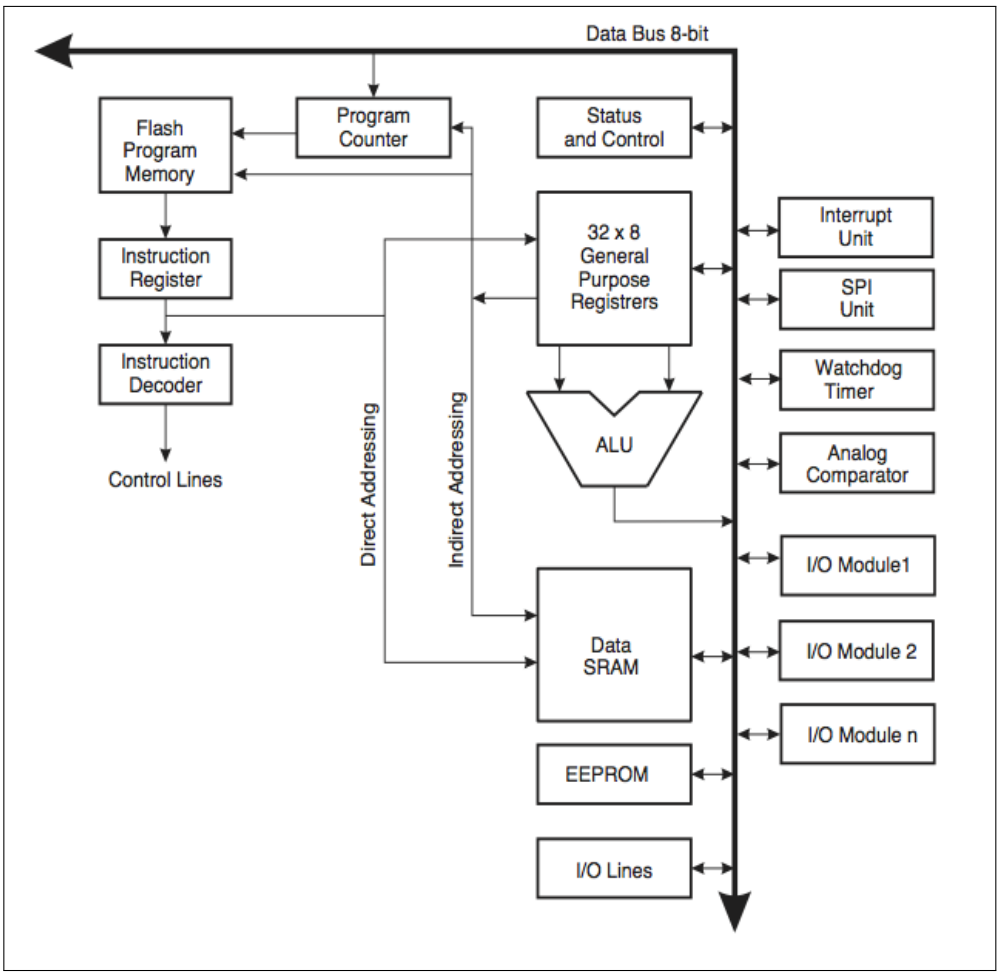


Abbildung 4.4: AVR-Architektur [12]

4.4.1 Timer-Einheit

Der Timer ist eine Einheit im Mikrocontroller, die einen Zähler enthält und diesen Zähler periodisch inkrementiert und/oder dekrementiert. Unmittelbar nach dem Auftreten eines Ereignisses

wird das Hauptprogramm unterbrochen, wobei diese Unterbrechung meist mit dem englischen Begriff *Interrupt* bezeichnet wird. Eine spezielle Art des Interrupts ist das Timer-Interrupt, weil speziell eingestellt werden kann, wann der Interrupt ausgelöst werden soll. Genau genommen ist ein Timer-Interrupt auch ein Zustand[5] des Zählers. Grundsätzlich wird der Timer-Interrupt genau dann ausgelöst, wenn der Wert des Zählers einen vordefinierten Wert erreicht oder ein externes Ereignis an einem Pin des Mikrocontrollers auftritt. Der ATMega16-Mikrocontroller hat drei verschiedene unabhängige Timer-Einheiten, wobei zwei mit 8-Bit und ein mit 16-Bit-Auflösung.

4.4.1.1 CTC-Modus

In dieser Arbeit wurde der 16-Bit Timer/Counter1 ausgewählt und dieser wurde im CTC-Modus konfiguriert. In diesem Modus wird der Timer-Interrupt nur dann ausgelöst, wenn der Zähler (*TCNT1*[6]) den maximalen Wert (*TOP*) erreicht. Anschließend fängt er wieder bei dem Wert (*BOTTOM*) an (siehe Abbildung 4.5).

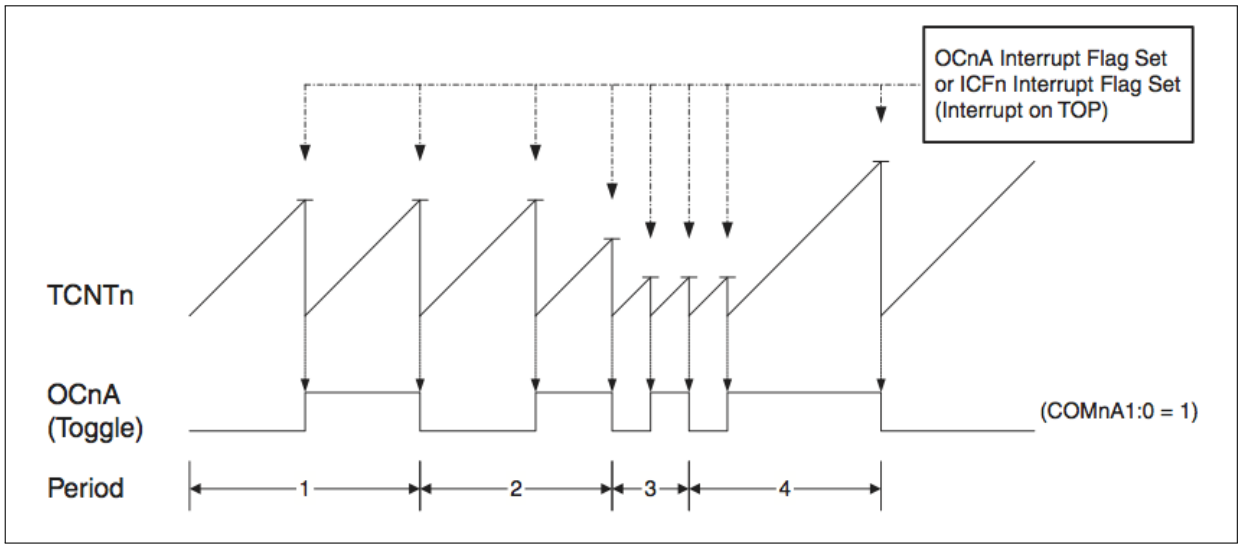


Abbildung 4.5: Timer-Diagramm für CTC-Mode [12]

Zum Konfigurieren des Timer/Counter1 gibt es zwei Steuerregister, wie in Tabelle 4.5 und Tabelle 4.6 beschrieben wird.

COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10

Tabelle 4.5: Timer/Counter1 Controler Register A (TCCR1A)

[5] Wenn der Interrupt auslöst, wird diese Stelle markiert, wobei diese Markierung als *Zeitstempel* in Echtzeit bezeichnet wird.

[6] Der Timer/Counter**1**-Zähler ist ein 16-Bit Register und enthält untereinander geschachtelte **LOW** (*TCNT1L*) und **HIGH** (*TCNT1H*) Bytes.

ICNC1	ICES1	(reserviert)	WGM13	WGM12	CS12	CS11	CS10

Tabelle 4.6: Timer/Counter1 Controler Register B (TCCR1B)

Jedes Bit an den Steuerregistern hat eine spezifische Aufgabe. Änderungen an diesen Bits rufen verschiedene Timer-Modi auf. Bei einem Timer sind verschiedenste Kombinationen möglich, welche im Datenblatt vom ATMega16 zu entnehmen ist.

Die zwei Byte Steuerregister des Timer/Counter1 sind wie folgt konfiguriert:

- CTC-Modus
- Der Zähler wurde periodisch in 1 HZ beschränkt.
- Input-Capture Noise Canceler
- Der Vorteiler wurde mit 256[7] gewählt.

4.4.1.2 Input Capture

Input Capture ist ein Hardwareteil, der zur genaueren Zeitmessung dient. Das Eingangssignal muss an den ICP angeschlossen werden, um eine Messung durchzuführen. Angewendet wird der Input Capture zwischen zwei aufeinanderfolgende Flanken, wobei die Erkennung entweder auf steigende oder fallende Flanken basiert.

Der Zähler läuft genau wie im CTC-Modus vom BOTTOM- zum TOP-Wert. Wenn eine eingestellte Zustandsänderung (fallende oder steigende Flanken) am ICP auftritt, wird der aktuelle Zählerwert auf eine Variable übergeben. Bei der nächsten Zustandsänderung wird die Differenz zwischen dem aktuellen Zählerwert und dem zuvor kopierten Zählerwert ermittelt. Diese Differenz ist die Messung zwischen zwei aufeinanderfolgenden Zustandsänderungen. Dieses Szenario findet bei jeder positiven bzw. negativen Flanke fortlaufend statt.

4.4.2 SPI-Schnittstelle

Die SPI-Schnittstelle ist ein Bus-System, welches synchron serielle Daten überträgt und von der Firma Motorola entwickelt wurde. Die Funktionsweise basiert auf dem Master-Slave-Prinzip. Die SPI-Schnittstelle unterstützt den richtungsunabhängigen (Full Duplex[8]) Nachrichtenaustausch. Die einfache Master-Slave-Verbindung kann wie in der Abbildung 4.6 hergestellt werden.

Das Master-Slave-Prinzip ist eine Zugriffsform, bei welcher die höchste Priorität am Bus zu schreiben beim Master liegt. Außerdem entscheidet der Master über den gemeinsam genutzten Übertragungskanal, bei welcher ein oder mehrere Slaves auch angeschlossen sind. Dies führt

[7] Dies bedeutet, dass der Systemtakt um den Faktor 256 geteilt wird.
[8] Daten können in beide Richtungen gleichzeitig übertragen werden.

dazu, dass wenn der Slave den Bus verwenden möchte, muss er warten, dass der Master ihn auffordert, Daten zu senden bzw. zu empfangen. Dieses Prinzip wird sehr oft auch *zentrales Polling*[9] genannt.

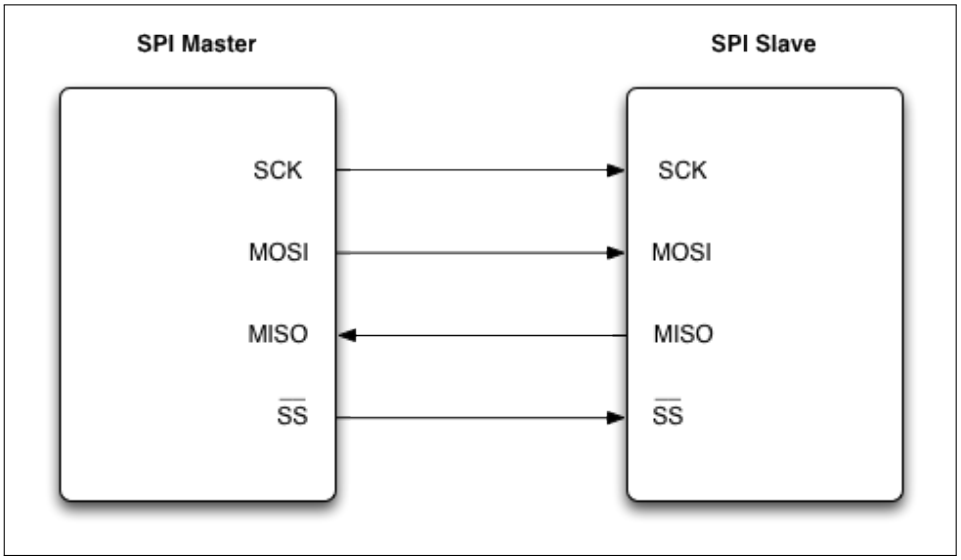


Abbildung 4.6: SPI-Verbindung zwichen einem Master und einem Slave

Der Übertragungstakt wird vom Master erzeugt und über die Ausgangsleitung (*SCK*-Pin) an alle Slaves geleitet. Im Master-Betrieb muss der SCK-Pin (*PB7*) des ATMega16-Mikrocontrollers als Ausgang eingestellt werden. Im Gegensatz muss der SCK-Pin im Slave-Betrieb als Eingang konfiguriert werden. Dabei wird der $\overline{SS}$-Pin **LOW** gesetzt, damit der Slave zum Empfangen bereit gestellt wird. Die wichtigste Pins und deren Initialisierungen im Master- und Slave-Betrib werden in der folgenden Tabelle 4.7 beschrieben:

Pin	Master-Betrieb	Slave-Betrieb
MOSI	als Ausgang zu initialisieren	automatisch Eingang
MISO	automatisch Eingang	als Ausgang zu initialisieren
SCK	als Ausgang zu initialisieren	automatisch Eingang
$\overline{SS}$	als Ausgang zu initialisieren	automatisch Eingang

Tabelle 4.7: Initialisierung von SPI-Pins

4.4.3 Debugging

Bei der Programmierung ist es sehr wichtig zu wissen, in welchem Zustand das Programm sich gerade befindet. Dies erleichtert das Debugging erheblich, da bei jeder Programmabzweigung beispielsweise eine Led eingeschaltet werden kann. Durch das gezielte Ein- bzw. Ausschalten der Leds wird genau erkannt, welchen Pfad das Programm durchläuft. In dieser Arbeit wurde

[9] Der Master fragt ununterbrochen die Slaves an, ob sie den Bus benötigen.

PORTA zum Debugging ausgewählt, an den die Leds verbunden sind. Es können nur so viele Leds als Ausgang dienen, so viele auch als Ausgänge definiert werden. Beispielsweise ist es sehr hilfreich, wenn eine Led für ein bestimmtes Ereignis (z.B. Timer-Interrupt) abwechselnd ein- bzw. ausgeschaltet wird (*toggle*).

4.5 Netzwerk-Modul

Damit zwei oder mehrere Geräte miteinander verbunden werden können, um gegenseitig Daten auszutauschen, wird eine Kommunikationsschnittstelle benötigt. Die Kommunikationsschnittstelle baut ein Übertragungskanal zwischen den kommunizierenden Geräten auf. Heutzutage gibt es viele Kommunikationsprotokolle mit verschiedenen Austauschstrategien auf unterschiedlichen Kommunikationsebene. Die Praxis zeigt, dass zwischen den Computern sehr oft das Ethernet-Kanal in Verwendung kommt. Die Vernetzung, welche auf Ethernet-Kanal basiert, wird als LAN bezeichnet. Dieses wird verwendet, um in einem lokalen Netz höhere Datenraten zu erzielen.

Im nächsten Kapitel werden mehrere Kommunikationsprotokolle näher beschrieben. Dabei wird sowohl der Aufbau als auch die Funktionsweise der einzelnen Schichten der Kommunikationsprotokolle aufgezählt. Auch die hardwarenahe Realisierung des Aufbaus und die Anordnung des Speichermoduls finden in diesem Unterkapitel Platz.

4.5.1 Aufbau

Der Hersteller *Mikrochip Technology Inc* [9] produziert sehr effiziente Mikrochips für Ethernets. Auch in dieser Arbeit wurde ein Mikrochip dieses Herstellers verwendet, und zwar das Modell **ENC28J60** [5]. Dieser Mikrochip ist ein *IEEE 802.3* [8] kompatibler Ethernet-Controller.

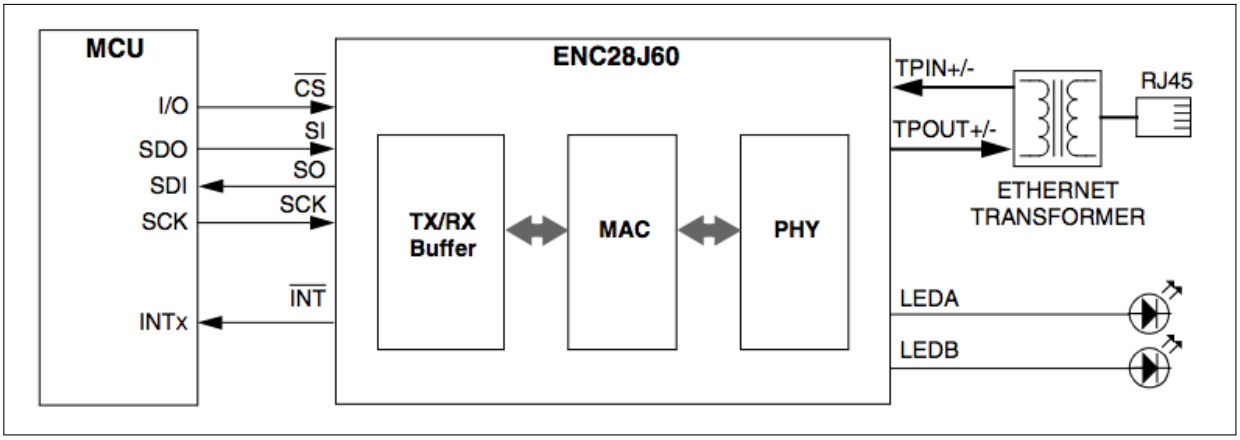


Abbildung 4.7: Basisverbindungen zwischen dem ENC28J60 und einem Mikrocontroller [11]

Der ENC28J60-Mikrochip ist im Gegensatz zu anderen Ethernet-Controllern ein kleiner Chip mit 28-Pins und besitzt einen kleinen Speicher. Es wurde gezielt dieser Mikrochip verwendet, da die Einfachheit und Übersichtlichkeit bei der Laboranwendungen nicht verloren geht.

Verglichen mit anderen Netzwerk-Mikrochips ist der ENC28J60 klein und dementsprechend auch sehr einfach, ihn in Schaltungen einzubauen. Er enthält eine SPI-Schnittstelle und kann über sie von einem Mikrocontroller leicht angesteuert werden. Die Programmierung der SPI-Schnittstelle ist sowohl im Vollduplex- als auch im Halbduplex-Modus möglich. Allein durch Basisverbindungen des Moduls ist die Funktion gegeben. Wie in der Abbildung 4.7 zu erkennen ist, handelt sich bei den Basisverbindungen um einige wenige Pins.

Weiters bietet der ENC28J60 folgende charakteristische Eigenschaften:

- integrierten MAC
- 10 $MBit/s$ Ethernet-Schnittstelle (10BASE-T Physical Layer)
- Voll-/Halb-Dublex Datenverkehr
- 8 Kilobytes internen Puffer
- SPI-Takt bis zu 25 MHz

Um mit einem Rechner im LAN zu kommunizieren, muss der ENC28J60-Mikrochip an eine Ethernet-Buchse angeschlossen werden, wie in der Abbildung 4.7 dargestellt wurde. Es wird ein festes Kommunikationskabel zwischen dem ENC28J60-Mikrochip und einem Server verlegt. Um das Ethernetkabel an die Buchse anzustecken, wird eine RJ45-Netzwerkstecker benötigt, welcher ein standardisierter Modularstecker ist. Für die **10BASE-T**-Kommunikation werden nicht alle Adernpaare benötigt. Belegt sind nur zwei Adernpaare, wobei das erste Adernpaar für den Datenausgang und das zweite für den Dateneingang verwendet werden. Diese Adernpaare besitzen je zwei Pins, wobei immer je ein positives und ein negatives Pin für das Eingangs- bzw. Ausgangssignal benötigt werden (siehe Tabelle 4.8). Die Verdrahtung zwischen der RJ45-Buchse und dem ENC28J60-Mikrochip wurde in der Tabelle 4.9 dargestellt.

