

IMPLEMENTIERUNG VON INTERNET-PROTOKOLLEN FÜR
REMOTE-CONTROL-APPLIKATIONEN AUF EINEM
MIKROCONTROLLER

DIPLOMARBEIT

VON

THOMAS FINKE

(MATRIKELNUMMER: 161032)

GEBOREN AM 01. NOVEMBER 1980 IN HEILBRONN-NECKARGARTACH

8. AUGUST 2007

BETREUER:

PROF. DR.-ING. JÜRGEN SCHRÖDER

HOCHSCHULE HEILBRONN

STUDIENGANG ELEKTRONIK UND INFORMATIONSTECHNIK

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt habe und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe und mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe. Die Arbeit hat in dieser oder ähnlicher Form noch keiner anderen Prüfungskommission vorgelegen.

Heilbronn, den 8. August 2007

(Thomas Finke)

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während meines Studiums und dem Anfertigen dieser Diplomarbeit unterstützt haben.

Zuallererst möchte ich Herrn Prof. Dr.-Ing. Jürgen Schröder für die Betreuung, Ermöglichung und Unterstützung dieser Diplomarbeit und vor allem für sein Engagement danken.

Für das Korrekturlesen und Einbringen vieler nützlicher Tipps bei der Erstellung dieser Arbeit danke ich: Fritz Göller, Erhard Dann, Detlef Nockert, Andreas Arz und meiner Freundin Bettina Göller.

An dieser Stelle möchte ich mich bei meiner Freundin auch noch für die vielen mit Geduld ertragenen Entbehrungen während meines Studiums bedanken.

In den letzten vier Jahren haben mich neben meiner Familie und Freunden auch meine Kommilitonen und Professoren begleitet. Daher möchte ich meinen Kommilitonen Andreas Wörner, Thomas Knorpp, Pascal Eisenhauer und Philipp Zipf für die schöne Studienzeit und die erfolgreichen Lerngruppen sowie meinen Professoren für die guten Vorlesungen danken.

Ein ganz besonderer Dank geht auch an Herrn Heinz Keller der *Keller Elektronik GmbH* [69] für die nette Unterstützung bei der Fertigung einer Platine für den ersten Versuchsaufbau.

Mein größter Dank gilt jedoch meiner Familie, welche mir stets ein sorgenfreies Studieren ermöglicht hat und zu jedem Zeitpunkt mit ihrem bedingungslosen Rückhalt hinter mir stand. Besonders hervorheben möchte ich hierbei die konstruktive Unterstützung durch meinen Vater Roland Finke.

Inhaltsverzeichnis

1. Einleitung	1
1.1. Einführung	1
1.2. Aufgabenstellung und Zielsetzung	2
2. Hardware des STK-LAN	5
2.1. Schnittstellen	7
2.2. Funktionalität	9
2.3. Realisierung	10
2.3.1. Ethernetcontroller	10
2.3.2. Spannungsversorgung	14
2.3.3. Pegelanpassung	16
2.3.4. Umschalten des SPI-Bus	17
2.3.5. Chipselect-Generierung	18
2.3.6. Übertrager	21
2.3.7. Programmierschnittstellen	22
2.3.8. Sonstiges	23
3. Software des Mikrocontrollerprogramms	25
3.1. Schnittstellen	27
3.2. Funktionalität	28
3.3. Realisierung	29
3.3.1. Netzwerktreiber	31
3.3.2. ARP	32
3.3.3. ICMP	34
3.3.4. Terminalserver	35
3.3.5. SNMP-Agent	41
3.3.6. TCP	46
3.3.7. Systemuhr	60
3.3.8. Buffer	62