PIN	Signal	Beschreibung	Farbe
1	TX+	positive Sendedaten	weiß/grün
2	TX-	negative Sendedaten	grün
3	RX+	positive Empfangsdaten	weiß/orange
4	nicht belegt		blau
5	nicht belegt		weiß/blau
6	RX-	negative Empfangsdaten	orange
7	nicht belegt		weiß/orange
8	nicht belegt		braun

Tabelle 4.8: Pinbelegung von RJ45-Modularbuchse bzw. -stecker [3]

Wie in der Abbildung 4.7 zu sehen ist, wird der ENC28J60-Mikrochip nach der Verdrachtung mit dem ATMega16-Mikrocontroller über $\overline{\mathbf{CS}}$ ausgewählt werden kann. Die Daten werden über den **SI**-Pin und über den **SO**-Pin ein- bzw. ausgelesen, welche mit dem **SCK** (Takt) von dem

ENC28J60		RJ45-Buchse	
PIN17	TPOUT+	TX+	1
PIN16	TPOUT-	TX-	2
PIN13	TPIN+	RX+	3
PIN12	TPIN-	RX-	6

Tabelle 4.9: Verdrahtung zwischen dem RJ45-Buchse und dem ENC28J60-Mikrochip

Mikrocontroller über die Synchron-Serielle Schnittstelle gesteuert werden können. Mit dem $\overline{\text{INT}}$-Pin gibt es eine Möglichkeit zu merken, ob ein passendes Ethernetpaket eingetroffen ist.

Außerdem besitzt der ENC28J60-Mikrochip zwei Pins (*PIN26* und *PIN27*) für die Leds. Während dem Datenempfang blinkt die *LEDB* und die *LEDA* während der Datensendung. Diese Funktion ist gegeben, wenn die Buchse sie unterstützt. Ansonsten können diese zwei Leds seperat verdrahtet werden, damit das Debugging beim Datenaustausch erleichtert wird.

4.5.2 Speicheranordnung

Der Gesamtspeicher des ENC28J60-Mikrochips wird als statisches RAM implementiert. Der Speicher wurde in drei Teile unterteilt, welche **Steuerregister**, **Ethernet-Puffer** und **PHY-Register** genannt werden, wie in der Abbildung 4.8 zu sehen ist.

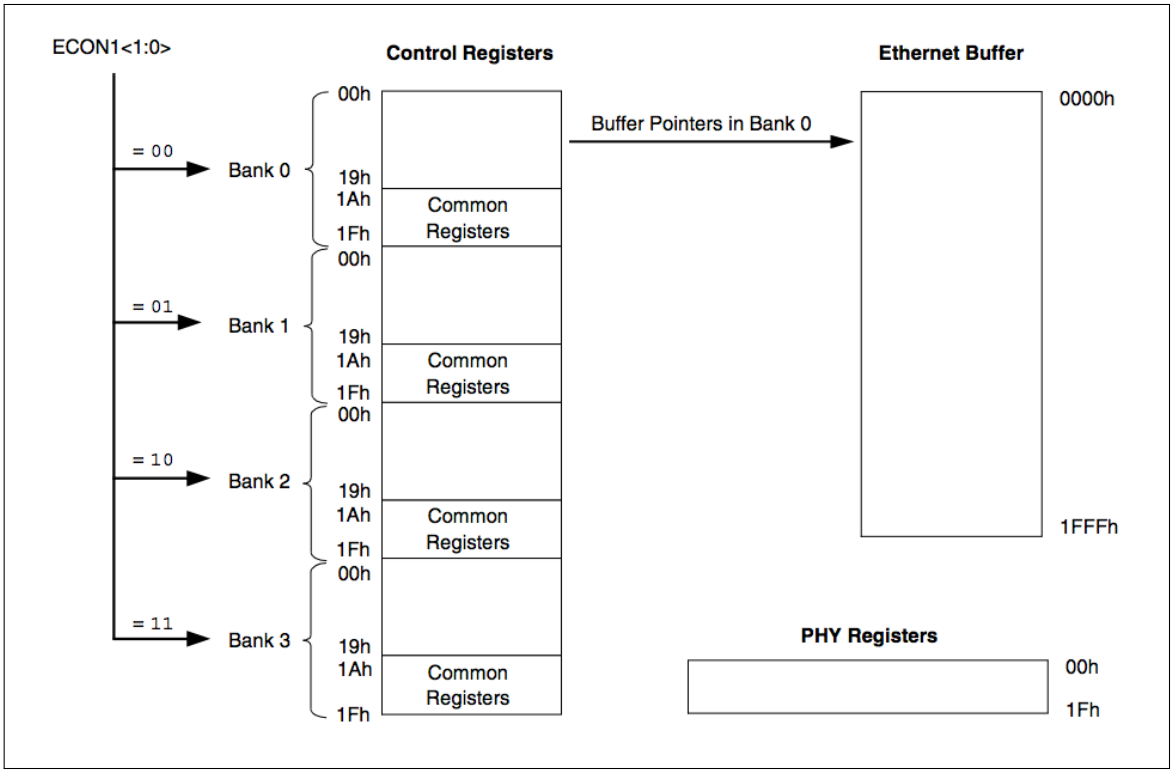


Abbildung 4.8: Speicheranordnung des ENC28J60-Mikrochips [11]

Das Steuerungsregister (Control Register) bildet die wichtigste Schnittstelle zwischen dem Mikrocontroller und ENC28J60-Mikrochip. Die Aufgaben des Steuerregisters sind die Konfiguration, die Steuerung und die Überwachung des Mikrochips. Das Steuerregister enthält vier Bänke untereinander. Jede Bank hat eine Größe von 32-Bit. Die letzten fünf Register (von $1B$ bis $1F$) jeder Bank weisen auf die gleichen Registernamen. Sie sind Schlüsselregister, welche als Steuerung bzw. Überwachung des Mikrochips verwendet werden. Um die aktive Bank auszuwählen, wird der **ECON1<1:0>** (sog. Common Register) eine von vier Kombinationen (00, 01, 10 oder 11) für die Bänke gesetzt.

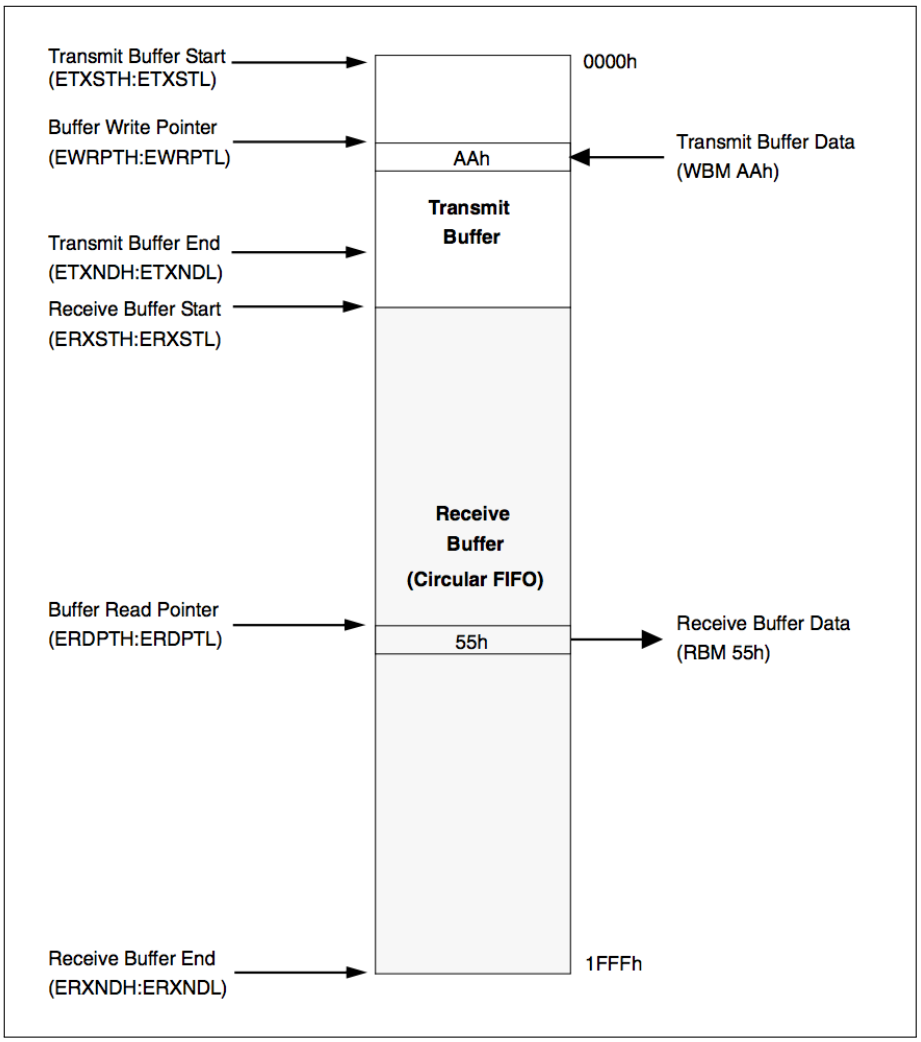


Abbildung 4.9: Pufferanordnung des ENC28J60-Mikrochips [11]

ENC28J60-Mikrochip hat 8 KB Pufferspeicher. Die erste vier Register der Bank-0 sind Zeiger auf die Adresse des Pufferspeichers, welche zum Lesen und Schreiben gedacht sind. Der Puffer ist unterteilt in zwei, nämlich in Empfangspuffer und Sendepuffer. Die Größe des Sendepuffers ist abhängig von der Größe des Empfangspuffers. Umso größer der Empfangspuffer ist, desto kleiner ist der Sendepuffer. Als Sendepuffer wird dieser Bereich im Speicher bezeichnet, welcher

nicht für den Empfang benötigt wird. Die empfangene Daten werden im Empfangspuffer nach dem FIFO-Prinzip (Ringwarteschleife) abarbeitet. Die gesamte Pufferandordung wurde in der Abbildung 4.9 dargestellt.

Um das PHY-Modul zu konfigurieren, wird das **PHY-Register** verwendet, welches verfügbare 16-Bit beinhaltet. Wegen den Sicherheitsgründen gibt es keinen direkten Zugang über die SPI-Schnittstelle zu diesem Register. Stattdessen wird der Zugang durch einen speziellen Satz von MAC-Steuerregister erzielt.

5 Ergebnisse

Zum Abschluss dieser Arbeit wird eine kurze Erklärung von Ergebnissen zusammengefasst. Wie in der Abbildung 5.1 zu sehen ist, wird ein kontinuierlich-sinusförmiges Signal an den Pin A und Pin B des Optokopplers angeschlossen. Das vom Frequenzgenerator generierte Signal wird in der Abbildung 5.2(a) dargestellt. Die Diode $D1$ spielt hier eine wichtige Rolle, damit der Strom von B nach A fließen kann. Danach wird das Eingangssignal des Optokopplers zwischen den Pin 1 und Pin 2 am Oszilloskop gemessen, wie in der Abbildung 5.2(b) zu sehen ist. Anschließend wird das Eingangssignal mit Hilfe des Optokopplers am Ausgangspin als *digitalisiertes Signal* entnommen, wie in der Abbildung 5.2(c) dargestellt wird.

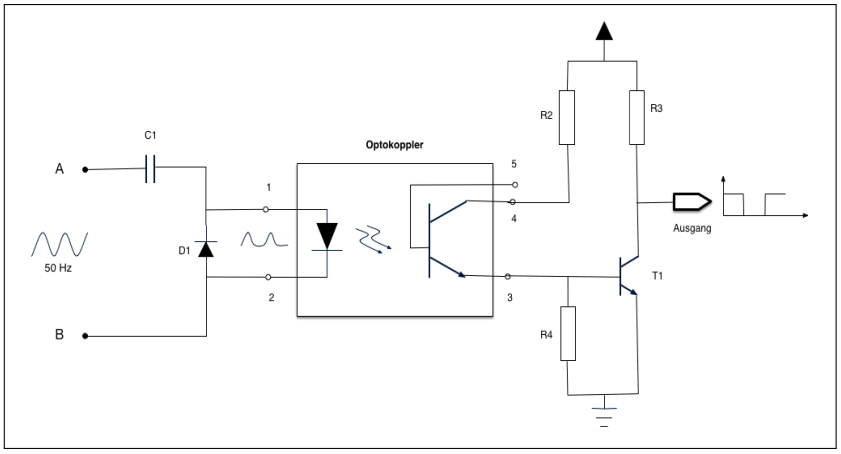
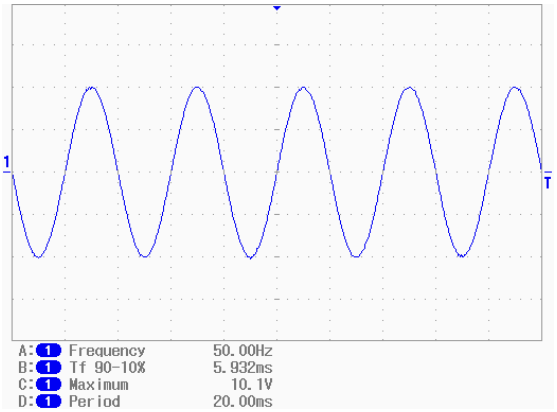


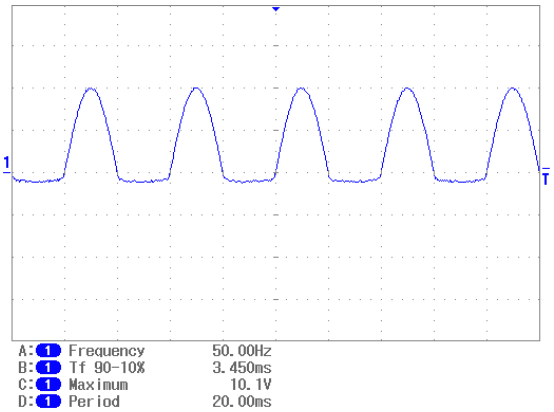
Abbildung 5.1: Schaltplan des Optokopplers

Am Schluss wird das digitalisierte Signal an den Eingangspin des ATMega16-Mikrocontrollers weitergeleitet. Wenn eine fallende Flanke am Pin des Mikrocontrollers eintrifft, wird der Timer-Interrupt ausgelöst, in dem der Timer-Wert in eine vordefinierte Variable gespeichert wird. Im nächsten Interrupt wird die Differenz zwischen zwei aufeinanderfolgenden fallenden Flanken gespeichert. In der Firmware wird ein Ringpuffer[1] erstellt, indem die gemessenen Zeitdifferenzen sich befinden. Bei der Abfrage der Frequenz aus dem UNIX-Server werden alle Zeitdifferenzen

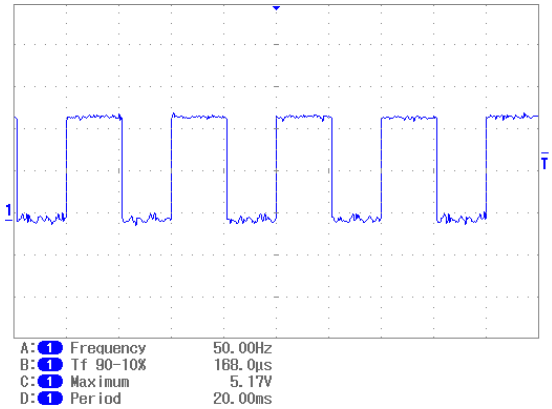
[1] ist ein Datenspeicher mit N Datenelementen, wobei der Zeiger ringförmig auf die Datenelemente zeigt.



(a) Sinusförmiges Signal vom Frequenzgenerator



(b) Eingangssignal des Optokopplers



(c) Digitalisiertes Ausgangssignal des Optokopplers

Abbildung 5.2: Eingangs- und Ausgangssignale des Optokopplers

aus dem Ringpuffer ausgelesen und der Mittelwert berechnet, wie in der folgenden Formel beschrieben wird:

$$Mittelwert = \frac{\sum_{i=1}^{N} Puffer[i]}{N} \tag{5.1}$$

wobei N die Länge des Ringpuffers ist. Um die Mittelfrequenz zu berechnen, wird die Taktfrequenz des ATMega16-Mikrocontrollers verwendet, wie in der folgenden Formel beschrieben wird:

$$Mittelfrequenz = \frac{F\_CPU}{Mittelwert * Timer\_Vorteiler} \tag{5.2}$$

wobei F_CPU mit 16 MHz bestimmt ist, und der $Timer\_Vorteiler$ als 256 definiert wurde.

Nach der Kompilierung der Firmware sieht die Ausgabe folgendemaßen aus:

```
 --- --- --- Target  Information --- --- ---
      AVR Model       : atmega16
      Board           : Frequenzmessung
      MCU Frequency   : 16000000 Hz
 --- --- --- --- --- --- --- --- --- --- ---

Size after:
===========================
main.elf  :
section            size      addr
.text              5970         0
.data                12   8388704
.bss                211   8388716
.comment             17         0
.debug_aranges      456         0
.debug_info        9127         0
.debug_abbrev      2740         0
.debug_line        2493         0
.debug_frame       1600         0
.debug_str         1127         0
.debug_loc         5429         0
.debug_ranges        48         0
Total             29230

AVR Memory Usage
----------------
Device: atmega16

Program:    5982 bytes (36.5% Full)
(.text + .data + .bootloader)

Data:        223 bytes (21.8% Full)
(.data + .bss + .noinit)
```

Mit dem `ping` -Programm wird eine Abfrage aus dem UNIX-Server an den ENC28J60-Mikrochip gesendet, ob er im Netz verfügbar ist. Es wurde in dieser Arbeit kein DHCP-Protokoll erstellt. Aus diesem Grund muss die IP-Adresse des Mikrochips in der Firmware manual eingetragen werden. Die IP-Adresse ist in der `main` -Datei als ”192.168.0.3” definiert. Wenn der Mikrochip eine richtige IP-Adresse im lokalen Netz besitzt, dann liefert der Mikrochip nach der `ping` -Abfrage folgende Antwort zurück:

```
$ ping -c 5 192.168.0.3
PING 192.168.0.3 (192.168.0.3): 56 data bytes
64 bytes from 192.168.0.3: icmp_seq=0 ttl=64 time=0.788 ms
64 bytes from 192.168.0.3: icmp_seq=1 ttl=64 time=0.438 ms
64 bytes from 192.168.0.3: icmp_seq=2 ttl=64 time=0.464 ms
64 bytes from 192.168.0.3: icmp_seq=3 ttl=64 time=0.529 ms
64 bytes from 192.168.0.3: icmp_seq=4 ttl=64 time=0.587 ms

--- 192.168.0.3 ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.438/0.561/0.788/0.125 ms
```

Wenn aber der Mikrochip keine gültige lokale IP-Adresse definiert hat, liefert der Mikrochip nach der `ping` -Abfrage folgende *Zeitüberschreitungsmeldung*:

```
$ ping -c 5 192.168.0.3
PING 192.168.0.3 (192.168.0.3): 56 data bytes
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
Request timeout for icmp_seq 2
Request timeout for icmp_seq 3
Request timeout for icmp_seq 4

--- 192.168.0.3 ping statistics ---
5 packets transmitted, 0 packets received, 100.0% packet loss
```

Mit dem UNIX-Dämon wird eine Abfrage über das UDP-Protokoll gesendet, um die Mittelfrequenz aus dem ENC28J60-Mikrochip zu erhalten. Der Dämon hat zwei Argumente, IP-Adresse und die Portnummer. Die Argumentbehandlung wird im folgenden Code-Feld beschrieben:

```
$ ./daemon
Usage:
        ./daemon -h <hostname> -p <port>
        Example: ./daemon -h FreeBSD.local -p 32000
```

Wie schon beschrieben wurde, wurde in der Firmware die IP-Adresse ”192.168.0.3” und die Portnummer ”1200” definiert. Aus diesem Grund wird der UNIX-Dämon wie in dem folgenden Code-Feld verwendet:

```
$ ./daemon -h 192.168.0.3 -p 1200
hostname: 192.168.0.3
port number: 1200
sending: start
waiting for packet ...
received packet from 192.168.0.3: 50.153 Hz
```

Wie in der letzten Zeile des obigen Code-Felds zu sehen ist, hat der Mikrochip nach der Abfrage die Mittelfrequenz als *50.153 Hz* gesendet bzw. beantwortet.