Inhaltsverzeichnis

4. Test und Inbetriebnahme	63
4.1. Hardware	63
4.1.1. Verbinden des STK-LAN mit dem STK500 und Konfiguration der Boards	63
4.1.2. Einschalten der Boards und Überprüfen der LEDs	67
4.1.3. Abnehmen des STK-LAN vom STK500	67
4.2. Software	69
4.2.1. Programmieren des Mikrocontrollers	69
4.2.2. SNMP	87
4.2.3. HTTP	109
4.2.4. Terminalserver	111
5. Resümee	117
A. Netzwerkgrundlagen	119
A.1. Ethernet	123
A.1.1. Präambel	124
A.1.2. MAC-Adressen	125
A.1.3. Typ-Feld	125
A.1.4. Nutzdaten	126
A.1.5. PAD-Feld	126
A.1.6. CRC-Prüfsumme	126
A.1.7. VLAN-Tag	126
A.2. Address Resolution Protocol	127
A.2.1. Aufbau eines ARP-Datagramms	129
A.2.2. Proxy ARP	131
A.2.3. Gratuitous ARP	132
A.2.4. Reverse ARP (RARP)	132
A.2.5. Schwachstellen von ARP	133
A.3. Internet Protocol	133
A.3.1. Adressierung mit IPv4	134
A.3.2. Aufbau des IPv4-Headers	135
A.4. Internet Control Message Protocol	144
A.4.1. Mögliche ICMP-Typen	145
A.4.2. Angabe des ICMP-Codes	147
A.4.3. Berechnung der Prüfsumme	150
A.5. Transmission Control Protocol	150
A.5.1. Aufbau des TCP-Headers	153

Inhaltsverzeichnis

A.5.2. Verbindungsauf-/abbau	158
A.5.3. Datenübertragung	161
A.5.4. Flusssteuerung	162
A.6. User Datagram Protocol	167
A.6.1. Aufbau des UDP-Headers	168
A.6.2. UDP-Lite	170
A.6.3. Flusssteuerung	170
A.7. HyperText Transfer Protocol	171
A.7.1. Protokollversionen	171
A.7.2. Funktionsweise	174
A.7.3. Aufbau des HTTP-Headers	177
A.8. Simple Network Management Protocol	208
A.8.1. Aufbau eines SNMP-Systems	211
A.8.2. Managed Information Base	214
A.8.3. Aufbau der Beschreibungssprache ASN.1 und den «Basic Encoding Rules» (BER)	218
A.8.4. Beschreibung der «Managed Information Base» (MIB)	225
A.8.5. SNMP-Nachrichtentypen	232
A.8.6. Die SNMP-Versionen	236
A.8.7. Das Protokoll	238
A.8.8. Analyse einer SNMP-Nachricht	243
B. Marktübersicht	245
C. Entwicklungsumgebung	247
C.1. Hardwareentwicklungstools	247
C.1.1. CadSoft EAGLE	247
C.1.2. STK500	255
C.1.3. AVR-USB-JTAG	256
C.1.4. AVRATJTAGICE mkII	257
C.2. Softwareentwicklungstools	258
C.2.1. WinAVR	258
C.2.2. AVR Studio	264
D. Erzeugung der Gerberdaten	273
E. Modifikation des Makefile	279
F. Schaltplan und Layout	281

Inhaltsverzeichnis

G. Stückliste	289
H. Versionen des STK-LAN	293
I. Erweiterungen des SNMP-Agent	295
I.1. Einbindung weiterer <i>Managed Objects</i>	295
I.2. Realisierung eines MIB-Compilers	296
J. Netzwerkonfiguration des STK-LAN	297
K. Übersicht der implementierten Managed Objects	299
L. Die CD-ROM	301
Quellenverzeichnis	XIX
Abkürzungsverzeichnis	XXXI
Glossar	XXXVII

Abbildungsverzeichnis

2.1. Die im Rahmen dieser Diplomarbeit entwickelte Erweiterungsplatine STK-LAN . . .	6
2.2. Blockschaltbild des Gesamtsystems	6
2.3. Schnittstellen des STK-LAN	7
2.4. Realisierung des STK-LAN	11
2.5. Schnittstellen der Ethernetcontroller	11
2.6. Schnittstellen der Spannungsversorgung	14
2.7. Realisierung der Spannungsversorgung	15
2.8. Schnittstellen des Pegelwandlers	16
2.9. 5-Volt-tolerante Ports	16
2.10. Schnittstellen der SPI-Umschaltung	18
2.11. Schnittstellen der Chipselect-Generierung	19
2.12. Schnittstellen der Übertrager	21
2.13. Schnittstellen für die Programmieradapter	22
3.1. Blockschaltbild des Mikrocontrollerprogramms	26
3.2. Schnittstellen des Hauptprogramms	27
3.3. Schnittstellen des Treibers	31
3.4. Schnittstellen von ARP	32
3.5. Schnittstellen des Ethernet Protocols	33
3.6. Schnittstellen von ICMP	34
3.7. Schnittstellen des Terminalservers	35
3.8. Schnittstellen von UDP	38
3.9. Schnittstellen des RS232-Treibers	40
3.10. Schnittstellen des SNMP-Agent	41
3.11. Schnittstellen von TCP	47
3.12. Schnittstellen von IP	52
3.13. Schnittstellen des HTTP-Daemon	53
3.14. Schnittstellen der Systemuhr	61

Abbildungsverzeichnis

4.1. STK500 mit aufgestecktem STK-LAN	64
4.2. Benötigte Kabelverbindungen auf dem STK500	65
4.3. Kompletter Hardwareaufbau	66
4.4. Aushebelvorrichtung für das STK-LAN	68
4.5. Abnehmen des STK-LAN	68
4.6. Öffnen eines WinAVR-Projekts	69
4.7. Meldungen nach einem erfolgreichen Kompilervorgang	71
4.8. Programmoberfläche nach dem Start des AVR Studios	72
4.9. Dialog zum Öffnen eines Projekts oder einer Datei	73
4.10. Dialog zum Speichern eines Projekts	74
4.11. Auswahl des verwendeten Bausteins und der Debug-Plattform	75
4.12. Fehler in der Verbindung zum Programmieradapter	75
4.13. Es ist eine neuere Firmwareversion für JTAGICE mkII verfügbar	76
4.14. Fehlende Target-Spannung	76
4.15. Auswahl des Projektverzeichnisses	77
4.16. Schaltflächen zum Verbinden mit dem Mikrocontroller	77
4.17. Eigenschaften der Programmieradapter	78
4.18. ISP-Fehler	78
4.19. Einstellen der Fuses	79
4.20. Einstellung der LockBits	80
4.21. Erweiterte Einstellungen	81
4.22. Einstellungen des Programmierboards	82
4.23. Definieren einer Befehlsliste	83
4.24. Schaltflächen zum Steuern des Debuggens und Betrachten von Speicherinhalten	83
4.25. GET-Request mit SnmpTool	89
4.26. Programmoberfläche des MG-SOFT MIB Browsers	90
4.27. Einstellungen zum Verbinden mit dem SNMP-Agent	90
4.28. Aufrufen der verschiedenen SNMP-Requests über einen Rechtsklick	91
4.29. Aufrufen der verschiedenen SNMP-Requests über das Menü	92
4.30. Oberfläche der «SNMP Trap Ringer Console»	93
4.31. Die Oberfläche des iReasoning MIB Browser	94
4.32. Der erweiterte MIB-Tree nach dem Laden der MIB-Datei	95
4.33. Senden von Requests mit dem iReasoning MIB Browser	96
4.34. Oberfläche des Trap Receivers	97
4.35. Programmoberfläche von <i>CurrPorts</i>	99
4.36. Programmoberfläche von <i>Wireshark</i>	100
4.37. Auswahl des Standard-Netzwerkinterfaces	101