6 Fazit

Es werden sowohl elektrotechnische als auch hardwarenahe Kentnisse für diese Arbeit vorausgesetzt. Außerdem werden spezifische Kentnisse über Funktionalität von Computer-Netzwerken benötigt.

Das Thema war sehr interessant. Nachdem jedoch die unterste Ebene angesprochen wird, wie beispielsweise die TCP/IP-Stacks für die Kommunikationsprotokolle, war die Umsetzung umso komplizierter. Eine große Hilfe bieten die RFC-Dokumente für die Kommunikationsprotokolle.

Die erste Hürde dieser Arbeit war der Entwurf des Boards, das auf einem Optokoppler, dem ATmega16-Mikrocontroller und auf dem ENC28J60-Mikrochip basiert. Danach wurde das kontinuierlich-sinusförmige Signal mit Hilfe des Optokopplers digitalisiert. Um die Frequenzen vom digitalisierten Signal zu messen, wurde der Timer-Treiber im CTC-Modus mit Input-Capture-Betrieb für den Mikrocontroller geschrieben. Der ENC28J60-Mikrochip wurde über die SPI-Schnittstelle angesteuert. Anschließend wurden ARP-, MAC- bzw. TCP/IP-Stacks realisiert. Die Eingangs- bzw. Ausgangssignale wurden schrittweise nach jeder Operation mit dem Oszilloskop gemessen. Zur Fehlerbehebung bzw. zum Debuggen wurden Leds verwendet, um zu sehen, ob das Programm bzw. die Firmware auf dem richrigen Pfad läuft. Nach der Abfrage, wie groß der Frequenzwert ist, wurde der mittlere Frequenzwert von letzten zehn Frequentwerte ausgerechnet und als Antwort an den Server übermittelt.

7 Zukünftige Entwicklung

Das Projekt ist so umfangreich, dass es mit Zusatzfunktionen ausgestattet werden kann. Beispielsweise könnte ein DHCP-Dienst realisiert werden, sodass der ENC28J60-Mikrochip eine automatische Beziehung der IP-Adresse vom DHCP-Server annimmt, um eine manuelle Eingabe der IP-Adresse zu automatisieren.

Eine weitere Entwicklung wäre ein NTP-Client. Dieser ermöglicht die Sammlung der Datenpakete von verschiedenen Standorten. Jedes Datenpaket wird mit einem lokalen Zeitstempel versehen. Anhand des Zeitstempels kann der Standort des Datenpakets eruiert werden. Sollten jedoch Datenpakete aus mehreren Standorte in der gleichen Zeitzone ankommen, müssen die Datenpakete mit einer weiteren Kennung gestempelt werden, damit der Server eindeutig identifizieren kann, aus welchem Standort das Datenpaket stammt. Um dieses zu realisieren, müsste jedes Board an verschiedenen Standorten mit einem RTC-Chip ausgestattet sein.

LITERATURE

[Bau13] BAUN, Christian: *Computernetze kompakt*. 2. aktualisierte und erweiterte Auflage. Fachhochschule Frankfurt am Main Frankfurt, Deutschland : Springer-Verlag Berlin Heidelberg, 2013. – 122 S. – ISBN 978–3–642–41653–8

[Dem12] DEMBOWSKI, Klaus: *Computernetzwerke*. 1. Aufl. Addison-Wesley Verlag, 2012. – ISBN 978–3–8273–3092–5

[Jon02] JONES, M. T.: *TCP/IP Application Layer Protocols for Embedded Systems*. 1st Edition. Charles River Media, 2002. – 5–19 S. – ISBN 1–58450–247–9

[MSC10] MAHATO, B. ; SHARMA, S. ; CHITRANSHI, G.: An embedded web controllable heater interface for industry application. In: *India Conference (INDICON), 2010 Annual IEEE*, 2010, S. 1–4

[OWN96] OPPENHEIM, Alan V. ; WILLSKY, Alan S. ; NAWAB, S. H.: *Signals &Amp; Systems (2Nd Ed.)*. Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 1996. – ISBN 0–13–814757–4

[Sch08] SCHMITT, Günter: *Mikrocomputertechnik mit Controllern der Atmel AVR-RISC-Familie - Programmierung in Assembler und C- Schaltungen und Anwendungen*. korrigierte Auflage. München : Oldenbourg Verlag, 2008. – ISBN 978–3–486–58790–6

[SF12] STEVENS, W. R. ; FALL, Kevin: *TCP/IP Illustrated, Volume 1: The Protocols*. 2nd. USA : Addison Wesley Longman Publishing Co., Inc., 2012 (Addison-Wesley Professional Computing Series). – ISBN 0321336313, 9780321336316

[SFR04] STEVENS, W. R. ; FENNER, Bill ; RUDOFF, Andrew M.: *UNIX Network Programming, Volume 1: The Sockets Networking API*. Boston (Mass.) : Addison-Wesley, 2004 (Addison-Wesley Professional Computing Series). – 86–212 S. – ISBN 0–13–141155–1

[SR13] STEVENS, W. R. ; RAGO, Stephen A.: *Advanced Programming in the UNIX Environment*. 3rd. Addison-Wesley, 2013 (Addison-Wesley Professional Computing Series). – ISBN 0321637739, 9780321637734

[Ste96] STEVENS, W. R.: *TCP/IP Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP, and the UNIX Domain Protocols*. Redwood City, CA, USA : Addison Wesley Longman Publishing Co., Inc., 1996 (Addison-Wesley Professional Computing Series). – ISBN 0–201–63495–3

[TS02] TIETZE, Ulrich ; SCHENK, Christoph: *Halbleiter-Schaltungstechnik - [neuer Teil: Nachrichtentechnische Schaltungen]*. 12. Aufl. Berlin : Springer DE, 2002. – ISBN 978–3–540–42849–7

[TW10] TANENBAUM, Andrew S. ; WETHERALL, David J.: *Computer Networks*. 5th. Upper Saddle River, NJ, USA : Prentice Hall Press, 2010. – ISBN 0132126958, 9780132126953

[WP09] WILFRIED PLASSMANN, Detlef S.: *Handbuch Elektrotechnik: Grundlagen und Anwendungen für Elektrotechniker*. 5., korrigierte Auflage. Vieweg+Teubner Verlag, 2009. – ISBN 978–3–8348–0470–9

WEBLINKS

[1] Atmel corporation. http://www.atmel.com.

[2] Avrdude - avr downloader/uploader. http://www.nongnu.org/avrdude/.

[3] Belegung rj45-stecker für ethernet. http://www.elektronik-kompendium.de/sites/net/0510151.htm.

[4] Cadsoft eagle pcb design software. http://www.cadsoft.de.

[5] Enc28j60 - ethernet controllers. http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en022889.

[6] Freebsd man pages. http://www.freebsd.org/cgi/man.cgi.

[7] Frequently asked questions. http://www.faqs.org/rfcs/.

[8] Ieee 802.3 ethernet working group. http://www.ieee802.org/3/.

[9] Microchip technology inc. http://www.microchip.com.

[10] Rfc editor. http://www.rfc-editor.org/.

[11] Stand-alone ethernet controller with spi interface. http://ww1.microchip.com/downloads/en/DeviceDoc/39662e.pdf.

[12] Atmel. 8-bit avr microcontroller with 16k bytes in-system programmable flash, atmega16, atmega16l, rev. 2466t–avr–07/10. http://www.atmel.com/Images/doc2466.pdf.

[13] Atmel. 8-bit avr microcontrollers, jtagice mkii quick start guide, rev. 2562c-avr-07/06. http://www.atmel.com/Images/doc2562.pdf.

[14] Atmel. Avr jtagice mkii. http://www.atmel.com/tools/avrjtagicemkii.aspx.

[15] Atmel. Jtagice mkii user's guide. http://support.atmel.com/knowledgebase/avrstudiohelp/mergedProjects/JTAGICEmkII/mkII/JTAGICEmkII.htm.

[16] N. Developers. Avrdude user manual. http://www.nongnu.org/avrdude/user-manual/avrdude_toc.html.

[17] Grüninger. Induktion durch drehen einer spule. http://www.schule-bw.de/unterricht/faecher/physik/online_material/e_lehre_2/induktion/drehspule.htm.

[18] D.-I. M. Hämmerling. Engbedded. http://www.engbedded.com/fusecalc.

[19] S. Knight. Ethernet numbers. http://www.iana.org/assignments/ethernet-numbers/ethernet-numbers.xhtml.

[20] Mikrocontroller.net. Optokoppler. http://www.mikrocontroller.net/articles/Optokoppler.

[21] F. Opatz. Netzwerkprogrammierung mit bsd-sockets. http://www.zotteljedi.de/socket-buch/index.html.

[22] P. J. Plate. Grundlagen computernetze. http://www.netzmafia.de/skripten/netze/.

[23] J. Postel. Internet control message protocol, std 5, rfc 792, september 1981. http://www.rfc-editor.org/info/rfc792.

[24] J. Postel. Internet protocol, std 5, rfc 791, september 1981. http://www.rfc-editor.org/info/rfc791.

[25] J. Postel. Transmission control protocol, std 7, rfc 793, september 1981. http://www.rfc-editor.org/info/rfc793.

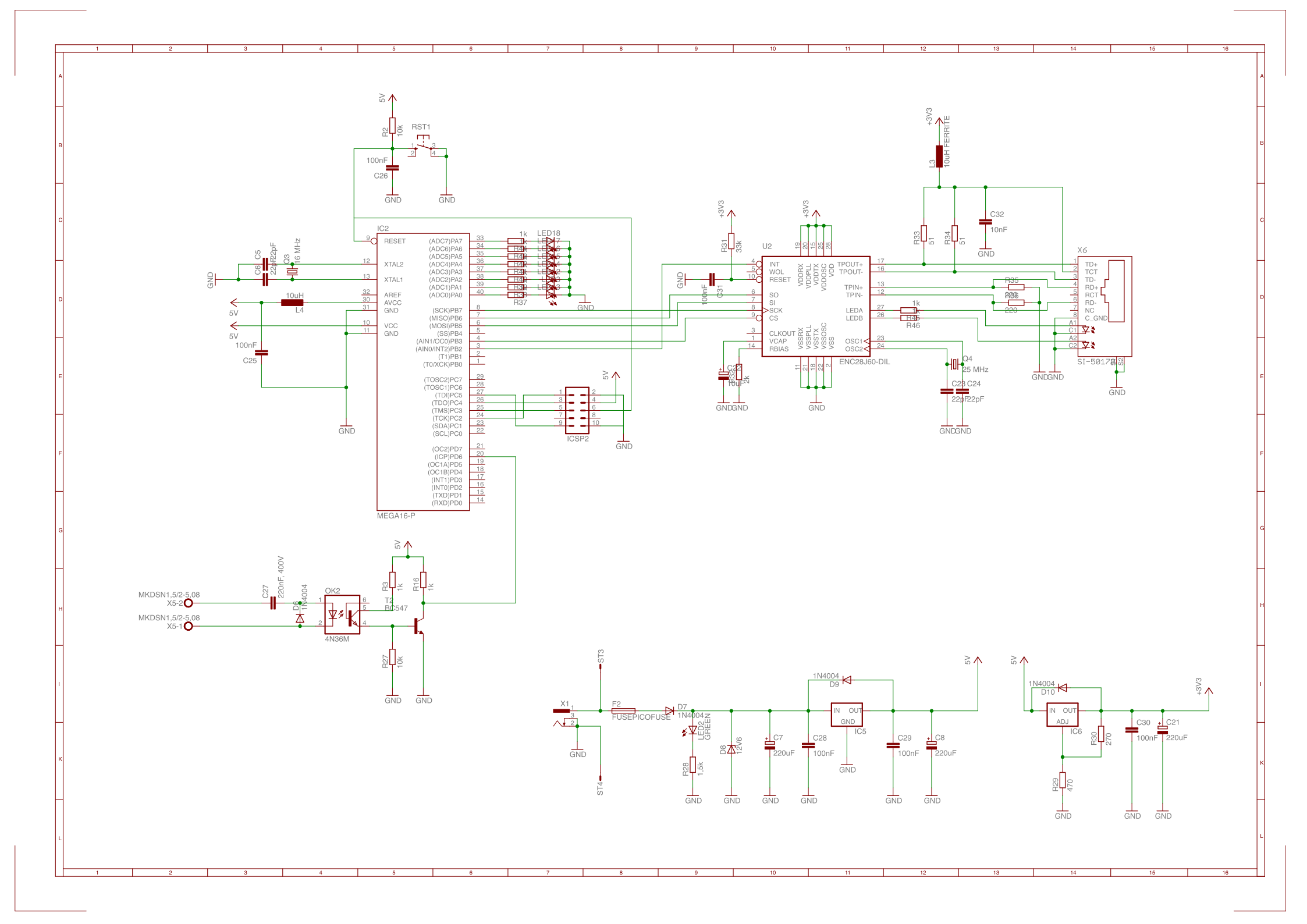
[26] J. Postel. User datagram protocol, std 6, rfc 768, august 1980. http://www.rfc-editor.org/info/rfc768.

[27] T. F. D. Project. Freebsd developers' handbook. http://www.freebsd.org/doc/en/books/developers-handbook/.

[28] G. Socher. Diy projects: Avr microcontroller electronics. http://tuxgraphics.org/electronics/.

[29] R. Watson. Freebsd and linux kernel cross-reference. http://fxr.watson.org/.

A Anhang


5V
R2
10k
RST1
100nF
C26
GND
GND
IC2
RESET
XTAL2
XTAL1
AREF
AVCC
GND
VCC
GND
C5
C6
22pF
22pF
Q3
16 MHz
10uH
L4
5V
5V
100nF
C25
GND
(ADC7)PA7
(ADC6)PA6
(ADC5)PA5
(ADC4)PA4
(ADC3)PA3
(ADC2)PA2
(ADC1)PA1
(ADC0)PA0
(SCK)PB7
(MISO)PB6
(MOSI)PB5
(SS)PB4
(AIN1/OC0)PB3
(AIN0/INT2)PB2
(T1)PB1
(T0/XCK)PB0
(TOSC2)PC7
(TOSC1)PC6
(TDI)PC5
(TDO)PC4
(TMS)PC3
(TCK)PC2
(SDA)PC1
(SCL)PC0
(OC2)PD7
(ICP)PD6
(OC1A)PD5
(OC1B)PD4
(INT1)PD3
(INT0)PD2
(TXD)PD1
(RXD)PD0
MEGA16-P
1k
LED18
R37
GND
ICSP2
5V
GND
+3V3
R31
33k
GND
100nF
C31
U2
INT
WOL
RESET
SO
SI
SCK
CS
CLKOUT
VCAP
RBIAS
VDDRX
VDDPLL
VDDTX
VDDOSC
VDD
VSSRX
VSSPLL
VSSTX
VSSOSC
VSS
TPOUT+
TPOUT-
TPIN+
TPIN-
LEDA
LEDB
OSC1
OSC2
ENC28J60-DIL
+3V3
+3V3
10uF
2k
GND
GND
GND
+3V3
L3
10uH FERRITE
R33
51
R34
51
C32
10nF
GND
R35
220
R36
220
1k
R45
R46
Q4
25 MHz
C23
22pF
C24
22pF
GND
GND
GND
GND
X6
TD+
TCT
TD-
RD+
RCT
RD-
NC
C_GND
SI-50170
GND
MKDSN1,5/2-5.08
X5-2
MKDSN1,5/2-5.08
X5-1
C27
220nF 400V
D6
1N4004
OK2
4N36M
5V
R3
1k
R16
1k
T2
BC547
R27
10k
GND
GND
ST3
X1
GND
ST4
F2
FUSEPICOFUSE
D7
1N4004
LED2
GREEN
R28
1,5k
GND
D8
12V6
GND
C7
220uF
GND
C28
100nF
GND
1N4004
D9
IN
OUT
GND
IC5
GND
C29
100nF
GND
C8
220uF
GND
5V
5V
1N4004
D10
IN
OUT
ADJ
IC6
R30
270
R29
470
GND
C30
100nF
GND
C21
220uF
GND
+3V3

B Anhang

Der Firmware-Code befindet sich in mehreren Verzeichnissen, damit er strukturell noch übersichtlicher ist und die Firmware in Zukunft noch leichter weiter entwickelt werden kann. Unter dem *board*-Verzeichnis befindet sich die Header-Datei, in der die PIN-Belegungen des entworfenen Boards definiert wird. Die Definition des Registers des ATMega16-Mikrocontrollers und des ENC28J60-Mikrochips sind unter dem Verzeichnis *device* zu finden. Die geschriebenen Treiber befinden sich unter dem Verzeichnis *driver*. Um die Zeitmessungswerte zu puffern, gibt es einen *Ringpuffer* im *lib*-Verzeichnis. Schlussendlich stehen die nützliche Definitionen wie *boolean*-Werte und *NULL-Pointer*, falls sie im Compiler nicht definiert ist, im *include*-Verzeichnis zur Verfügung. Die Gesamtstruktur ist in der folgenden Liste zu lesen:

```
firmware/
        |-- board/
        |   `-- freq.h
        |
        |-- device/
        |   |-- atmega16.h
        |   `-- enc28j60.h
        |
        |-- driver/
        |   |-- enc28j60.c
        |   |-- icp.c
        |   |-- icp.h
        |   |-- leds.c
        |   |-- leds.h
        |   |-- net.c
        |   |-- net.h
        |   |-- spi.c
        |   |-- spi.h
        |   |-- timer0.c
        |   `-- timer0.h
        |
        |-- include/
        |   |-- common.h
        |   `-- stdbool.h
        |
        |-- lib/
        |   |-- buffer.c
        |   `-- buffer.h
        |
        |-- AUTHORS
        |-- BSDmakefile
        |-- GNUmakefile
        |-- LICENSE
        |-- Makefile
        |-- Makefile.mk
        |-- README
        |-- config.mk
        |-- doxy.file
        `-- main.c
```

Der Firmware-Code und der UNIX-Dämon-Code ist als Open Source Software unter der **BSD**-Lizenz[1] lizenziert bzw. veröffentlicht:

```
Copyright © 2013, M. Ozgan. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted
provided that the following conditions are met:

    o Redistributions of source code must retain the above copyright notice, this list of
      conditions and the following disclaimer.
    o Redistributions in binary form must reproduce the above copyright notice, this list of
      conditions and the following disclaimer in the documentation and/or other materials
      provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

[1] ist eine Lizenz für freie Software. Solche Software darf kostenfrei auch für kommerzielle Zwecke verwendet, verändert und vertrieben werden.

B.1 Firmware-Code

main-Programm

main.c

```
/*-
 * Copyright (c) 2013, mozgan.
 * All Rights Reserved with BSD License.
 * Read LICENSE file.
 */

/**
 * @file    main.c
 * @brief    main program
 *
 * @author    M. Ozgan, <mozgan@mozgan.org>
 * @version    0.5
 * @date    19.08.2013 15:20:15
 * @internal
 *    $Compiler:    gcc (on Mac, and 4.4BSD) $
 *    $Company:    TU Wien $
 *
 *  @bug    51. signal has not readed!
 *  @todo    none
 */

/*
 *    @(#) main.c        TU Wien 19.08.2013
 *  $Id: main.c,v 0.5 19.08.2013 15:20:15 mozgan Exp $
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <avr/io.h>
#include <util/delay.h>

#include <include/common.h>
#include <board/freq.h>
#include <lib/buffer.h>

#include <driver/leds.h>
#include <device/enc28j60.h>
#include <driver/net.h>
#include <driver/icp.h>

#define        BUF_SIZE    150

static uint8_t buf[BUF_SIZE];

uint8_t mymac[6] = {0xab,0xbc,0x6f,0x55,0x1c,0xc2};
uint8_t myip[4] = {192, 168, 0, 3};
uint16_t myport = 1200;

volatile uint16_t plen;

volatile uint16_t new, old;
volatile uint16_t diff;
volatile float freq;

volatile int snd = 0;

/**
* @brief return-function from input capture interrupt routine
*
* @param icr input capture timer state
*/
void
trigger(uint16_t icr)
{
    old = new;
    new = icr;

    if (new > old) {
        buffer_write((new - old));

        if (snd < LEN)
```

```
74                snd++;
75            }
76
77            /* debug */
78        #ifdef DEBUG
79            led_toggle(0);
80        #endif
81        }
82
83        /**
84        * @brief main function (program start)
85        *
86        * @return returns zero if SUCCESS, otherwise non-zero
87        */
88        int
89        main(void)
90        {
91            buffer_init();        /* initialize the buffer ring */
92            leds_init();        /* leds for debuging */
93
94            /* initialize the ENC28J60 device with the physical mac address */
95            enc_init(mymac);
96            _delay_ms(100);
97            enc_init_phy();
98
99            /* initialize the ip address for the given mac address */
100           net_init(mymac, myip);
101
102           /* initialize the input capture pin and the timer/counter1 for capturing */
103           icp_init(trigger);
104
105           /* enable the interrupt routine */
106           sei();
107
108           while (TRUE) {
109               /* read a packet from ENC28J60 device */
110               plen = enc_recv_packet(BUF_SIZE, buf);
111
112               if (plen == 0)
113                   continue;
114
115               if (eth_arp(buf, plen)) {
116                   arp_answer(buf, plen);
117                   continue;
118               }
119
120               if (eth_ip(buf, plen) == 0)
121                   continue;
122
123               /* if there is a ping, then send a pong */
124               if ((buf[IP_PROTO_P] == IP_PROTO_ICMP_V) &&
125                    buf[ICMP_TYPE_P] == ICMP_REQUEST_V) {
126                   echo_reply(buf, plen);
127                   continue;
128               }
129
130               /*
131                * if received a UDP protocol, as we waiting it,
132                * then send the frequency from ring buffer
133                */
134               if (buf[IP_PROTO_P] == IP_PROTO_UDP_V) {
135                   if (snd == LEN) {
136                       char str[12];
137                       static float diff;
138
139                       diff = buffer_medium();
140                       freq = (float)F_CPU / diff;
141                       freq /= 256;
142
143                       dtostrf((float)freq, 12, 3, str);
144
145                       udp_reply(buf, str, strlen(str), myport);
146                   }
147               }
148           }
149
150           return 0;
151       }
```

board-Verzeichnis

freq.h

```
/*-
 * Copyright (c) 2013, mozgan.
 * All Rights Reserved with BSD License.
 * Read LICENSE file.
 */

/**
 * @file    freq.h
 * @brief    pin locations for "Frequenzmessung-Board"
 *
 * @author    M. Ozgan, <mozgan@mozgan.org>
 * @version    0.2
 * @date    19.08.2013 15:13:45
 * @internal
 *    $Compiler:    gcc (on Mac, GNU/Linux and 4.4BSD) $
 *    $Company:    TU Wien $
 *
 *  @bug    none
 *  @todo    none
 */

/*
 *    @(#) board/freq.h        TU Wien 19.08.2013
 *  $Id: freq.h,v 0.2 19.08.2013 15:13:45 mozgan Exp $
 */

#ifndef __FREQ_H__
#define __FREQ_H__  1

#include <device/atmega16.h>

/* LEDs for debuging */
#define        LED_DDR        DDRA
#define        LED_PORT        PORTA
#define        LED_PIN        PINA

#define        LED0        PA0
#define        LED1        PA1
#define        LED2        PA2
#define        LED3        PA3
#define        LED4        PA4
#define        LED5        PA5
#define        LED6        PA6
#define        LED7        PA7

#define        LED_ALL        0xFF
#define        LED_OFF        0x00

/* SPI */
#define        SPI_DDR        DDRB
#define        SPI_PORT        PORTB
#define        SPI_SS        PB4
#define        SPI_MOSI        PB5
#define        SPI_MISO        PB6
#define        SPI_SCK        PB7

/* ENC28J60 */
#define        ENC_CTRL_DDR    DDRB
#define        ENC_CTRL_PORT   PORTB
#define        ENC_CTRL_CS        PB3


#endif /* __FREQ_H__ */
```

device-Verzeichnis

atmega16.h

```
/*-
 * Copyright (c) 2013, mozgan.
 * All Rights Reserved with BSD License.
 * Read LICENSE file.
 */

```

```
/**
 * @file    atmega16.h
 * @brief    specific definitions for ATMega16 device
 *
 * @author    M. Ozgan, <mozgan@mozgan.org>
 * @version    0.3
 * @date    19.08.2013 15:18:15
 * @internal
 *    $Compiler:    gcc (on Mac, and 4.4BSD) $
 *    $Company:    TU Wien $
 *
 *  @bug    none
 *  @todo    none
 */

/*
 *    @(#) device/atmega16.h        TU Wien 19.08.2013
 *  $Id: atmega16.h,v 0.3 19.08.2013 15:18:15 mozgan Exp $
 */

#ifndef __ATMEGA16_H__
#define __ATMEGA16_H__    1

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#include <avr/iom16.h>            /* avr-gcc library */

/*
 * If cpu freq. in Makefile not set, define it 16 MHz.
 * (Compiler option: -DF_CPU=16000000)
 */
#ifndef F_CPU
#warning    "F_CPU not yet set!"
#define F_CPU 16000000            /* CPU FREQ: 16 MHz */
#endif    /* F_CPU */

/*****************************************************************************
 *        Timer/Counter Registers
 *****************************************************************************/

/*
 * TIMSK - Timer/Counter Interrupt Mask Register
 * +-----------------------------------------------------------------+
 * | OCIE2 | TOIE2 | TICIE1 | OCIE1A | OCIE1B | TOIE1 | OCIE0 | TOIE0 |
 * +-----------------------------------------------------------------+
 */
#define        TIMER_INT_MASK  TIMSK

/*
 * TIFR - Timer/Counter Interrupt Flag Register
 * +-----------------------------------------------------------+
 * | OCF2 | TOV2 | ICF1 | OCF1A | OCF1B | TOV1 | OCF0 | TOV0 |
 * +-----------------------------------------------------------+
 */
#define        TIMER_INT_FLG   TIFR

/*****************************************************************************
 * 8-bit Timer/Counter 0
 */

#define        TIMER0_CNT        TCNT0
#define        TIMER0_OCR        OCR0

/*
 * TCCR0 - Timer/Counter Control Register
 * +-----------------------------------------------------------+
 * | FOC0 | WGM00 | COM01 | COM00 | WGM01 | CS02 | CS01 | CS00 |
 * +-----------------------------------------------------------+
 */
#define        TIMER0_TCCR        TCCR0

/*
 * Timer/Counter 0 Clock Select
 */
#define        TIMER0_NO_CLK    (TIMER0_TCCR & 0xF8)    /*(~((1 << CS00) (1 << CS01)
#define        TIMER0_PRES_1    (TIMER0_NO_CLK | (1 << CS00))
#define        TIMER0_PRES_8    (TIMER0_NO_CLK | (1 << CS01))
#define        TIMER0_PRES_64    (TIMER0_NO_CLK | (1 << CS00) | (1 << CS01))
```

```
87  #define         TIMER0_PRES_256     (TIMER0_NO_CLK | (1 << CS02))
88  #define         TIMER0_PRES_1024     (TIMER0_NO_CLK | (1 << CS00) |
89  #define         TIMER0_T0_FALL     (TIMER0_NO_CLK | (1 << CS01) | (
90  #define         TIMER0_T0_RIS     (TIMER0_NO_CLK | (1 << CS00) | (1
91
92  /*
93   * CTC-Mode & non-PWM
94   */
95  #define         TIMER0_NORMAL_P     (TIMER0_TCCR & 0x3F)
96  #define         TIMER0_CTC         (TIMER0_NORMAL_P | (1 << WGM01))
97
98  /*****************************************************************************
99   * 16-bit Timer/Counter 1
100  */
101 #define         TIMER1_CNT         TCNT1
102 #define         TIMER1_OCRA         OCR1A
103 #define         TIMER1_ICR         ICR1
104
105 /*****************************************************************************
106  * SPI
107  */
108
109 #define         MASTER         0x01
110 #define         SLAVE         0x02
111
112 #define         SPI_DATA_REG     SPDR
113
114 /*
115  * SPRC - SPI Control Register
116  * +-------------------------------------------------------+
117  * | SPIE | SPE | DORD | MSTR | CPOL | CPHA | SPR1 | SPR0 |
118  * +-------------------------------------------------------+
119  */
120 #define     SPI_CONTROL_REG         SPCR
121
122 #define     SPI_ENABLE         SPE
123 #define     SPI_INT_ENABLE         SPIE
124 #define     SPI_MASTER         MSTR
125
126 /*
127  * SPSR - SPI Status Register
128  * +-------------------------------------------------------+
129  * | SPIF | WCOL | ---- | ---- | ---- | ---- | ---- | SPI2X |
130  * +-------------------------------------------------------+
131  */
132 #define     SPI_STATUS_REG         SPSR
133
134 #define     SPI_INT_FLAG         SPIF
135 #define     SPI_DOUBLE_SPEED     SPI2X
136
137
138 /*****************************************************************************
139 #endif /* __ATMEGA16_H__ */
```

enc28j60.h

```
1   /*-
2    * Copyright (c) 2013, mozgan.
3    * All Rights Reserved with BSD License.
4    * Read LICENSE file.
5    */
6
7   /**
8    * @file     enc28j60.h
9    * @brief     registers of ENC28J60 device
10   *
11   * @author     M. Ozgan, <mozgan@mozgan.org>
12   * @version     1.0
13   * @date     19.08.2013 15:23:13
14   * @internal
15   *     $Compiler:     gcc (on Mac, and 4.4BSD) $
16   *     $Company:     TU Wien $
17   *
18   *  @bug     none
19   *  @todo     none
20   */
21
22  /*
23   *     @(#) device/enc28j60.h         TU Wien 19.08.2013
```

```
 *  $Id: enc28j60.h,v 1.0 19.08.2013 15:23:13 mozgan Exp $
 */

#ifndef __ENC28J60_H__
#define __ENC28J60_H__    1

#include <board/freq.h>
#include <include/common.h>

#include <inttypes.h>

/*
 * ENC28J60 Control Registers
 *
 * Control registers are generically grouped as ETH, MAC and MII registers.
 * Register names starting with 'E' belong "ETH" group. Similarly, registers names starting with 'M' belong to the "MAC" group and registers prefixed with
 *
 * - Register address        (bits 0-4)
 * - Bank number         (bits 5-6)
 * - MAC/PHY indicator        (bit 7)
 */

#define        ADDR_MASK        0x1F
#define        BANK_MASK        0x60
#define        SPRD_MASK        0x80

/*
 * EIE: Ethernet Interrupt Enable Register - Address: 0x1B
 *
 * +-------------------------------------------------------------------+
 * | INTIE | PKTIE | DMAIE | LINKIE | TXIE |  r  | TXERIE | RXERIE |
 * +-------------------------------------------------------------------+
 */
#define        EIE            0x1B

#define        INTIE        0x80
#define        PKTIE        0x40
#define        DMAIE        0x20
#define        LINKIE        0x10
#define        TXIE        0x08
#define        EIE_WOLIE        0x04
#define        TXERIE        0x02
#define        RXERIE        0x01

/*
 * EIR: Ethernet Interrupt Request (Flag) Register - Address: 0x1C
 *
 * +-------------------------------------------------------------------+
 * |  -  | PKTIF | DMAIF | LINKIF | TXIF |  r  | TXERIF | RXERIF |
 * +-------------------------------------------------------------------+
 */
#define        EIR            0x1C

#define        PKTIF        0x40
#define        DMAIF        0x20
#define        LINKIF        0x10
#define        TXIF        0x08
#define        EIR_WOLIF        0x04
#define        TXERIF        0x02
#define        RXERIF        0x01

/*
 * ESTAT: Ethernet Status Register - Address: 0x1D
 *
 * +-------------------------------------------------------------------+
 * | INT | BUFER |  r  | LATECOL |  -  | RXBUSY | TXABRT | CLKRDY |
 * +-------------------------------------------------------------------+
 */
#define        ESTAT        0x1D

#define        INT            0x80
#define        BUFER        0x40
#define        ESTAT_R        0x20
#define        LATECOL        0x10
#define        RXBUSY        0x04
#define        TXABRT        0x02
#define        CLKRDY        0x01

/*
 * ECON2: Ethernet Control Register 2 - Address: 0x1E
```

```
104   *
105   * +-------------------------------------------------------+
106   * | AUTOINC | PKTDEC | PWRSV |  r  | VRPS |  -  |  -  |  -  |
107   * +-------------------------------------------------------+
108   */
109  #define        ECON2        0x1E
110
111  #define        AUTOINC        0x80
112  #define        PKTDEC        0x40
113  #define        PWRSV        0x20
114  #define        ECON2_R        0x10
115  #define        VRPF        0x08
116
117  /*
118   * ECON1: Ethernet Control Register 1 - Address: 0x1F
119   *
120   * +-----------------------------------------------------------------+
121   * | TXRST | RXRST | DMAST | CSUMEN | TXRTS | RXEN | BSEL1 | BSEL0 |
122   * +-----------------------------------------------------------------+
123   */
124  #define        ECON1        0x1F
125
126  #define        TXRST        0x80
127  #define        RXRST        0x40
128  #define        DMAST        0x20
129  #define        CSUMEN        0x10
130  #define        TXRTS        0x08
131  #define        E_RXEN        0x04
132  #define        BSEL1        0x02
133  #define        BSEL0        0x01
134
135
136
137  /* Bank 0 Registers */
138  #define        BANK0        (0x00)
139
140  #define        ERDPTL        (0x00 | BANK0)
141  #define        ERDPTH          (0x01 | BANK0)
142  #define        EWRPTL          (0x02 | BANK0)
143  #define        EWRPTH          (0x03 | BANK0)
144  #define        ETXSTL          (0x04 | BANK0)
145  #define        ETXSTH          (0x05 | BANK0)
146  #define        ETXNDL          (0x06 | BANK0)
147  #define        ETXNDH          (0x07 | BANK0)
148  #define        ERXSTL          (0x08 | BANK0)
149  #define        ERXSTH          (0x09 | BANK0)
150  #define        ERXNDL          (0x0A | BANK0)
151  #define        ERXNDH          (0x0B | BANK0)
152  #define        ERXRDPTL        (0x0C | BANK0)
153  #define        ERXRDPTH        (0x0D | BANK0)
154  #define        ERXWRPTL        (0x0E | BANK0)
155  #define        ERXWRPTH        (0x0F | BANK0)
156  #define        EDMASTL         (0x10 | BANK0)
157  #define        EDMASTH         (0x11 | BANK0)
158  #define        EDMANDL         (0x12 | BANK0)
159  #define        EDMANDH         (0x13 | BANK0)
160  #define        EDMADSTL        (0x14 | BANK0)
161  #define        EDMADSTH        (0x15 | BANK0)
162  #define        EDMACSL         (0x16 | BANK0)
163  #define        EDMACSH         (0x17 | BANK0)
164
165
166
167  /* Bank 1 Registers */
168  #define        BANK1        (0x20)
169
170  #define        EHT0            (0x00 | BANK1)
171  #define        EHT1            (0x01 | BANK1)
172  #define        EHT2            (0x02 | BANK1)
173  #define        EHT3            (0x03 | BANK1)
174  #define        EHT4            (0x04 | BANK1)
175  #define        EHT5            (0x05 | BANK1)
176  #define        EHT6            (0x06 | BANK1)
177  #define        EHT7            (0x07 | BANK1)
178  #define        EPMM0           (0x08 | BANK1)
179  #define        EPMM1           (0x09 | BANK1)
180  #define        EPMM2           (0x0A | BANK1)
181  #define        EPMM3           (0x0B | BANK1)
182  #define        EPMM4           (0x0C | BANK1)
183  #define        EPMM5           (0x0D | BANK1)
```

```
184 #define         EPMM6           (0x0E | BANK1)
185 #define         EPMM7           (0x0F | BANK1)
186 #define         EPMCSL          (0x10 | BANK1)
187 #define         EPMCSH          (0x11 | BANK1)
188 #define         EPMOL           (0x14 | BANK1)
189 #define         EPMOH           (0x15 | BANK1)
190 #define         EWOLIE          (0x16 | BANK1)
191 #define         EWOLIR          (0x17 | BANK1)
192 #define         ERXFCON         (0x18 | BANK1)
193 #define         EPKTCNT         (0x19 | BANK1)
194
195
196
197 /* Bank 2 Registers */
198 #define         BANK2         (0x40)
199
200 #define         MACON1          (0x00 | BANK2 | 0x80)
201 #define         MACON2          (0x01 | BANK2 | 0x80)
202 #define         MACON3          (0x02 | BANK2 | 0x80)
203 #define         MACON4          (0x03 | BANK2 | 0x80)
204 #define         MABBIPG         (0x04 | BANK2 | 0x80)
205 #define         MAIPGL          (0x06 | BANK2 | 0x80)
206 #define         MAIPGH          (0x07 | BANK2 | 0x80)
207 #define         MACLCON1        (0x08 | BANK2 | 0x80)
208 #define         MACLCON2        (0x09 | BANK2 | 0x80)
209 #define         MAMXFLL         (0x0A | BANK2 | 0x80)
210 #define         MAMXFLH         (0x0B | BANK2 | 0x80)
211 #define         MAPHSUP         (0x0D | BANK2 | 0x80)
212 #define         MICON           (0x11 | BANK2 | 0x80)
213 #define         MICMD           (0x12 | BANK2 | 0x80)
214 #define         MIREGADR        (0x14 | BANK2 | 0x80)
215 #define         MIWRL           (0x16 | BANK2 | 0x80)
216 #define         MIWRH           (0x17 | BANK2 | 0x80)
217 #define         MIRDL           (0x18 | BANK2 | 0x80)
218 #define         MIRDH           (0x19 | BANK2 | 0x80)
219
220
221 /* Bank 3 Registers */
222 #define         BANK3         (0x60)
223
224 #define         MAADR1          (0x00 | BANK3 | 0x80)
225 #define         MAADR0          (0x01 | BANK3 | 0x80)
226 #define         MAADR3          (0x02 | BANK3 | 0x80)
227 #define         MAADR2          (0x03 | BANK3 | 0x80)
228 #define         MAADR5          (0x04 | BANK3 | 0x80)
229 #define         MAADR4          (0x05 | BANK3 | 0x80)
230 #define         EBSTSD          (0x06 | BANK3)
231 #define         EBSTCON         (0x07 | BANK3)
232 #define         EBSTCSL         (0x08 | BANK3)
233 #define         EBSTCSH         (0x09 | BANK3)
234 #define         MISTAT          (0x0A | BANK3 | 0x80)
235 #define         EREVID          (0x12 | BANK3)
236 #define         ECOCON          (0x15 | BANK3)
237 #define         EFLOCON         (0x17 | BANK3)
238 #define         EPAUSL          (0x18 | BANK3)
239 #define         EPAUSH          (0x19 | BANK3)
240
241
242 /*
243  * ENC28J60 PHY Registers
244  */
245
246 /*
247  * PHCON1: PHY Control Register 1
248  *
249  * +-------------------------------------------------------------+
250  * | PRST | PLOOPBK |  -  |  -  |  PPWRSV |  r  |  -  | PDPXMD | bit 8
251  * +-------------------------------------------------------------+
252  * |   r  |    -    |  -  |  -  |    -    |  -  |  -  |    -   | bit 0
253  * +-------------------------------------------------------------+
254  */
255 #define         PHCON1          0x00
256
257 #define         PRST          0x8000
258 #define         PLOOPBK         0x4000
259 #define         PPWRSV          0x0800
260 #define         PDPXMD          0x0100
261
262 /*
263  * PHSTAT1: Physical Layer Status Register 1
```