Abbildungsverzeichnis

4.38. Auswahl der Capture-Optionen	102
4.39. Aktiver Capture-Vorgang	103
4.40. Auswertung eines Datenpakets	105
4.41. Darstellung mit Microsoft Internet Explorer 7.0.5730.11	110
4.42. Oberfläche des Hercules SETUP utility	112
4.43. Registerkarte für die Kommunikation über RS232	113
4.44. Registerkarte für die Kommunikation über UDP	114
A.1. Kommunikation im OSI-Modell	120
A.2. Vergleich des OSI-Modells mit dem TCP/IP-Modell	123
A.3. Ethernet-II-Frameformat aus IEEE 802.3, inklusive VLAN Tag aus IEEE 802.1Q . .	124
A.4. Ablauf zum Senden eines IP-Pakets	128
A.5. Aufbau eines ARP-Datagramms	130
A.6. Aufbau des IPv4-Headers	136
A.7. Aufbau des TCP-Headers	153
A.8. Berechnung der TCP-Checksumme	156
A.9. Ablauf eines Drei-Wege-Handshakes	159
A.10. Vier-Wege-Verfahren zum Verbindungsabbau	161
A.11. Verwendung eines Sliding Window	163
A.12. Verwendung des Slow-Start-Verfahrens	165
A.13. Aufbau des UDP-Headers	168
A.14. Berechnung der UDP-Checksumme	169
A.15. Anforderung einer Ressource mit HTTP/1.0	172
A.16. Pipelining bei HTTP/1.0	173
A.17. Virtuelle Kanäle bei HTTP-NG	175
A.18. Form einer URL	175
A.19. Beispiel für einen TRACE-Request	188
A.20. Aufbau eines SNMP-Systems	210
A.21. Dr. Jeffrey D. Case, alias «Dr. SNMP»	210
A.22. Beispiel eines aufgesetzten SNMP-Agents	211
A.23. Verwendung eines Proxy-Agent	212
A.24. Kommunikation verschiedener SNMP-Communities	214
A.25. Struktur des MIB-I-Baums	215
A.26. Aufbau des Tag-Felds	219
A.27. Verschiedenen Formen der Längenangabe	220
A.28. Beispiel für den Datentyp OCTET STRING	221
A.29. Beispiel für den Datentyp OBJECT IDENTIFIER	222

Abbildungsverzeichnis

A.30. Beispiel für den Datentyp SEQUENCE	224
A.31. Öffnen einer Datei im MG-SOFT MIB-Compiler	229
A.32. Die Programmoberfläche des MG-SOFT MIB-Compilers nach erfolgreichem Kompilieren	230
A.33. Importieren des MIB-Moduls	231
A.34. Ergebnis der MIB-Beschreibungsdatei	231
A.35. Beispiel einer lexikografischen Sortierung	233
A.36. Kommunikation zwischen SNMP-Entities	238
A.37. Aufbau eines SNMPv1-Nachrichtenpakets	239
A.38. Nachrichtenpaket einer SNMPv1-Response	244
 B.1. Beck-IPC Modul im DIL32-Gehäuse	 245
 C.1. Startfenster der EAGLE-Installation	 248
C.2. Willkommensbildschirm beim Start der Installation	248
C.3. Zustimmung des Lizenzvertrags	249
C.4. Auswahl des Installationspfads	250
C.5. Beginn der eigentlichen Installation	250
C.6. Anzeige des Installationsfortschritts	251
C.7. Erfolgreiche Installation	251
C.8. Abschließen der Installation	252
C.9. Auswahl der gewünschten Lizenz	253
C.10. Oberfläche des Control Panels	253
C.11. Öffnen eines bestehenden Projekts	254
C.12. Das im Mikrocomputerlabor verwendete Entwicklungsboard STK500	256
C.13. Nachbau des Programmieradapters ATJTAGICE mit der Bezeichnung AVR-JTAG-USB	257
C.14. Alle Komponenten des AVRATJTAGICE mkII	258
C.15. Auswahl der gewünschten Sprache	259
C.16. Willkommens-Bildschirm beim Start der Installation	260
C.17. Annehmen der Lizenzvereinbarung	261
C.18. Auswahl des Zielverzeichnisses	262
C.19. Selektion der zu installierenden Komponenten	262
C.20. Kopieren der Installationsdateien	263
C.21. Beenden der Installationsroutine	263
C.22. Anzeige des Begrüßungsbildschirms nach dem Start der Installation	264
C.23. Anerkennung der Lizenz	265