```
 *
 * +-------------------------------------------------------------+
 * |  -  |  -  |  -  | PFDPX | PHDPX |   --   |   --   |  -  | bit 8
 * +-------------------------------------------------------------+
 * |  -  |  -  |  -  |   -   |   -   | LLSTAT | JBSTAT |  -  | bit 0
 * +-------------------------------------------------------------+
 */
#define         PHSTAT1         0x01

#define         PFDPX         0x1000
#define         PHDPX         0x0800
#define         LLSTAT         0x0004
#define         JBSTAT         0x0002

/* PHHID1: PHY Identifier 1 */
#define         PHHID1         0x02

/* PHHID1: PHY Identifier 2 */
#define         PHHID2         0x03

/*
 * PHCON2: PHY Control Register 2
 *
 * +-------------------------------------------------------------+
 * |  -  | FRCLNK | TXDIS |  r  |  r  | JABBER |  r  | HDLDIS | bit 8
 * +-------------------------------------------------------------+
 * |  r  |    r   |   r   |  r  |  r  |    r   |  r  |    r   | bit 0
 * +-------------------------------------------------------------+
 */
#define         PHCON2         0x10

#define         FRCLNK         0x4000
#define         TXDIS         0x2000
#define         JABBER         0x0400
#define         HDLDIS         0x0100

/*
 * PHSTAT2: Physical Layer Status Register 2
 *
 * +-------------------------------------------------------------+
 * |  -  |  -  | TXSTAT | RXSTAT | COLSTAT | LSTAT | DPXSTAT |  -  | bit 8
 * +-------------------------------------------------------------+
 * |  -  |  -  | PLRITY |   --   |   ---   |   -   |    -    |  -  |bit 0
 * +-------------------------------------------------------------+
 */
#define         PHSTAT2         0x11

#define         TXSTAT         0x2000
#define         RXSTAT         0x1000
#define         COLSTAT         0x0800
#define         LSTAT         0x0400
#define         DPXSTAT         0x0200
#define         PLRITY         0x0020

/*
 * PHIE: PHY Interrupt Enable Register
 *
 * +-------------------------------------------------+
 * |  r  |  r  |  r  |    r   |  r  |  r  |   r   |  r  | bit 8
 * +-------------------------------------------------+
 * |  r  |  r  |  r  | PLNKIE |  r  |  r  | PGEIE |  r  | bit 0
 * +-------------------------------------------------+
 */
#define         PHIE         0x12

#define         PLNKIE         0x0010
#define         PGEIE         0x0002

/*
 * PHIR: PHY Interrupt Request (Flag) Regiter
 *
 * +-------------------------------------------------+
 * |  r  |  r  |  r  |    r   |  r  |   r   |  r  |  r  | bit 8
 * +-------------------------------------------------+
 * |  r  |  r  |  r  | PLNKIF |  r  | PGIF  |  r  |  r  | bit 0
 * +-------------------------------------------------+
 */
#define         PHIR         0x13

#define         PLNKIF         0x0010
```

```
344 #define        PGIF        0x0004
345
346 /*
347  * PHLCON: PHY Module Led Control Register
348  *
349  * +-------------------------------------------------------------------+
350  * |    r    |    r    |    r    |    r    | LACFG3 | LACFG2 | LACFG1 |
351  * +-------------------------------------------------------------------+
352  * | LBCFG3 | LBCFG2 | LBCFG1 | LBCFG0 | LFRQ1  | LFRQ0  | STRCH |
353  * +-------------------------------------------------------------------+
354  */
355 #define        PHLCON        0x14
356
357 #define        LACFG3        0x0800
358 #define        LACFG2        0x0400
359 #define        LACFG1        0x0200
360 #define        LACFG0        0x0100
361 #define        LBCFG3        0x0080
362 #define        LBCFG2        0x0040
363 #define        LBCFG1        0x0020
364 #define        LBCFG0        0x0010
365 #define        LFRQ1        0x0008
366 #define        LFRQ0        0x0004
367 #define        STRCH        0x0002
368
369 /*
370  * ERXFCON: Ethernet Receive Filter Control Register
371  *
372  * +-------------------------------------------------------+
373  * | UCEN | ANDOR | CRCEN | PMEN | MPEN | HTEN | MCEN | BCEN |
374  * +-------------------------------------------------------+
375  */
376 /*#define        ERXFCON        (0x18 | BANK1)    */
377 #define        UCEN        0x80
378 #define        ANDOR        0x40
379 #define        CRCEN        0x20
380 #define        PMEN        0x10
381 #define        MPEN        0x08
382 #define        HTEN        0x04
383 #define        MCEN        0x02
384 #define        BCEN        0x01
385
386 /*
387  * MACON1: MAC Control Register 1
388  *
389  * +-----+-------------------------------------------------------------+
390 LACFG0 | bit 8  |  -  |  LOOPBK  | TXPAUS | RXPAUS | PASSALL | MARXEN |
391  * +-----+-------------------------------------------------------------+
392  *r   | bit 0
393 /*#define        MACON1        (0x00 | BANK2 | 0x80)    */
394 #define        LOOPBK        0x10
395 #define        TXPAUS        0x08
396 #define        RXPAUS        0x04
397 #define        PASSALL        0x02
398 #define        MARXEN        0x01
399
400 /*
401  * MACON2: MAC Control Register 2
402  *
403  * +-------------------------------------------------------------------+
404  * | MARST | RNDRST |  -  |  -  | MARXRST | RFUNRST | MATXRST | TFUNRST |
405  * +-------------------------------------------------------------------+
406  */
407 /*#define        MACON2        (0x01 | BANK2 | 0x80)    */
408 #define        MARST        0x80
409 #define        RNDRST        0x40
410 #define        MARXRST        0x08
411 #define        RFUNRST        0x04
412 #define        MATXRST        0x02
413 #define        TFUNRST        0x01
414
415 /*
416  * MACON3: MAC Control Register 3
417  *
418  * +-------------------------------------------------------------------+
419  * | PADCFG2 | PADCFG1 | PADCFG0 | TXCRCEN | PHDREN | HFRMEN | FRMLNEN | FULDPX |
420  * +-------------------------------------------------------------------+
421  */
422 /*#define        MACON3        (0x02 | BANK2 | 0x80)    */
423 #define        PADCFG2        0x80
```

```
#define         PADCFG1         0x40
#define         PADCFG0         0x20
#define         TXCRCEN         0x10
#define         PHDREN          0x08
#define         HFRMEN          0x04
#define         FRMLNEN         0x02
#define         FULDPX          0x01

/*
 * MICMD: MII Command Register
 *
 * +---------------------------------------------------+
 * |  -  |  -  |  -  |  -  |  -  |  -  | MIISCAN | MIIRD |
 * +---------------------------------------------------+
 */
/*#define         MICMD           (0x12 | 0x80 | 0x80)     */
#define         MIISCAN         0x02
#define         MIIRD           0x01

/*
 * MISTAT: MII Status Register
 *
 * +---------------------------------------------------+
 * |  -  |  -  |  -  |  -  |  r  | NVALID | SCAN | BUSY |
 * +---------------------------------------------------+
 */
/*#define         MISTAT          (0x0A | 0x60 | 0x80)     */
#define         NVALID          0x04
#define         SCAN            0x02
#define         BUSY            0x01

/*
 * Packet Control Bytes
 *
 * +-----------------------------------------------------------------+
 * |  -  |  -  |  -  |  -  | PHUGEEN | PPADEN | PCRCEN | POVERRIDE |
 * +-----------------------------------------------------------------+
 */
#define         PHUGEEN         0x08
#define         PPADEN          0x04
#define         PCRCEN          0x02
#define         POVERRIDE       0x01


/* SPI Operation Codes */
#define         READ_CTRL_REG   0x00
#define         READ_BUF_MEM    0x3A
#define         WRITE_CTRL_REG  0x40
#define         WRITE_BUF_MEM   0x7A
#define         BIT_FIELD_SET   0x80
#define         BIT_FIELD_CLR   0xA0
#define         SOFT_RESET      0xFF

#define         RX_START_INIT   0x00
#define         RX_STOP_INIT    (0x1FFF - 0x0600 - 1)
#define         TX_START_INIT   (0x1FFF - 0x0600)
#define         TX_STOP_INIT    0x1FFF

#define         MAX_FRAME_LEN   1500


extern void         enc_start(void);
extern void         enc_stop(void);

extern void         enc_init(uint8_t *);
extern void         enc_write_op(uint8_t, uint8_t, uint8_t);
extern void         enc_write(uint8_t, uint8_t);
extern void         enc_set_bank(uint8_t);
extern void         enc_write_phy(uint8_t, uint16_t);
extern uint16_t         enc_read_phy(uint8_t);
extern uint8_t          enc_read(uint8_t);
extern uint8_t          enc_read_op(uint8_t, uint8_t);
extern void         enc_init_phy(void);
extern void             enc_set_mac(uint8_t *);

extern void         enc_read_buf(uint16_t, uint8_t *);
extern void         enc_write_buf(uint16_t, uint8_t *);
extern uint8_t          enc_get_rev(void);
extern void         enc_send_packet(uint16_t, uint8_t *);
```

```
504 extern uint16_t         enc_recv_packet(uint16_t, uint8_t *);
505
506
507
508 #endif /* __ENC28J60_H__ */
```

driver-Verzeichnis

enc28j60.c

```
1  /*-
2   * Copyright (c) 2013, mozgan.
3   * All Rights Reserved with BSD License.
4   * Read LICENSE file.
5   */
6
7  /**
8   * @file     enc28j60.c
9   * @brief    ENC28J60 ethernet driver
10  *
11  * @author    M. Ozgan, <mozgan@mozgan.org>
12  * @version    1.0
13  * @date    19.08.2013 18:51:18
14  * @internal
15  *    $Compiler:    gcc (on Mac, and 4.4BSD) $
16  *    $Company:    TU Wien $
17  *
18  *  @bug    'enc_send_packet' gives a warning with compiler option '-Wcast-align' because it reads bytes the 16-bit registers.
19  *  @todo    none
20  */
21
22 /*
23  *    @(#) driver/enc28j60.c        TU Wien 19.08.2013
24  *  $Id: enc28j60.c,v 1.0 19.08.2013 18:51:18 mozgan Exp $
25  */
26
27 #include <board/freq.h>
28 #include <device/enc28j60.h>
29 #include <driver/spi.h>
30
31 uint8_t enc_bank;
32 unsigned int p_next;
33
34 /**
35  * @brief start ENC28J60 pin to write or to read
36  */
37 void
38 enc_start(void)
39 {
40     ENC_CTRL_PORT &= ~(1 << ENC_CTRL_CS);
41 }
42
43 /**
44  * @brief stop the ENC28J60 pin
45  */
46 void
47 enc_stop(void)
48 {
49     ENC_CTRL_PORT |= (1 << ENC_CTRL_CS);
50 }
51
52 /**
53  * @brief set the mac-address to BANK3 register
54  *
55  * @param mymac[] my mac address
56  */
57 void
58 enc_set_mac(uint8_t mymac[])
59 {
60     enc_write(MAADR5, mymac[0]);
61     enc_write(MAADR4, mymac[1]);
62     enc_write(MAADR3, mymac[2]);
63     enc_write(MAADR2, mymac[3]);
64     enc_write(MAADR1, mymac[4]);
65     enc_write(MAADR0, mymac[5]);
66 }
67
68 /**
69  * @brief initialize the ENC28J60
```

```
 *
 * @param mymac[] my mac address
 */
void
enc_init(uint8_t mymac[])
{
    ENC_CTRL_DDR |= (1 << ENC_CTRL_CS);
    ENC_CTRL_PORT |= (1 << ENC_CTRL_CS);

    spi_init();

    enc_write_op(SOFT_RESET, 0, SOFT_RESET);
    _delay_ms(100);

    while (!(enc_read(ESTAT) & CLKRDY))
        ;


    /* do bank 0 stuff */
    p_next = RX_START_INIT;
    enc_write(ERXSTL, RX_START_INIT & 0xFF);
    enc_write(ERXSTH, RX_START_INIT >> 8);

    enc_write(ERXRDPTL, RX_START_INIT & 0xFF);
    enc_write(ERXRDPTH, RX_START_INIT >> 8);

    enc_write(ERXNDL, RX_STOP_INIT & 0xFF);
    enc_write(ERXNDH, RX_STOP_INIT >> 8);

    enc_write(ETXSTL, TX_START_INIT & 0xFF);
    enc_write(ETXSTH, TX_START_INIT >> 8);

    enc_write(ETXNDL, TX_STOP_INIT & 0xFF);
    enc_write(ETXNDH, TX_STOP_INIT >> 8);

    /* do bank 1 stuff */
    enc_write(ERXFCON, (UCEN | CRCEN | PMEN));
    enc_write(EPMM0, 0x3F);
    enc_write(EPMM1, 0x30);
    enc_write(EPMCSL, 0xF9);
    enc_write(EPMCSH, 0xF7);

    /* do bank 2 stuff */
    enc_write(MACON1, (MARXEN | TXPAUS | RXPAUS));
    enc_write(MACON2, 0x00);

    enc_write_op(BIT_FIELD_SET, MACON3, (PADCFG0 | TXCRCEN | FRMLNEN));

    enc_write(MAIPGL, 0x12);
    enc_write(MAIPGH, 0x0C);

    enc_write(MABBIPG, 0x12);

    enc_write(MAMXFLL, MAX_FRAME_LEN & 0xFF);
    enc_write(MAMXFLH, MAX_FRAME_LEN >> 8);


    /* do bank 3 stuff */
    enc_set_mac(mymac);

    enc_write_phy(PHCON2, HDLDIS);

    enc_set_bank(ECON1);
    enc_write_op(BIT_FIELD_SET, EIE, (INTIE | PKTIE));
    enc_write_op(BIT_FIELD_SET, ECON1, E_RXEN);
}

/**
 * @brief write the operation
 *
 * @param op operation to write on SPI
 * @param addr address to write
 * @param data data to send on SPI
 */
void
enc_write_op(uint8_t op, uint8_t addr, uint8_t data)
{
    enc_start();

    spi_write_op(op, addr);
```

```
150         spi_write(data);
151
152         enc_stop();
153     }
154
155     /**
156      * @brief write on ENC28J60
157      *
158      * @param addr address to write
159      * @param data data to send
160      */
161     void
162     enc_write(uint8_t addr, uint8_t data)
163     {
164         enc_set_bank(addr);
165         enc_write_op(WRITE_CTRL_REG, addr, data);
166     }
167
168     /**
169      * @brief set the bank on address
170      *
171      * @param addr address to set
172      */
173     void
174     enc_set_bank(uint8_t addr)
175     {
176         if ((addr & ADDR_MASK) != enc_bank) {
177             enc_write_op(BIT_FIELD_CLR, ECON1, (BSEL1 | BSEL0));
178             enc_write_op(BIT_FIELD_SET, ECON1, (addr & BANK_MASK) >> 5);
179             enc_bank = (addr & BANK_MASK);
180         }
181     }
182
183     /**
184      * @brief write the data on ENC28J60 physical address
185      *
186      * @param addr address
187      * @param data data
188      */
189     void
190     enc_write_phy(uint8_t addr, uint16_t data)
191     {
192         enc_write(MIREGADR, addr);
193
194         enc_write(MIWRL, data);
195         enc_write(MIWRH, data >> 8);
196
197         while (enc_read(MISTAT) & BUSY)
198             ;
199     }
200
201     /**
202      * @brief read from physical address of ENC28J60
203      *
204      * @param addr address
205      *
206      * @return data
207      */
208     uint16_t
209     enc_read_phy(uint8_t addr)
210     {
211         uint16_t data;
212
213         enc_write(MIREGADR, addr);
214         enc_write(MICMD, MIIRD);
215
216         while (enc_read(MISTAT) & BUSY)
217             ;
218
219         enc_write(MICMD, 0x00);
220
221         data = enc_read(MIRDL);
222         data |= enc_read(MIRDH);
223
224         return data;
225     }
226
227     /**
228      * @brief read from from address of ENC28J60
229      *
```

```
 * @param addr address
 *
 * @return operation
 */
uint8_t
enc_read(uint8_t addr)
{
    enc_set_bank(addr);

    return enc_read_op(READ_CTRL_REG, addr);
}

/**
 * @brief read the operation from address of ENC28J60
 *
 * @param op operation
 * @param addr address
 *
 * @return data from SPI
 */
uint8_t
enc_read_op(uint8_t op, uint8_t addr)
{
    uint8_t data;

    enc_start();

    spi_write_op(op, addr);
    spi_write(0x00);
    data = spi_read_addr(addr);

    enc_stop();

    return data;
}

/**
 * @brief initialize the physical interface of ENC28J60
 */
void
enc_init_phy(void)
{
    enc_write_phy(PHLCON, 0x0880);
    _delay_ms(500);

    enc_write_phy(PHLCON, 0x0990);
    _delay_ms(500);

    enc_write_phy(PHLCON, 0x0880);
    _delay_ms(500);

    enc_write_phy(PHLCON, 0x0990);
    _delay_ms(500);

    enc_write_phy(PHLCON, 0x0476);
    _delay_ms(500);
}

/**
 * @brief read from the buffer
 *
 * @param len length of data
 * @param data data to read
 */
void
enc_read_buf(uint16_t len, uint8_t *data)
{
    enc_start();

    spi_write(READ_BUF_MEM);

    while (len--)
        *data++ = spi_read();

    enc_stop();
}

/**
 * @brief write the buffer
 *
```

```
 * @param len length of data
 * @param data data
 */
void
enc_write_buf(uint16_t len, uint8_t *data)
{
    enc_start();

    spi_write(WRITE_BUF_MEM);

    while (len--)
        spi_write(*data++);

    enc_stop();
}

/**
 * @brief get the revision of ENC28J60
 *
 * @return revision
 */
uint8_t
enc_get_rev(void)
{
    return (enc_read(EREVID));
}

/**
 * @brief send a packet to ethernet
 *
 * @param len length of packet
 * @param packet packet to send
 */
void
enc_send_packet(uint16_t len, uint8_t *packet)
{
    enc_write(EWRPTL, TX_START_INIT);
    enc_write(EWRPTH, TX_START_INIT >> 8);

    enc_write(ETXNDL, (TX_START_INIT + len));

    enc_write(ETXNDH, (TX_START_INIT + len) >> 8);

    enc_write_op(WRITE_BUF_MEM, 0, 0x00);

    enc_write_buf(len, packet);

    enc_write_op(BIT_FIELD_SET, ECON1, TXRTS);
}

/**
 * @brief receive a packet from ethernet
 *
 * @param maxlen maximum length of packet to receive
 * @param packet packet to receive
 *
 * @return length of received packet
 */
uint16_t
enc_recv_packet(uint16_t maxlen, uint8_t *packet)
{
    unsigned int rxstat;
    unsigned int len;

    if (!enc_read(EPKTCNT))
        return 0;

    enc_write(ERDPTL, (p_next));
    enc_write(ERDPTH, (p_next) >> 8);

    p_next = enc_read_op(READ_BUF_MEM, 0);
    p_next |= (enc_read_op(READ_BUF_MEM, 0) << 8);

    len = enc_read_op(READ_BUF_MEM, 0);
    len |= (enc_read_op(READ_BUF_MEM, 0) << 8);

    rxstat = enc_read_op(READ_BUF_MEM, 0);
    rxstat |= (enc_read_op(READ_BUF_MEM, 0) << 8);

    len = MIN(len, maxlen);

```

```
390     enc_read_buf(len, packet);
391
392     enc_write(ERXRDPTL, (p_next));
393     enc_write(ERXRDPTH, (p_next) >> 8);
394
395     enc_write_op(BIT_FIELD_SET, ECON2, PKTDEC);
396
397     return len;
398 }
```

icp.h