Abbildungsverzeichnis

C.24. Auswahl des Installationspfades	266
C.25. Auswahl der zu installierenden Module	267
C.26. Bereit zum Kopieren der Dateien	267
C.27. Kopieren der Installationsdaten	268
C.28. Meldung über den erfolgreichen Installationsvorgang	268
C.29. Begrüßung zu Beginn der Installation	269
C.30. Auswahl der Installationsoptionen	270
C.31. Kopiervorgang der Daten	271
C.32. Installation abgeschlossen	271
D.1. Starten des CAM-Prozessors	274
D.2. Öffnen eines CAM-Jobs	275
F.1. Schaltplan des STK-LAN (Seite 1/4: Spannungsversorgung)	282
F.2. Schaltplan des STK-LAN (Seite 2/4: Ethernetport 1)	283
F.3. Schaltplan des STK-LAN (Seite 3/4: Ethernetport 2)	284
F.4. Schaltplan des STK-LAN (Seite 4/4: Steckverbinder und Logikbausteine)	285
F.5. Layout des STK-LAN (Top)	286
F.6. Layout des STK-LAN (Bottom)	287
L.1. Inhalt der beiliegenden CD-ROM zu dieser Diplomarbeit	302

Abbildungsverzeichnis

Tabellenverzeichnis

2.1. Steckerbelegung von 10Base-T	8
2.2. Pinbelegungen der ISP-Anschlüsse	9
2.3. Belegung von JTAG	10
2.4. Vergleich verschiedener Ethernetcontroller	13
2.5. Logiktablelle zur Generierung der Chipselect-Signale	20
3.1. Position des Terminalserver im TCP/IP-Referenzmodell	38
3.2. Vergleich der SNMP-Agent-Implementierungen	43
4.1. Auswertung des IP-Headers	106
4.2. Auswertung der SNMP-Nachricht	108
A.1. Position von Ethernet im TCP/IP-Referenzmodell	123
A.2. Position von ARP im TCP/IP-Referenzmodell	127
A.3. Position von IP im TCP/IP-Referenzmodell	133
A.4. Reservierungen einiger spezieller IPv4-Adressen	135
A.5. Beispielrechnung eines Subnetzes	135
A.6. Versionen des IP-Protokolls	137
A.7. Angaben im «Type Of Service»-Feld	138
A.8. Mögliche Prioritätsangaben bei IPv4	138
A.9. Ursprüngliches IP-Datagramm	140
A.10.Mögliche Fragmentierung des IP-Datagramms	140
A.11.Aufbau des «Option-Type Octet»	143
A.12.Werte für die «Option Class»	143
A.13.Mögliche Angaben für die «Option Number»	144
A.14.Position von ICMP im TCP/IP-Referenzmodell	145
A.15.Aufbau eines ICMP-Pakets	145
A.16.Auflistung der ICMP-Typnummern nach IANA [52]	146
A.17.Auflistung der ICMP-Codes nach IANA [52]	147
A.18.Position von TCP im TCP/IP-Referenzmodell	151

Tabellenverzeichnis

A.19.Optionen des TCP-Headers	157
A.20.Position von UDP im TCP/IP-Referenzmodell	167
A.21.Position von HTTP im TCP/IP-Referenzmodell	171
A.22.Position von SNMP im TCP/IP-Referenzmodell	208
A.23.Auflistung der Tag-Klassen	219
A.24.Einige ASN.1-Kennungen der SNMP-Datentypen	220
A.25.Einige ASN.1-Kennungen der SNMP-Datentypen	237
B.1. Übersicht über die Beck-IPC Module	246
G.1. Stückliste des STK-LAN	289
J.1. Für das STK-LAN reservierte IP-Adressen an der Hochschule Heilbronn	298

1. Einleitung

1.1. Einführung

Seit den 90er-Jahren hält das Internet rasant Einzug in immer mehr Anwendungsgebiete und die Anzahl der Menschen, welche das Internet zur alltäglichen Kommunikation nutzen, nimmt stetig zu. Bei den Wörtern Internet und Netzwerk denken die meisten Benutzer allerdings zuerst an PCs, die miteinander verbunden sind, um Daten auszutauschen. Neben den sehr bekannten Anwendungen, wie beispielsweise E-Mail, existieren aber auch Anwendungsgebiete, die noch nicht so sehr bekannt sind.

Eines dieser Anwendungsgebiete ist der «intelligente» Kühlschrank, welcher selbstständig Lebensmittel nachbestellen kann. Eine andere Anwendung ist der über das Internet steuerbare Herd. Durch Verwendung eines solchen Herds muss niemand mehr auf seiner Urlaubsfahrt umkehren, um den «vergessenen» Herd noch schnell auszuschalten. Stattdessen wird auf der Fahrt einfach beim nächsten Internetcafé angehalten und das Gerät über dessen Internetseite ausgeschaltet.

Neben diesen Anwendungen für den privaten Bereich existieren aber auch etliche industrielle Anwendungsgebiete. Ein Beispiel hierzu sind die kleinen Vermittlungsstellen der Telekommunikationsanbieter, welche an Straßenrändern angebracht sind. Diese können über eine Netzwerkverbindung gewartet und parametrierbar werden. Somit muss der Techniker bei einer Störung nicht erst zur Vermittlungsstelle fahren, sondern kann von seinem Büroarbeitsplatz aus einfach über das Netzwerk auf die Vermittlungsstelle zugreifen und die nötigen Parameter abrufen beziehungsweise verändern.

Weitere Anwendungsgebiete dieser Technik sind Alarmanlagen sowie Überwachungsfunktionen in der Produktion, welche im Alarm- beziehungsweise Fehlerfall automatisch die zuständige Person, beispielsweise per E-Mail, benachrichtigen können.

Für viele der oben genannten Anwendungen wäre der Einsatz eines Personal Computer (PC) jedoch total überdimensioniert, daher wird für kleinere Anwendungen oft *Ethernet* in *Embedded Systems* verwendet, welches die nötigen Dienste zur Verfügung stellt. Die dabei verwendeten Mikrocontroller

KAPITEL 1. EINLEITUNG

haben eine viel kleinere Bauform als ein Standard-PC und sind nebenbei noch viel stromsparender. Hinzu kommt, dass in vielen Systemen bereits ein Mikrocontroller vorhanden ist, welcher das Gerät steuert. Wenn dieser noch genug freie Ressourcen besitzt, kann die gewünschte Netzwerkanbindung einfach zusätzlich implementiert werden.

Da diese Form der Gerätesteuerung und -überwachung in immer mehr Produkten Anwendung findet und ein sehr breites Marktsegment abdeckt, sollen den Studierenden die Grundlagen von *Ethernet* in *Embedded Systems* im Rahmen des Mikrocomputerlabors der *Hochschule Heilbronn* näher gebracht werden.

1.2. Aufgabenstellung und Zielsetzung

Diese Diplomarbeit soll ein hierfür geeignetes System vorstellen und einen möglichst einfachen Einstieg in das Thema bieten. Aus diesem Grund wird in Anhang A dieser Arbeit sehr ausführlich auf die verwendeten Netzwerkprotokolle eingegangen. Weiterhin werden verschiedene Netzwerkprotokolle auf diesem System implementiert, um einen Einstieg in *Remote-Control-Applikationen*, also die Fernwartung von Geräten, zu ermöglichen.

Da sich die Studierenden im Rahmen des Mikrocomputerlabors mit *Atmel AVR Mikrocontrollern* auf dem *Atmel STK500 Starterkit* beschäftigen, soll die Ethernetanbindung auch auf diesem System basieren. Somit ist gewährleistet, dass die Studierenden mit einem bekannten System arbeiten und neue Erkenntnisse erlangen können.