```
1  /*-
2   * Copyright (c) 2013, mozgan.
3   * All Rights Reserved with BSD License.
4   * Read LICENSE file.
5   */
6
7  /**
8   * @file    icp.h
9   * @brief    header file for input capture pin operations
10  *
11  * @author    M. Ozgan, <mozgan@mozgan.org>
12  * @version    0.1
13  * @date    22.08.2013 15:56:37
14  * @internal
15  *    $Compiler:    gcc (on Mac, and 4.4BSD) $
16  *    $Company:    TU Wien $
17  *
18  *  @bug    none
19  *  @todo    none
20  */
21
22 /*
23  *    @(#) driver/icp.h        TU Wien 22.08.2013
24  *  $Id: icp.h,v 0.1 22.08.2013 15:56:37 mozgan Exp $
25  */
26
27 #ifndef __ICP_H__
28 #define __ICP_H__    1
29
30 #include <device/atmega16.h>
31
32
33
34 void            (*icp_handler)(uint16_t);        /**< call back function */
35 extern void        icp_init(void (*)(uint16_t));
36
37
38
39 #endif /* __ICP_H__ */
```

icp.c

```
1  /*-
2   * Copyright (c) 2013, mozgan.
3   * All Rights Reserved with BSD License.
4   * Read LICENSE file.
5   */
6
7  /**
8   * @file    icp.c
9   * @brief    driver for input capture pin operations
10  *
11  * @author    M. Ozgan, <mozgan@mozgan.org>
12  * @version    0.1
13  * @date    22.08.2013 15:57:28
14  * @internal
15  *    $Compiler:    gcc (on Mac, and 4.4BSD) $
16  *    $Company:    TU Wien $
17  *
18  *  @bug    none
19  *  @todo    none
20  */
21
22 /*
23  *    @(#) driver/icp.c        TU Wien 22.08.2013
24  *  $Id: icp.c,v 0.1 22.08.2013 15:57:28 mozgan Exp $
```

```
 */

#include <driver/leds.h>
#include <driver/icp.h>

/**
 * @brief initialize the input capture pin on timer 1
 *
 * @param call_back call back function from interrupt to call
 */
void
icp_init(void (*call_back)(uint16_t))
{
    uint32_t temp_ocr;

    if (call_back != NULL)
        icp_handler = call_back;

    TCNT1H = 0x00;
    TCNT1L = 0x00;

    TCCR1A = 0x00;

    /* CTC Mode - TOP: OCR1A */
    TCCR1B |= (1 << WGM12);
    /* Input Capture Noise Canceller on - Fallig Edge as trigger */
    TCCR1B |= (1 << ICNC1) | (1 << ICES1);

    /*
     * Freq: 1 Hz
     *
     *                F_CPU
     * OCR1A =  --------------------- - 1
     *            2 * PRES * Freq
     *
     */

    temp_ocr = (uint32_t)(F_CPU / 2);
    temp_ocr /= 256;
    temp_ocr--;
    OCR1A = temp_ocr;

    /* Pres.: 256 */
    TCCR1B |= (1 << CS12);

    TIMSK |= (1 << TICIE1);
/*    TIMSK |= (1 << OCIE1A); */  /* not needed the compare interrupt */
}

/*
ISR(TIMER1_COMPA_vect)
{
//    led_toggle(0);
}
*/

/**
 * @brief Timer1 Input Capture Interrupt
 *
 * @param TIMER1_CAPT_vect interrupt name
 */
ISR(TIMER1_CAPT_vect)
{
    if (icp_handler != NULL)
        icp_handler(ICR1);
}
```

leds.h

```
/*-
 * Copyright (c) 2013, mozgan.
 * All Rights Reserved with BSD License.
 * Read LICENSE file.
 */

/**
 * @file    leds.h
 * @brief    specific header file for leds driver
 *
```

```
 * @author    M. Ozgan, <mozgan@mozgan.org>
 * @version    1.0
 * @date    19.08.2013 15:28:53
 * @internal
 *    $Compiler:    gcc (on Mac, and 4.4BSD) $
 *    $Company:    TU Wien $
 *
 *  @bug    none
 *  @todo    none
 */

/*
 *    @(#) driver/leds.h        TU Wien 19.08.2013
 *  $Id: leds.h,v 1.0 19.08.2013 15:28:53 mozgan Exp $
 */

#ifndef __LEDS_H__
#define __LEDS_H__  1

#include <board/freq.h>
#include <include/common.h>

extern void        leds_init(void);
extern void        led_init(uint8_t);

extern void        leds_on(void);
extern void        led_on(uint8_t);

extern void        leds_off(void);
extern void        led_off(uint8_t);

extern void        leds_toggle(void);
extern void        led_toggle(uint8_t);



#endif /* __LEDS_H__ */
```

leds.c

```
/*-
 * Copyright (c) 2013, mozgan.
 * All Rights Reserved with BSD License.
 * Read LICENSE file.
 */

/**
 * @file    leds.c
 * @brief    driver for leds (to use for debugging)
 *
 * @author    M. Ozgan, <mozgan@mozgan.org>
 * @version    1.0
 * @date    19.08.2013 15:27:38
 * @internal
 *    $Compiler:    gcc (on Mac, GNU/Linux, 4.4BSD) $
 *    $Company:    TU Wien $
 *
 *  @bug    none
 *  @todo    none
 */

/*
 *    @(#) driver/leds.c        TU Wien 19.08.2013
 *  $Id: leds.c,v 1.0 19.08.2013 15:27:38 mozgan Exp $
 */

#include <driver/leds.h>

/**
 * @brief initialize all LEDs
 */
void
leds_init(void)
{
    LED_DDR    = LED_ALL;
    LED_PORT = LED_OFF;
}

```

```
39  /**
40   * @brief initialize a led
41   *
42   * @param led the number of led
43   */
44  void
45  led_init(uint8_t led)
46  {
47      LED_DDR |= (1 << led);
48      LED_PORT &= ~(1 << led);
49  }
50
51  /**
52   * @brief turn on all LEDs
53   */
54  void
55  leds_on(void)
56  {
57      LED_PORT = LED_ALL;
58  }
59
60  /**
61   * @brief turn on a led
62   *
63   * @param led the number of led
64   */
65  void
66  led_on(uint8_t led)
67  {
68      LED_PORT |= (1 << led);
69  }
70
71  /**
72   * @brief turn off all LEDs
73   */
74  void
75  leds_off(void)
76  {
77      LED_PORT = LED_OFF;
78  }
79
80  /**
81   * @brief turn off a led
82   *
83   * @param led the number of led
84   */
85  void
86  led_off(uint8_t led)
87  {
88      LED_PORT &= ~(1 << led);
89  }
90
91  /**
92   * @brief toggle all LEDs
93   */
94  void
95  leds_toggle(void)
96  {
97      LED_PORT ^= (LED_ALL);
98  }
99
100 /**
101  * @brief toggle a led
102  *
103  * @param led the number of led
104  */
105 void
106 led_toggle(uint8_t led)
107 {
108     LED_PORT ^= (1 << led);
109 }
```

net.h

```
1  /*-
2   * Copyright (c) 2013, mozgan.
3   * All Rights Reserved with BSD License.
4   * Read LICENSE file.
5   */
```

```
6
7   /**
8    * @file    net.h
9    * @brief    header file for network communication
10   *
11   * @author    M. Ozgan, <mozgan@mozgan.org>
12   * @version    0.2
13   * @date    20.08.2013 13:27:19
14   * @internal
15   *    $Compiler:    gcc (on Mac, GNU/Linux and 4.4BSD) $
16   *    $Company:    TU Wien $
17   *
18   *  @bug    none
19   *  @todo    none
20   */
21
22  /*
23   *    @(#) driver/net.h        TU Wien 20.08.2013
24   *  $Id: net.h,v 0.2 20.08.2013 13:27:19 mozgan Exp $
25   */
26
27  #ifndef __NET_H__
28  #define __NET_H__    1
29
30  /*
31   * Ethernet Frame Formats (v2)
32   * ---------------------------
33   *
34   * +----+----+------+------+-----+
35   * | DA | SA | Type | Data | FCS |
36   * +----+----+------+------+-----+
37   *
38   * Packet format:
39   * +-----------------------------------------------------+
40   * | Preamble | Dest. MAC | Src. MAC | Type | Data | FCS |
41   * +-----------------------------------------------------+
42   *
43   * Preamble                (8 bytes)
44   * DA      Destination MAC Address (6 bytes)
45   * SA      Source MAC Address      (6 bytes)
46   * Type    Protocol Type           (2 bytes)
47   * Data    Protocol Data           (46 - 1500 bytes)
48   * FCS     Frame Checksum          (4 bytes)
49   *
50   * Header:
51   * IP (the Internet protocol)    : 0x0800
52   * ARP                : 0x0806
53   *
54   * source:
55   * http://tools.ietf.org/id/draft-kaplan-isis-ext-eth-02.txt
56   * http://www.infocellar.com/networks/ethernet/frame.htm
57   * http://wiki.wireshark.org/Ethernet
58   */
59
60  /* ETH */
61  #define        ETH_HEADER_LEN    14
62
63  #define        ETHTYPE_ARP_H_V    0x08
64  #define        ETHTYPE_ARP_L_V    0x06
65  #define        ETHTYPE_IP_H_V    0x08
66  #define        ETHTYPE_IP_L_V    0x00
67
68  /* Ethernet type field */
69  #define        ETH_TYPE_H_P    12
70  #define        ETH_TYPE_L_P    13
71
72  #define        ETH_DEST_MAC    0
73  #define        ETH_SRC_MAC        6
74
75  /* ARP */
76  #define        ARP_OPCODE_REPLY_H_V    0x00
77  #define        ARP_OPCODE_REPLY_L_V    0x02
78
79  #define        ARP_L_V        0x06
80  #define        ARP_DEST_IP_P    0x26
81
82  #define        ARP_OPCODE_H_P    0x14
83  #define        ARP_OPCODE_L_P    0x15
84
85  #define        ARP_SRC_MAC_P    0x16
```

```
#define        ARP_SRC_IP_P    0x1C
#define        ARP_DEST_MAC_P    0x20
#define        ARP_DEST_IP_P    0x26

/*
 * Internet Header Format (v4)
 * ---------------------------
 *     0                   1                   2                   3
 *     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 *     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 *     |Version|  IHL  |Type of Service|          Total Length         |
 *     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 *     |         Identification        |Flags|      Fragment Offset    |
 *     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 *     |  Time to Live |    Protocol   |         Header Checksum       |
 *     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 *     |                       Source Address                          |
 *     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 *     |                    Destination Address                        |
 *     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 *     |                    Options                    |    Padding    |
 *     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 *             (Example Internet Datagram Header)
 *
 * IHL   -  Internet Header Length
 * TTL   -  Time To Live
 *
 * source:
 * http://tools.ietf.org/html/rfc791
 * http://tools.ietf.org/html/rfc760
 */
#define        IP_HEADER_LEN    20

#define        IP_SRC_P        0x1A
#define        IP_DEST_P        0x1E
#define        IP_CHECKSUM_P    0x18
#define        IP_TTL_P        0x16
#define        IP_FLAGS_P        0x14
#define        IP_P        0x0E
#define        IP_TOTLEN_H_P    0x10
#define        IP_TOTLEN_L_P    0x11

#define        IP_PROTO_P        0x17

#define        IP_PROTO_ICMP_V    1
#define        IP_PROTO_TCP_V    6
#define        IP_PROTO_UDP_V    17

/*
 * Internet Control Message Protocol (ICMP)
 * ----------------------------------------
 *
 * Message Formats:
 *    Version: 4
 *    IHL: Internet header length in 32-bit words.
 *    Type of Service: 0
 *    Total Length: Length of internet header and data in octets.
 *    Identification, Flags, Fragment Offset: see http://tools.ietf.org/html/rfc791
 *    Time to Live (TTL): Time to live in seconds.
 *    Protocol: ICMP = 1
 *    Header Checksum: The 16 bit one's complement of the one's complement
 *              sum of all 16 bit words in the header.
 *    Source Address: The address of the gateway or host that composes the
 *              ICMP message.
 *    Destination Address: The address of the gateway or host to which the
 *                  message should be sent.
 *
 * source:
 * http://tools.ietf.org/html/rfc792
 */
#define        ICMP_REPLY_V    0
#define        ICMP_REQUEST_V    8

#define        ICMP_TYPE_P        0x22
#define        ICMP_CHECKSUM_P    0x24

/*
 * User Datagram Protocol (UDP)
 * ----------------------------
 *
```

```
 * Format:
 *           0       7 8      15 16     23 24     31
 *          +--------+--------+--------+--------+
 *          |     Source       |   Destination    |
 *          |      Port        |      Port        |
 *          +--------+--------+--------+--------+
 *          |                  |                  |
 *          |     Length       |    Checksum      |
 *          +--------+--------+--------+--------+
 *          |
 *          |          data octets ...
 *          +---------------- ...
 *
 * Fields:
 *           0       7 8      15 16     23 24     31
 *          +--------+--------+--------+--------+
 *          |           source address           |
 *          +--------+--------+--------+--------+
 *          |        destination address         |
 *          +--------+--------+--------+--------+
 *          |  zero  |protocol|   UDP length     |
 *          +--------+--------+--------+--------+
 *
 *
 */
#define         UDP_HEADER_LEN     8

#define         UDP_SRC_PORT_H_P     0x22
#define         UDP_SRC_PORT_L_P     0x23
#define         UDP_DEST_PORT_H_P     0x24
#define         UDP_DEST_PORT_L_P     0x25

#define         UDP_LEN_H_P         0x26
#define         UDP_LEN_L_P         0x27
#define         UDP_CHECKSUM_H_P     0x28
#define         UDP_CHECKSUM_L_P     0x29
#define         UDP_DATA_P         0x2A
```

```
extern void         net_init(uint8_t *, uint8_t *);

extern uint8_t         eth_arp(uint8_t *, uint8_t);
extern uint8_t         eth_ip(uint8_t *, uint8_t);

extern void         arp_answer(uint8_t *, uint8_t);
extern void         echo_reply(uint8_t *, uint8_t);
extern void         udp_reply(uint8_t *, char *, uint8_t, uint16_t);

extern void         make_eth(uint8_t *);
extern void         make_ip(uint8_t *);

extern uint16_t         checksum(uint8_t *, uint16_t, uint8_t);

#endif /* __NET_H__ */
```

net.c

```
/*-
 * Copyright (c) 2013, mozgan.
 * All Rights Reserved with BSD License.
 * Read LICENSE file.
 */

/**
 * @file    net.c
 * @brief    driver for network communication
 *
 * @author    M. Ozgan, <mozgan@mozgan.org>
 * @version    0.2
 * @date    20.08.2013 13:29:36
 * @internal
 *    $Compiler:    gcc (on Mac, GNU/Linux, 4.4BSD) $
 *    $Company:    TU Wien $
 *
 *  @bug    none
 *  @todo    none
 */
```

```
/*
 *    @(#) driver/net.c        TU Wien 20.08.2013
 *  $Id: net.c,v 0.2 20.08.2013 13:29:36 mozgan Exp $
 */

#include <board/freq.h>
#include <device/enc28j60.h>
#include <driver/net.h>

#include <driver/leds.h>

static uint8_t macaddr[6];
static uint8_t ipaddr[4];

/**
 * @brief initialize the network
 *
 * @param mymac my mac address
 * @param myip my ip address
 */
void
net_init(uint8_t *mymac, uint8_t *myip)
{
    uint8_t i = 0;
    while (i < 4) {
        ipaddr[i] = myip[i];
        i++;
    }

    i = 0;
    while (i < 6) {
        macaddr[i] = mymac[i];
        i++;
    }
}

/**
 * @brief make the network wirh arp
 *
 * @param buf buffer
 * @param len length of buffer
 *
 * @return return 1 if success, otherwise zero
 */
uint8_t
eth_arp(uint8_t *buf, uint8_t len)
{
    uint8_t i = 0;

    if (len < 41)
        return 0;

    if ((buf[ETH_TYPE_H_P] != ETHTYPE_ARP_H_V) ||
        (buf[ETH_TYPE_L_P] != ETHTYPE_ARP_L_V))
        return 0;

    while (i < 4) {
        if (buf[ARP_DEST_IP_P + i] != ipaddr[i])
            return 0;

        i++;
    }

    return 1;
}

/**
 * @brief make the network communication with ip address
 *
 * @param buf buffer
 * @param len length of buffer
 *
 * @return returns 1 if success, otherwise zero
 */
uint8_t
eth_ip(uint8_t *buf, uint8_t len)
{
    uint8_t i = 0;

    if (len < 42)
```

```
102         return 0;
103
104     if ((buf[ETH_TYPE_H_P] != ETHTYPE_IP_H_V) ||
105         (buf[ETH_TYPE_L_P] != ETHTYPE_IP_L_V))
106         return 0;
107
108     while (i < 4) {
109         if (buf[IP_DEST_P + i] != ipaddr[i])
110             return 0;
111
112         i++;
113     }
114
115     return 1;
116 }
117
118 /**
119  * @brief make an answer if request
120  *
121  * @param buf buffer
122  * @param len length of buffer
123  */
124 void
125 arp_answer(uint8_t *buf, uint8_t len)
126 {
127     uint8_t i = 0;
128
129     make_eth(buf);
130
131     buf[ARP_OPCODE_H_P] = ARP_OPCODE_REPLY_H_V;
132     buf[ARP_OPCODE_L_P] = ARP_OPCODE_REPLY_L_V;
133
134     while (i < 6) {
135         buf[ARP_DEST_MAC_P + i] = buf[ARP_SRC_MAC_P + i];
136         buf[ARP_SRC_MAC_P + i] = macaddr[i];
137         i++;
138     }
139
140     i = 0;
141     while (i < 4) {
142         buf[ARP_DEST_IP_P + i] = buf[ARP_SRC_IP_P + i];
143         buf[ARP_SRC_IP_P + i] = ipaddr[i];
144         i++;
145     }
146
147     /* eth + arp = 42 bytes */
148     enc_send_packet(42, buf);
149 }
150
151 /**
152  * @brief make an echo answer (ping - pong) if request
153  *
154  * @param buf buffer
155  * @param len length of buffer
156  */
157 void
158 echo_reply(uint8_t *buf, uint8_t len)
159 {
160     make_eth(buf);
161     make_ip(buf);
162
163     buf[ICMP_TYPE_P] = ICMP_REPLY_V;
164
165     if (buf[ICMP_CHECKSUM_P] > (0xFF - 0x08))
166         buf[ICMP_CHECKSUM_P + 1]++;
167
168     buf[ICMP_CHECKSUM_P] += 0x08;
169
170     enc_send_packet(len, buf);
171 }
172
173 /**
174  * @brief make the UDP protocol
175  *
176  * @param buf buffer
177  * @param data data
178  * @param len length
179  * @param port port number
180  */
181 void
```