Dies ist jedoch mit keinem der in Anhang B vorgestellten Systeme möglich, da diese immer eine Kombination aus Mikrocontroller und Ethernetanbindung darstellen und im Rahmen des Labors eine Etherneterweiterung für ein bestehendes Mikrocontrollerboard benötigt wird. Aus diesem Grund und dem erhöhten Lerneffekt wird daher für das *STK500* eine Aufsteckplatine entworfen, welche den Studierenden die Möglichkeit bietet, das *STK500* mit einem Netzwerk zu verbinden. Zusätzlich soll für dieses System ein Programm realisiert werden, welches einen *Webserver*, einen *Terminalserver* und einen *SNMP-Agent* zur Verfügung stellt. Der Code soll in C erstellt und möglichst übersichtlich gehalten werden, damit sich die Studierenden leicht einarbeiten können.

An diese Erweiterungsplatine werden noch einige weitere Anforderungen gestellt: Die Netzwerkerweiterung soll als Aufsteckplatine auf die Erweiterungsports des *STK500* aufgesteckt werden können und über diese Ports die nötigen Signal- und Versorgungsleitungen erhalten. Außerdem sollen auf der Erweiterung zwei Ethernetanschlüsse vorgesehen sein, so dass auch eine kleine *Router-* oder *Gateway-Applikation* realisiert werden kann. Für diese Ethernetanbindung müssen auf der Platine

KAPITEL 1. EINLEITUNG

die Ethernetcontroller sowie deren Spannungsversorgung integriert werden. Die Erweiterungsplatine muss zusätzlich einfach zu handhaben sein und die benötigten Programmieranschlüsse für den Mikrocontroller beinhalten.

Alle benötigten Sourcecodes, Programme und Unterlagen sind auf der beiliegenden CD-ROM (Anhang L) vorhanden.

KAPITEL 1. EINLEITUNG

2. Hardware des STK-LAN

Zur Realisierung einer Netzwerkanbindung wurde für das *STK500* eine Aufsteckplatine entworfen. Dieses Aufsteckmodul ist vergleichbar mit den für das *STK500* erhältlichen *Erweiterungsplatinen* von Atmel [23], wie beispielsweise dem *STK501*, dem *STK502*, dem *STK503*, dem *STK504* oder dem *Flamingo*, und trägt den Namen *STK-LAN*. Dieser Namen ist angelehnt an die *Atmel Starterkits* (*Starterkit (STK)*) und die Verwendung in einem *Local Area Network (LAN)*. Das *STK-LAN* (Abbildung: 2.1) kann zwar neben LANs auch in anderen Netzwerken betrieben werden, bei Lehrveranstaltungen im Rahmen des Studiums wird sich die Netzwerkanbindung jedoch größtenteils auf lokale Netze beschränken. Die Erweiterungsplatine kann, wie alle anderen *STK500-Erweiterungsplatinen*, direkt auf die beiden Erweiterungsports (*EXPAND0* und *EXPAND1*) des *STK500* aufgesteckt werden, was Abbildung 2.2 verdeutlicht.