```
182 udp_reply(uint8_t *buf, char *data, uint8_t len, uint16_t port)
183 {
184     uint16_t ck;
185     uint8_t i = 0;
186
187     make_eth(buf);
188
189     if (len > 220)
190         len = 220;
191
192     buf[IP_TOTLEN_H_P] = 0;
193     buf[IP_TOTLEN_L_P] = IP_HEADER_LEN + UDP_HEADER_LEN + len;
194     make_ip(buf);
195
196     buf[UDP_DEST_PORT_H_P] = buf[UDP_SRC_PORT_H_P];
197     buf[UDP_DEST_PORT_L_P] = buf[UDP_SRC_PORT_L_P];
198     buf[UDP_SRC_PORT_H_P] = port >> 8;
199     buf[UDP_SRC_PORT_L_P] = port & 0xFF;
200
201     buf[UDP_LEN_H_P] = 0;
202     buf[UDP_LEN_L_P] = UDP_HEADER_LEN + len;
203
204     buf[UDP_CHECKSUM_H_P] = 0;
205     buf[UDP_CHECKSUM_L_P] = 0;
206
207     while (i < len) {
208         buf[UDP_DATA_P + i] = data[i];
209         i++;
210     }
211
212     ck = checksum(&buf[IP_SRC_P], 16 + len, 1);
213     buf[UDP_CHECKSUM_H_P] = ck >> 8;
214     buf[UDP_CHECKSUM_L_P] = ck & 0xFF;
215
216     enc_send_packet(UDP_HEADER_LEN + IP_HEADER_LEN + ETH_HEADER_LEN + len, buf);
217 }
218
219 /**
220  * @brief make the ethernet header
221  *
222  * @param buf buffer
223  */
224 void
225 make_eth(uint8_t *buf)
226 {
227     uint8_t i = 0;
228
229     while (i < 6) {
230         buf[ETH_DEST_MAC + i] = buf[ETH_SRC_MAC + i];
231         buf[ETH_SRC_MAC + i] = macaddr[i];
232         i++;
233     }
234 }
235
236 /**
237  * @brief make the ip header
238  *
239  * @param buf
240  */
241 void
242 make_ip(uint8_t *buf)
243 {
244     uint16_t ck;
245     uint8_t i = 0;
246
247     while (i < 4) {
248         buf[IP_DEST_P + i] = buf[IP_SRC_P + i];
249         buf[IP_SRC_P + i] = ipaddr[i];
250         i++;
251     }
252
253     /* clear checksum */
254     buf[IP_CHECKSUM_P] = 0;
255     buf[IP_CHECKSUM_P + 1] = 0;
256
257     buf[IP_FLAGS_P] = 0x40;         /* don't fragment */
258     buf[IP_FLAGS_P + 1] = 0;     /* fragment offset */
259     buf[IP_TTL_P] = 64;          /* TTL */
260
261     /* calculate the checksum */
```

```
262     ck = checksum(&buf[IP_P], IP_HEADER_LEN, 0);
263     buf[IP_CHECKSUM_P] = ck >> 8;
264     buf[IP_CHECKSUM_P + 1] = ck & 0xFF;
265 }
266
267 /**
268  * @brief checksum
269  *
270  * @param buf buffer
271  * @param len length
272  * @param type type
273  *
274  * @return sum
275  */
276 uint16_t
277 checksum(uint8_t *buf, uint16_t len, uint8_t type)
278 {
279     uint32_t sum = 0;
280
281     /* Type = UDP */
282     if (type == 1) {
283         sum += IP_PROTO_UDP_V;
284         sum += len - 8;
285     }
286
287     /* Type = TCP */
288     if (type == 2) {
289         sum += IP_PROTO_TCP_V;
290         sum += len - 8;
291     }
292
293     /* build the sum of 16 bit words */
294     while (len > 1) {
295         sum += 0xFFFF & (*buf << 8 | *(buf + 1));
296         buf += 2;
297         len -= 2;
298     }
299
300     if (len)
301         sum += (0xFF & *buf) << 8;
302
303     while (sum >> 16)
304         sum = (sum & 0xFFFF) + (sum >> 16);
305
306     return ((uint16_t) (sum ^ 0xFFFF));
307 }
```

spi.h

```
1  /*-
2   * Copyright (c) 2013, mozgan.
3   * All Rights Reserved with BSD License.
4   * Read LICENSE file.
5   */
6
7  /**
8   * @file    spi.h
9   * @brief    header file for spi interface
10  *
11  * @author    M. Ozgan, <mozgan@mozgan.org>
12  * @version    0.1
13  * @date    20.08.2013 14:52:02
14  * @internal
15  *    $Compiler:    gcc (on Mac, GNU/Linux and 4.4BSD) $
16  *    $Company:    TU Wien $
17  *
18  *  @bug    none
19  *  @todo    none
20  */
21
22 /*
23  *    @(#) driver/spi.h        TU Wien 20.08.2013
24  *  $Id: spi.h,v 0.1 20.08.2013 14:52:02 mozgan Exp $
25  */
26
27 #ifndef __SPI_H__
28 #define __SPI_H__    1
29
30 #include <board/freq.h>
```

```
#include <device/atmega16.h>
#include <device/enc28j60.h>


extern void         spi_init(void);
extern void         spi_write_op(uint8_t, uint8_t);
extern void         spi_write(uint8_t);
extern uint8_t         spi_read_addr(uint8_t);
extern uint8_t         spi_read(void);



#endif /* __SPI_H__ */
```

spi.c

```
/*-
 * Copyright (c) 2013, mozgan.
 * All Rights Reserved with BSD License.
 * Read LICENSE file.
 */

/**
 * @file    spi.c
 * @brief    driver for spi interface
 *
 * @author    M. Ozgan, <mozgan@mozgan.org>
 * @version    0.1
 * @date    20.08.2013 14:52:49
 * @internal
 *    $Compiler:    gcc (on Mac, and 4.4BSD) $
 *    $Company:    TU Wien $
 *
 *  @bug    none
 *  @todo    none
 */

/*
 *    @(#) driver/spi.c        TU Wien 20.08.2013
 *  $Id: spi.c,v 0.1 20.08.2013 14:52:49 mozgan Exp $
 */

#include <driver/spi.h>

/**
 * @brief initialize SPI communication
 */
void
spi_init(void)
{
    SPI_PORT |= (1 << SPI_SCK);

    SPI_DDR |= (1 << SPI_SCK);
    SPI_DDR &= ~(1 << SPI_MISO);
    SPI_DDR |= (1 << SPI_MOSI);
    SPI_DDR |= (1 << SPI_SS);

    SPI_CONTROL_REG |= (1 << SPI_MASTER);
    SPI_STATUS_REG |= (1 << SPI_DOUBLE_SPEED);
    SPI_CONTROL_REG |= (1 << SPI_ENABLE);
}

/**
 * @brief write an operation to address
 *
 * @param op operation
 * @param addr address
 */
void
spi_write_op(uint8_t op, uint8_t addr)
{
    SPI_DATA_REG = op | (addr & ADDR_MASK);

    while (!(SPI_STATUS_REG & (1 << SPI_INT_FLAG)))
        ;
}

/**
 * @brief write the data to SPI
```

```
64   *
65   * @param data data
66   */
67  void
68  spi_write(uint8_t data)
69  {
70      SPDR = data;
71
72      while (!(SPI_STATUS_REG & (1 << SPI_INT_FLAG)))
73          ;
74  }
75
76  /**
77   * @brief read from address
78   *
79   * @param addr address
80   *
81   * @return
82   */
83  uint8_t
84  spi_read_addr(uint8_t addr)
85  {
86      if (addr & 0x80) {
87          SPDR = 0x00;
88
89          while (!(SPSR & (1 << SPIF)))
90              ;
91      }
92
93      return SPDR;
94  }
95
96  /**
97   * @brief read from SPI data register
98   *
99   * @return value of SPI data register
100  */
101 uint8_t
102 spi_read(void)
103 {
104     SPDR = 0x00;
105
106     while (!(SPSR & (1 << SPIF)))
107         ;
108
109     return SPDR;
110 }
```

timer0.h

```
1   /*-
2    * Copyright (c) 2013, mozgan.
3    * All Rights Reserved with BSD License.
4    * Read LICENSE file.
5    */
6
7   /**
8    * @file    timer0.h
9    * @brief    specific header file for timer/counter 0
10   *
11   * @author    M. Ozgan, <mozgan@mozgan.org>
12   * @version    0.2
13   * @date    19.08.2013 15:54:58
14   * @internal
15   *    $Compiler:    gcc (on Mac, and 4.4BSD) $
16   *    $Company:    TU Wien $
17   *
18   *  @bug    none
19   *  @todo    none
20   */
21
22  /*
23   *    @(#) driver/timer0.h        TU Wien 19.08.2013
24   *  $Id: timer0.h,v 0.2 19.08.2013 15:54:58 mozgan Exp $
25   */
26
27  #ifndef __TIMER0_H__
28  #define __TIMER0_H__    1
29
```

```
30 #include <board/freq.h>
31 #include <include/common.h>
32
33 uint16_t     int_cnt0;     /**< timer 0 interrupt counter */
34
35
36
37 void         (*timer0_handler)(void);     /**< call back function from interrupt */
38
39 extern void     timer0_init_f(uint16_t, void (*)(void));
40 extern void     timer0_stop(void);
41
42
43
44 #endif /* __TIMER0_H__ */
```

timer0.c

```
1  /*-
2   * Copyright (c) 2013, mozgan.
3   * All Rights Reserved with BSD License.
4   * Read LICENSE file.
5   */
6
7  /**
8   * @file     timer0.c
9   * @brief    driver of timer/counter 0
10  *
11  * @author     M. Ozgan, <mozgan@mozgan.org>
12  * @version    0.2
13  * @date     19.08.2013 15:55:03
14  * @internal
15  *     $Compiler:     gcc (on Mac, and 4.4BSD) $
16  *     $Company:     TU Wien $
17  *
18  *  @bug     none
19  *  @todo     none
20  */
21
```

```
22 /*
23  *     @(#) driver/timer0.c         TU Wien 19.08.2013
24  *  $Id: timer0.c,v 1.0 19.08.2013 15:55:03 mozgan Exp $
25  */
26
27 #include <driver/timer0.h>
28 #include <driver/leds.h>
29
30 volatile int cnt0 = 0;
31
32 /**
33  * @brief initialize timer/counter 0 with frequency
34  *
35  * @param freq timer frequency
36  * @param call_back if timer interrupt request, returns to call_back function
37  */
38 void
39 timer0_init_f(uint16_t freq, void (*call_back)(void))
40 {
41     int temp_ocr0;
42
43     temp_ocr0 = (uint16_t)(((uint32_t)(F_CPU) / (uint32_t)(2*256*freq)));
44     temp_ocr0--;
45     int_cnt0 = 0;
46
47     timer0_handler = call_back;
48     TIMER0_CNT = 0x00;
49
50     /* if the OCR bigger as 255 then need the interrupt counter */
51     if (temp_ocr0 > 255) {
52         int_cnt0 = temp_ocr0 / 255;
53         TIMER0_OCR = 0xFF;
54     } else
55         TIMER0_OCR = temp_ocr0;
56
57     TIMER0_TCCR |= TIMER0_CTC;
58     TIMER0_TCCR |= TIMER0_PRES_256;
59
60     TIMER_INT_MASK |= (1 << OCIE0);
61     TIMER_INT_FLG |= (1 << OCF0);
```

```
62  }
63
64  /**
65   * @brief stop the timer 0
66   */
67  void
68  timer0_stop(void)
69  {
70      TIMER0_TCCR = TIMER0_NO_CLK;
71  }
72
73  /**
74   * @brief Interrupt for timer/counter 0 of campare output match mode
75   *
76   * @param TIMER0_COMP_vect Interrupt vector
77   */
78  ISR(TIMER0_COMP_vect)
79  {
80      if (int_cnt0 == 0) {
81          if (timer0_handler != NULL)
82              timer0_handler();
83      } else {
84          if (cnt0 % int_cnt0 == 0) {
85              if (timer0_handler != NULL)
86                  timer0_handler();
87          }
88          cnt0++;
89      }
90  }
```

include-Verzeichnis

common.h

```
1   /*-
2    * Copyright (c) 2013, mozgan.
3    * All Rights Reserved with BSD License.
4    * Read LICENSE file.
5    */
6
7   /**
8    * @file    common.h
9    * @brief    some useful definitions
10   *
11   * @author    M. Ozgan, <mozgan@mozgan.org>
12   * @version    0.1
13   * @date    19.08.2013 15:06:36
14   * @internal
15   *    $Compiler:    gcc (on Mac and 4.4BSD) $
16   *    $Company:    TU Wien $
17   *
18   *  @bug    none
19   *  @todo    none
20   */
21
22  /*
23   *    @(#) include/common.h        TU Wien 19.08.2013
24   *  $Id: common.h,v 0.1 19.08.2013 15:06:36 mozgan Exp $
25   */
26
27  #ifndef __COMMON_H__
28  #define __COMMON_H__    1
29
30  #include <stdbool.h>
31
32  /**
33   * @brief find minimum
34   *
35   * @param x first parameter
36   * @param y second parameter
37   *
38   * @return returns the minimum
39   */
40  #define        MIN(x, y)        (((x) < (y)) ? (x) : (y))
41
42  /**
43   * @brief find maximum
44   *
45   * @param x first parameter
```

```
 * @param y second parameter
 *
 * @return returns the maximum
 */
#define        MAX(x, y)        (((x) > (y)) ? (x) : (y))

#ifndef        TRUE        /**< if TRUE not defined, define it */
#define        TRUE        (bool)(1)
#endif

#ifndef        FALSE        /**< if FALSE not defined, define it */
#define        FALSE        (bool)(0)
#endif

#ifndef        true        /**< if true not defined, define it */
#define        true        (bool)(1)
#endif

#ifndef        false        /**< if false not defined, define it */
#define        false        (bool)(0)
#endif

#ifndef        NULL        /**< if NULL not defined, define it */
#define        NULL        (void *)(0)
#endif



#endif /* __COMMON_H__ */
```

stdbool.h

```
/*-
 * Copyright (c) 2013, mozgan.
 * All Rights Reserved with BSD License.
 * Read LICENSE file.
 */

/**
 * @file    stdbool.h
 * @brief    boolean definitions
 *
 * @author    M. Ozgan, <mozgan@mozgan.org>
 * @version    0.1
 * @date    19.08.2013 15:07:52
 * @internal
 *    $Compiler:    gcc (on Mac and 4.4BSD) $
 *    $Company:    TU Wien $
 *
 *  @bug    none
 *  @todo    none
 */

/*
 *    @(#) include/stdbool.h        TU Wien 19.08.2013
 *  $Id: stdbool.h,v 0.1 19.08.2013 15:07:52 mozgan Exp $
 */
#ifndef __STDBOOL_H__
#define __STDBOOL_H__    1

#include <stdio.h>
#include <stdint.h>

#ifndef        bool        /**< define the boolean data typ, if not defined */
#define        bool        _BOOL
#endif

#ifndef        _BOOL
#define        _BOOL        int
#endif



#endif /* __STDBOOL_H__ */
```

lib-Verzeichnis

buffer.h

```
/*-
 * Copyright (c) 2013, mozgan.
 * All Rights Reserved with BSD License.
 * Read LICENSE file.
 */

/**
 * @file    buffer.h
 * @brief    header file for ring buffer
 *
 * @author    M. Ozgan, <mozgan@mozgan.org>
 * @version    0.5
 * @date    20.08.2013 20:24:11
 * @internal
 *    $Compiler:    gcc (on Mac, and 4.4BSD) $
 *    $Company:    TU Wien $
 *
 *  @bug    none
 *  @todo    none
 */

/*
 *    @(#) lib/buffer.h        TU Wien 20.08.2013
 *  $Id: buffer.h,v 0.5 20.08.2013 20:24:11 mozgan Exp $
 */

#ifndef __BUFFER_H__
#define __BUFFER_H__    1

#include <stdio.h>
#include <include/common.h>

#define        LEN        10    /**< max. length of ring buffer */

volatile uint16_t buffer[LEN];    /**< ring buffer */
volatile uint8_t head;        /**< point the head of ring buffer */
volatile uint8_t tail;        /**< point the tail of ring buffer */


extern void        buffer_init(void);
extern void        buffer_write(uint16_t);
extern float        buffer_medium(void);


#endif /* __BUFFER_H__ */
```

buffer.c

```
/*-
 * Copyright (c) 2013, mozgan.
 * All Rights Reserved with BSD License.
 * Read LICENSE file.
 */

/**
 * @file    buffer.c
 * @brief    library for ring-buffer
 *
 * @author    M. Ozgan, <mozgan@mozgan.org>
 * @version    0.5
 * @date    20.08.2013 20:28:03
 * @internal
 *    $Compiler:    gcc (on Mac, and 4.4BSD) $
 *    $Company:    TU Wien $
 *
 *  @bug    none
 *  @todo    none
 */

/*
 *    @(#) lib/buffer.c        TU Wien 20.08.2013
 *  $Id: buffer.c,v 0.5 20.08.2013 20:28:03 mozgan Exp $
 */

#include <lib/buffer.h>
```

```
28
29  /**
30   * @brief initialize the ring buffer
31   */
32  void
33  buffer_init(void)
34  {
35      head = 0;
36      tail = 0;
37  }
38
39  /**
40   * @brief write the new data in ring buffer
41   *
42   * @param c new data to write in ring buffer
43   */
44  void
45  buffer_write(uint16_t c)
46  {
47      buffer[head] = c;
48
49      if (++head >= LEN)
50          head = 0;
51  }
52
53  /**
54   * @brief calculate the medium of ring buffer
55   *
56   * @return
57   */
58  float
59  buffer_medium(void)
60  {
61      uint8_t i;
62      float sum = 0;
63
64      for (i = 0; i < LEN; i++) {
65          sum += buffer[i];
66      }
67
68      sum /= LEN;
69
70      return sum;
71  }
```

Makefile(s)

BSDmakefile

```
1  # $FreeBSD$
2  #
3  # $Id: BSDmakefile, v1.0 19.08.2013 15:22 mozgan Exp $
4  #
5
6  .include "config.mk"
7  .include "Makefile.mk"
8  .include "Makefile"
```

GNUmakefile

```
1  #
2  # $Id: GNUmakefile,v1.0 19.08.2013 15:25  mozgan Exp $
3  #
4
5  include config.mk
6  include Makefile.mk
7  include Makefile
```

config.mk

```
1  #
2  # $Id: config.mk,v 1.0 19.08.2013 15:23 mozgan Exp $
3  #
4
5  ## ~ ~ ~ Header ~ ~ ~ ##
6  PROJECT         =    main
7  MCU         =    atmega16
8  F_CPU         =    16000000
```

Makefile.mk

```
#
# $Id: Makefile.mk,v 1.0 19.08.2013 15:23 mozgan Exp $
#

TARGET    =    $(PROJECT)

## Optimisation
OPT    =    s

## DEBUG
DEBUG    =    dwarf-2
DEBUG    +=    -DDEBUG

## Compiler C Standard
CSTD    =    -std=gnu99

## AVR-GCC Flags
DEPFLAGS=    -MD -MP -MF .dep/${@F}.d

CFLAGS    =    -g$(DEBUG)
CFLAGS    +=    -O$(OPT)
CFLAGS    +=    -funsigned-char -funsigned-bitfields -fpack-struct -fshort-enums
CFLAGS    +=    -Wall -Wstrict-prototypes
CFLAGS    +=    -Wa,-adhlns=$(<:.c=.lst)
CFLAGS    +=    $(patsubst %,-I%,$(EXTRAINCDIRS))
CFLAGS    +=    $(CSTD)
CFLAGS    +=    -DF_CPU=$(F_CPU)

ALL_CFLAGS    = -mmcu=$(MCU) -I$(PRJ_DIR) $(CFLAGS) $(DEPFLAGS)

## Libraries
MATH_LIB    =    -lm

## Linker flags
LDFLAGS    =    -Wl,-Map=$(TARGET).map,--cref
LDFLAGS    +=    $(MATH_LIB)

BOARD        =    Frequenzmessung

## ~ ~ ~ AVRDUDE ~ ~ ~ ##
AVRDUDE_PROGRAMMER    =    jtagmkII
AVRDUDE_PORT        =    usb

## ~ ~ ~ AVR-OBJCOPY ~ ~ ~ ##
FORMAT        =    ihex

## ~ ~ ~ Directories ~ ~ ~ ##
PRJ_DIR        =    .

BOARD_DIR    =    $(PRJ_DIR)/board
DEVICE_DIR    =    $(PRJ_DIR)/device
DRIVER_DIR    =    $(PRJ_DIR)/driver
INCLUDE_DIR    =    $(PRJ_DIR)/include
LIB_DIR        =    $(PRJ_DIR)/lib

## ~ ~ ~ Drivers ~ ~ ~ ##
LEDS        =    $(DRIVER_DIR)/leds.c
TIMER0        =    $(DRIVER_DIR)/timer0.c
ICP        =    $(DRIVER_DIR)/icp.c
SPI        =    $(DRIVER_DIR)/spi.c
ETHERNET    =    $(DRIVER_DIR)/enc28j60.c
NET        =    $(DRIVER_DIR)/net.c

## ~ ~ ~ Sources ~ ~ ~ ##
SRC        =    $(PROJECT).c

ERROR        =    #$(LIB_DIR)/error.c
BUFFER        =    $(LIB_DIR)/buffer.c

DRIVER_SRC    =    $(LEDS) $(TIMER0) $(ICP) $(SPI) $(ETHERNET) $(NET)
LIBRARY        =    $(ERROR) $(BUFFER)
EXTRAINCDIRS    =    $(INCLUDE_DIR) $(DEVICE_DIR) $(BOARD_DIR) $(DRIVER_DIR) $(LIB_DIR)
```

```
## ~ ~ ~ AVRDUDE ~ ~ ~ ##
AVRDUDE_FLAGS         =     -p $(MCU) -P $(AVRDUDE_PORT) -c $(AVRDUDE_PROGRAMMER)
AVRDUDE_WRITE_FLASH     =     -U flash:w:$(TARGET).hex
```

Makefile

```
#=============================================================================#
#
#      File: Makefile
#
#     Usage: make        (generate hex file to flash on chip
#            make flash    (flash the hex file on chip            )
#            make clean    (remove object, executables, prerequisits
#            make doxy    (generate doxygen documentation
#
#
#
#    Author: M. Ozgan, <mozgan@mozgan.org>
#      Created: 19.08.2013 15:22
#      Company: TU Wien
#
# $Id: Makefile,v 1.0 19.08.2013 15:22 mozgan Exp $
#
#=============================================================================#

## === Shell Commands ========================================================= ##
MAKE     = `which make`
SHELL     = `which sh`
CC     = `which avr-gcc`
OBJCOPY     = `which avr-objcopy`
OBJDUMP     = `which avr-objdump`
SIZE     = `which avr-size`
NM     = `which avr-nm`
AVRDUDE     = `which avrdude`
RM     = -rm -f
RM_DIR     = -rm -rf
COPY     = `which cp`
DOX     = `which doxygen`
DOXYFILE= doxy.file
## === Makefile Messages ====================================================== ##
MSG_LINE         = ============================
MSG_ERRORS_NONE         = Errors: none
MSG_BEGIN         = --- --- --- begin --- --- ---
MSG_END             = --- --- --- end --- --- ---
MSG_GCC_VERSION         = GCC Version
MSG_SIZE_BEFORE         = Size before:
MSG_SIZE_AFTER         = Size after:
MSG_COFF         = Converting to AVR COFF:
MSG_EXTENDED_COFF     = Converting to AVR Extended COFF:
MSG_FLASH         = Creating load file for Flash:
MSG_EEPROM         = Creating load file for EEPROM:
MSG_EXTENDED_LISTING     = Creating Extended Listing:
MSG_SYMBOL_TABLE     = Creating Symbol Table:
MSG_LINKING         = Linking:
MSG_COMPILING         = Compiling:
MSG_ASSEMBLING         = Assembling:
MSG_CLEANING         = Cleaning project:
MSG_CREATING_LIBRARY     = Creating library:

## === Sources ================================================================ ##
ALL_SRC         = $(DRIVER_SRC) $(LIBRARY) $(SRC)
OBJ         = $(ALL_SRC:.c=.o) #$(DRIVER_SRC:.c=.o) $(SRC:.c=.o) $(LIBRARY:.c=.o)
LST         = $(ALL_SRC:.c=.lst) #$(DRIVER_SRC:.c=.lst) $(LIBRARY:.c=.lst) $(SRC:.c=.lst
## ============================================================================ ##


LIBNAME         = lib$(TARGET).a

## === ELFSIZE - HEXSIZE ====================================================== ##
ELFSIZE     = $(SIZE) --mcu=$(MCU) -d -t -A $(TARGET).elf
ELFMEM     = $(SIZE) --mcu=$(MCU) -d -t -C $(TARGET).elf
HEXSIZE     = $(SIZE) --target=$(FORMAT) $(TARGET).hex

## === Targets ================================================================ ##
all: begin gccversion sizebefore build showtarget sizeafter finished end

lib: $(LIBNAME)
```

```
72
73  # === begin === #
74  begin    :
75      @echo
76      @echo $(MSG_BEGIN)
77
78  # === gccversion === #
79  gccversion:
80      @echo $(MSG_GCC_VERSION)
81      @echo $(MSG_LINE)
82      @$(CC) --version
83
84  # === sizebefore === #
85  sizebefore:
86      @if [ -f $(TARGET).elf ]; then                  \
87          echo;                                       \
88          echo $(MSG_SIZE_BEFORE);                    \
89          echo $(MSG_LINE);                           \
90          $(ELFSIZE);                                 \
91          $(ELFMEM);                                  \
92          echo;                                       \
93      fi
94
95  # === sizeafter === #
96  sizeafter:
97      @if [ -f $(TARGET).elf ]; then                  \
98          echo;                                       \
99          echo $(MSG_SIZE_AFTER);                     \
100         echo $(MSG_LINE);                           \
101         $(ELFSIZE);                                 \
102         $(ELFMEM);                                  \
103         echo;                                       \
104     fi
105
106 # === showtarget === #
107 showtarget:
108     @echo
109     @echo " --- --- --- Target  Information --- --- --- "
110     @echo "      AVR Model       : $(MCU)"
111     @echo "      Board           : $(BOARD)"
112     @echo "      MCU Frequency   : $(F_CPU) Hz"
113     @echo " --- --- --- --- --- --- --- --- --- --- --- "
114
115 # === finished === #
116 finished:
117     @echo $(MSG_ERRORS_NONE)
118
119 # === end === #
120 end:
121     @echo $(MSG_END)
122     @echo
123
124 # === build === #
125 build: elf hex eep lss sym
126
127 # === build targets === #
128 elf    : $(TARGET).elf
129 hex    : $(TARGET).hex
130 eep    : $(TARGET).eep
131 lss    : $(TARGET).lss
132 sym    : $(TARGET).sym
133
134 #.PRECIOUS: $(OBJ)
135 #.SECONDARY: $(TARGET).elf
136
137 $(TARGET).elf: $(OBJ)
138     @echo
139     @echo $(MSG_LINKING) $@
140     @echo $(MSG_LINE)
141     $(CC) $(ALL_CFLAGS) $(OBJ) $(LDFLAGS) --output $@
142
143 $(TARGET).hex: $(TARGET).elf
144     @echo
145     @echo $(MSG_FLASH) $@
146     @echo $(MSG_LINE)
```

```
147     $(OBJCOPY) -O $(FORMAT) -R .eeprom $(TARGET).elf $@
148
149 $(TARGET).eep: $(TARGET).elf
150     @echo
151     @echo $(MSG_EXTENDED_LISTING) $(TARGET).hex
152     @echo $(MSG_LINE)
153     $(OBJCOPY) -j .eeprom --set-section-flags=.eeprom="alloc,load
154         --change-section-lma .eeprom=0 -O $(FORMAT) $(TARGET).elf $@
155
156 $(TARGET).lss: $(TARGET).elf
157     @echo
158     @echo $(MSG_EXTENDED_LISTING) $(TARGET).lss
159     @echo $(MSG_LINE)
160     $(OBJDUMP) -h -S $(TARGET).elf > $(TARGET).lss
161
162 $(TARGET).sym: $(TARGET).elf
163     @echo
164     @echo $(MSG_SYMBOL_TABLE) $(TARGET).sym
165     @echo $(MSG_LINE)
166     $(NM) -n $(TARGET).elf > $(TARGET).sym
167
168 # === Object Files === #
169 .c.o    : $(ALL_SRC)
170     @echo
171     @echo $(MSG_COMPILING) $<
172     @echo $(MSG_LINE)
173     @if [ ! -e .dep ]; then                                    \
174         mkdir .dep 2>/dev/null $(wildcard .dep/*);             \
175     fi
176     $(CC) $(ALL_CFLAGS) -c $< -o $@
177
178 # === FLASH === #
179 flash: $(TARGET).hex $(TARGET).eep
180     $(AVRDUDE) $(AVRDUDE_FLAGS) $(AVRDUDE_WRITE_FLASH) $(AVRDUDE_WRITE_EEPROM)
181
182 # === clean === #
183 clean: begin clean_all finished end
184
185 clean_all:
186     @echo
187     @echo $(MSG_CLEANING)
188     $(RM) $(TARGET).hex
189     $(RM) $(TARGET).eep
190     $(RM) $(TARGET).obj
191     $(RM) $(TARGET).cof
192     $(RM) $(TARGET).elf
193     $(RM) $(TARGET).map
194     $(RM) $(TARGET).obj
195     $(RM) $(TARGET).sym
196     $(RM) $(TARGET).lnk
197     $(RM) $(TARGET).lss
198     $(RM) $(OBJ)
199     $(RM) $(LST)
200     $(RM) $(SRC:.c=.s)
201     $(RM) $(SRC:.c=.d)
202     $(RM_DIR) .dep
203     $(RM_DIR) doc
204
205 # === doxy === #
206 doxy:
207     @if [ ! -x $(DOX) ]; then                                  \
208         echo "you must install doxygen";                       \
209     else                                                       \
210         $(DOX) $(DOXYFILE);                                    \
211     fi
212
213 # === help === #
214 help:
215     @echo
216     @echo    "Usage: make [options]"
217     @echo    "       make       generate hex file to flash on chip"
218     @echo    "       flash      flash the hex file on chip"
219     @echo    "       clean      remove objects, executables, and prerequisits"
```

```
220     @echo    "       doxy       generate doxygen documentation"
221     @echo
222
223 # === .PHONY === #
224 .PHONY: all begin finish end sizebefore sizeafter gccversion build           \
225     elf hex eep lss sym coff extcoff clean clean_all flash
```

B.2 UNIX-Dämon

main-Programm

main.c

```c
/*-
 * Copyright (c) 2013, mozgan.
 * All Rights Reserved with BSD License.
 * Read LICENSE file.
 */

/**
 * @file    main.c
 * @brief    udp communication
 *
 * @author    M. Ozgan, <mozgan@mozgan.org>
 * @version    0.1
 * @date    23.08.2013 22:00:56
 * @internal
 *    $Compiler:    gcc (on Mac, and 4.4BSD) $
 *    $Company:    TU Wien $
 *
 *  @bug    none
 *  @todo    none
 */

/*
 *    @(#) src/main.c        TU Wien 23.08.2013
 *  $Id: main.c,v 0.1 23.08.2013 22:00:56 mozgan Exp $
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include "udp.h"
#include "error.h"

const char  *optstr = "h:p:";            /**< options */
char        *prg_name = "<not yet set>";    /**< program name */

/**
 * @brief usage of program how to call this
 */
void
usage(void)
{
    (void)fprintf(stdout, "Usage: \n");
    (void)fprintf(stdout, "\t%s -h <hostname> -p <port>\n", prg_name);
    (void)fprintf(stdout, "\tExample: %s -h FreeBSD.local -p 32000\n", prg_name);

    exit(EXIT_FAILURE);
}


/**
 * @brief main - start program
 *
 * @param argc count of arguments
 * @param argv vector of arguments
 *
 * @return int (if success returns EXIT_SUCCESS, otherwise returns EXIT_FAILURE)
 */
int
main(int argc, char *argv[])
{
    int         ch;
    int         opt_h, opt_p;
    char        *end_ptr;
    char        *hostname;    /**< hostname to bind */

    prg_name = argv[0];
    opt_h = opt_p = 0;

    /* it should be exactly 5 arguments */
    if (argc != 5)
        usage();

```

```
74      /* parse arguments */
75      while((ch = getopt(argc, argv, optstr)) != -1) {
76          switch(ch) {
77              case 'h':
78                  hostname = optarg;
79                  printf ("hostname: %s \n", hostname);
80                  opt_h = 1;
81                  break;
82              case 'p':
83                  port_nr = (unsigned int)strtol(optarg, &end_ptr, 10);
84                  printf ("port number: %d\n", port_nr);
85                  opt_p = 1;
86                  break;
87              case '?':
88                  usage();
89                  exit(EXIT_FAILURE);
90              default:
91                  abort();
92                  exit(EXIT_FAILURE);
93          }
94      }
95
96      /* check the options correct gived */
97      if (!opt_h || !opt_p)
98          usage();
99
100     /* no more arguments accept */
101     if (optind != argc)
102         usage();
103
104     make_socket(hostname, port_nr);        /* make the socket */
105     binding();                  /* bind to host */
106     communication();              /* send to host, recv from host */
107     close_socket();                 /* close the socket */
108
109     exit(EXIT_SUCCESS);
110 }
```


UDP-Protokoll

udp.h

```
1   /*-
2    * Copyright (c) 2013, mozgan.
3    * All Rights Reserved with BSD License.
4    * Read LICENSE file.
5    */
6
7   /**
8    * @file    udp.h
9    * @brief    header file for udp communication
10   *
11   * @author    M. Ozgan, <mozgan@mozgan.org>
12   * @version    0.1
13   * @date    23.08.2013 22:02:31
14   * @internal
15   *    $Compiler:    gcc (on Mac, and 4.4BSD) $
16   *    $Company:    TU Wien $
17   *
18   *  @bug    none
19   *  @todo    none
20   */
21
22  /*
23   *    @(#) src/udp.h        TU Wien 23.08.2013
24   *  $Id: udp.h,v 0.1 23.08.2013 22:02:31 mozgan Exp $
25   */
26
27  #ifndef __UDP_H__
28  #define __UDP_H__
29
30  #include <stdio.h>
31  #include <stdlib.h>
32  #include <string.h>
33  #include <unistd.h>
34
35  #include <netdb.h>
36  #include <sys/types.h>
```

```
37 #include <netinet/in.h>
38 #include <arpa/inet.h>
39 #include <sys/socket.h>
40
41 /*
42  * many compiler do not know the type 'u_long'
43  */
44 #ifndef    u_long
45 #define    u_long    unsigned long
46 #endif
47
48 /*
49  * in many systems has not defined 'h_addr'
50  */
51 #ifndef h_addr
52 #define h_addr h_addr_list[0]
53 #endif
54
55 struct hostent *h_name;            /**< real hostname */
56
57 struct sockaddr_in dest_addr;        /**< destination address */
58 struct sockaddr_in serv_addr;        /**< server address */
59
60 int        sock_fd;        /**< socket description */
61 unsigned int    port_nr;        /**< port number */
62
63 void        make_socket(const char *, unsigned int);  /**< make a
64 void        binding(void);        /**< bind to the socket */
65 void        communication(void);    /**< communication */
66 void        close_socket(void);    /**< close the open socket */
67
68 #endif /* __UDP_H__ */
```

udp.c

```
1  /*-
2   * Copyright (c) 2013, mozgan.
3   * All Rights Reserved with BSD License.
4   * Read LICENSE file.
5   */
6
7  /**
8   * @file    udp.c
9   * @brief    udp communication
10  *
11  * @author    M. Ozgan, <mozgan@mozgan.org>
12  * @version    0.1
13  * @date    23.08.2013 22:01:48
14  * @internal
15  *    $Compiler:    gcc (on Mac, and 4.4BSD) $
16  *    $Company:    TU Wien $
17  *
18  *  @bug    none
19  *  @todo    none
20  */
21
22 /*
23  *    @(#) src/udp.c        TU Wien 23.08.2013
24  *  $Id: udp.c,v 0.1 23.08.2013 22:01:48 mozgan Exp $
25  */
26
27 #include "udp.h"
28 #include "error.h"
29
30 /**
31  * @brief make the socket
32  *
33  * @param hostname get the hostname
34  * @param port port number to connection
35  */
36 void
37 make_socket(const char *hostname, unsigned int port)
38 {
39     memset(&dest_addr, 0, sizeof(dest_addr));
40
41     h_name = gethostbyname(hostname);
42
43     dest_addr.sin_family = AF_INET;
44     dest_addr.sin_addr.s_addr = *(u_long *)h_name->h_addr_list[0];
```

```
45      dest_addr.sin_port = htons(port);
46
47      serv_addr.sin_family = AF_INET;
48      serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
49      serv_addr.sin_port = htons(port);
50      memset(&serv_addr.sin_zero, 0, sizeof(serv_addr.sin_zero));
51
52      if ((sock_fd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
53          sys_err("creating socket failed");
54  }
55
56  /**
57   * @brief bind to host
58   */
59  void
60  binding(void)
61  {
62      if (bind(sock_fd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)))
63          sys_err("bind socket failed");
64  }
65
66  /**
67   * @brief make the communication with host (recv or send the datagram)
68   */
69  void
70  communication(void)
71  {
72      int         len;
73      char        buf[BUFSIZ];
74      socklen_t    sin_size;
75
76
77      /*** send to client ***/
78
79      if (strcpy(buf, "start") == NULL)
80          sys_err("copy the string to buffer failed");
81
82      (void)fprintf(stdout, "sending: %s\n", buf);
83
84      if (sendto(sock_fd, buf, strlen(buf), 0, (struct sockaddr *)&dest_addr, sizeof(struct sockaddr)) == -1)
85          sys_err("sendto failed");
86
87
88
89      /*** receive from client ***/
90
91      sin_size = (socklen_t)sizeof(struct sockaddr_in);
92      (void)fprintf(stdout, "waiting for packet ...\n");
93
94      if((len = recvfrom(sock_fd, buf, BUFSIZ, 0, (struct sockaddr *)&dest_addr, &sin_siz
95          sys_err("recvfrom failed");
96
97      (void)fprintf(stdout, "received packet from %s: ", inet_ntoa(dest_addr.sin_addr));
98
99      buf[len] = '\0';
100     (void)fprintf(stdout, "%s Hz\n", buf);
101
102 }
103
104 /**
105  * @brief close the socket if not to use in future
106  */
107 void
108 close_socket(void)
109 {
110     if (close(sock_fd) == -1)
111         sys_err("close socket failed");
112 }
```

Fehlerbehandlung

error.h

```
1   /*-
2    * Copyright (c) 2013, mozgan.
3    * All Rights Reserved with BSD License.
4    * Read LICENSE file.
5    */
```

```
/**
 * @file    error.h
 * @brief    header file for error
 *
 * @author    M. Ozgan, <mozgan@mozgan.org>
 * @version    0.1
 * @date    23.08.2013 22:20:44
 * @internal
 *    $Compiler:    gcc (on Mac, and 4.4BSD) $
 *    $Company:    TU Wien $
 *
 *  @bug    none
 *  @todo    none
 */

/*
 *    @(#) src/error.h        TU Wien 23.08.2013
 *  $Id: error.h,v 0.1 23.08.2013 22:20:44 mozgan Exp $
 */

#ifndef __ERROR_H__
#define __ERROR_H__

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <assert.h>

void        sys_err(const char *);    /* system error */



#endif /* __ERROR_H__ */
```

error.c

```
/*-
 * Copyright (c) 2013, mozgan.
 * All Rights Reserved with BSD License.
 * Read LICENSE file.
 */

/**
 * @file    error.c
 * @brief    error handling
 *
 * @author    M. Ozgan, <mozgan@mozgan.org>
 * @version    0.1
 * @date    23.08.2013 22:21:22
 * @internal
 *    $Compiler:    gcc (on Mac, and 4.4BSD) $
 *    $Company:    TU Wien $
 *
 *  @bug    none
 *  @todo    none
 */

/*
 *    @(#) src/error.c        TU Wien 23.08.2013
 *  $Id: error.c,v 0.1 23.08.2013 22:21:22 mozgan Exp $
 */

#include "error.h"

/**
 * @brief if an error occured then print the error message and exit from program
 *
 * @param msg programmer message to print
 */
void
sys_err(const char *msg)
{
    if (errno != 0)
        (void)fprintf(stderr, "Error: %s - %s\n", msg, strerror(errno));
    else
        (void)fprintf(stderr, "Error: %s\n", msg);

    exit(EXIT_FAILURE);
}
```