KAPITEL 2. HARDWARE DES STK-LAN

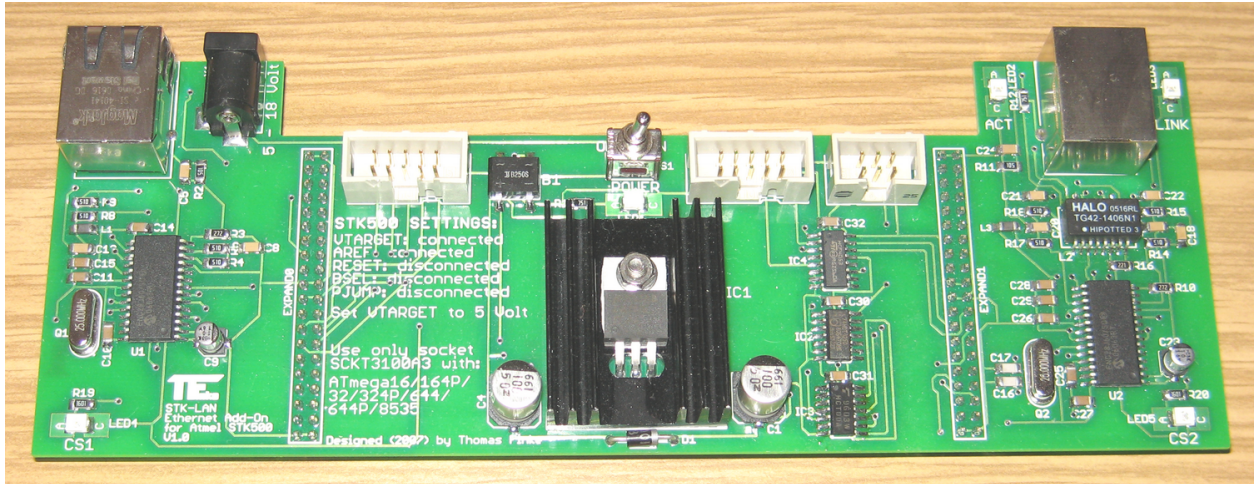


Abbildung 2.1.: Die im Rahmen dieser Diplomarbeit entwickelte Erweiterungsplatine STK-LAN

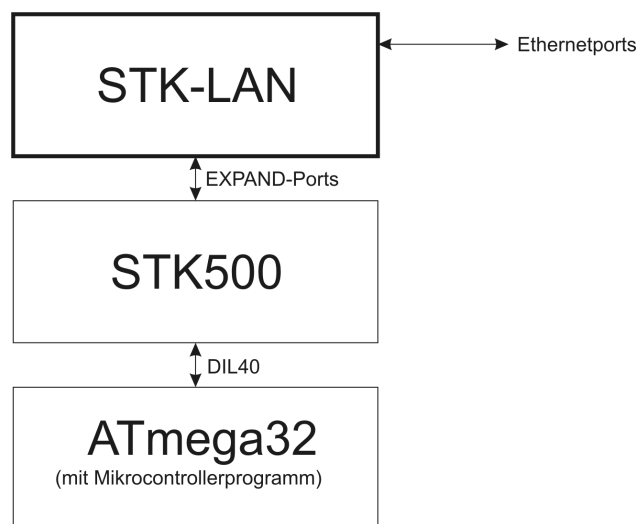


Abbildung 2.2.: Blockschaltbild des Gesamtsystems

2.1. Schnittstellen

Zur Kommunikation und Versorgung besitzt das *STK-LAN* einige Schnittstellen; diese sind:

- Spannungsversorgung

Die Erweiterungsplatine benötigt eine Versorgungsspannung im Bereich von 8 bis 12 Volt. Das *STK-LAN* benötigt mindestens 5 Volt zur Versorgung. Das *STK500* enthält einen 5 Volt Festspannungsregler und muss daher mit mindestens circa 8 Volt versorgt werden. Wird die untere Grenze der Versorgungsspannung des *STK-LAN* ebenfalls auf 8 Volt gelegt, können beide Platinen am selben Netzteil betrieben werden. Der auf dem *STK-LAN* verwendete Spannungsregler kann maximal mit 18 Volt versorgt werden. Durch die relativ hohe Stromaufnahme des *STK-LAN* muss in diesem Fall jedoch eine sehr hohe Verlustleistung vom Spannungsregler abgeführt werden. Aus diesem Grund wird die maximale Versorgungsspannung des *STK-LAN* auf 12 Volt begrenzt. Diese Spannung kann ebenfalls für die Versorgung des *STK500* verwendet werden. Durch einen integrierten Brückengleichrichter kann zur Versorgung der Platine eine Gleich- oder Wechselspannung verwendet werden. Die maximale Stromaufnahme des *STK-LAN* beläuft sich auf circa 400mA; zusammen mit dem *STK500* auf circa 900mA.

- Ethernetports

Zum Anschluss des *STK-LAN* an ein Netzwerk stehen zwei Ethernetports zur Verfügung. Diese sind durch *8P8C-Modularbuchsen*¹ nach außen geführt. Die Belegung dieser Modularbuchsen ist Tabelle 2.1 zu entnehmen.

¹*8P8C-Modularstecker* und *-buchsen* werden oft und fälschlicherweise auch als *RJ45* bezeichnet. Bei *RJ45* handelt es sich korrekterweise jedoch nur um eine Norm zur Verkabelung von Telekommunikationssystemen.

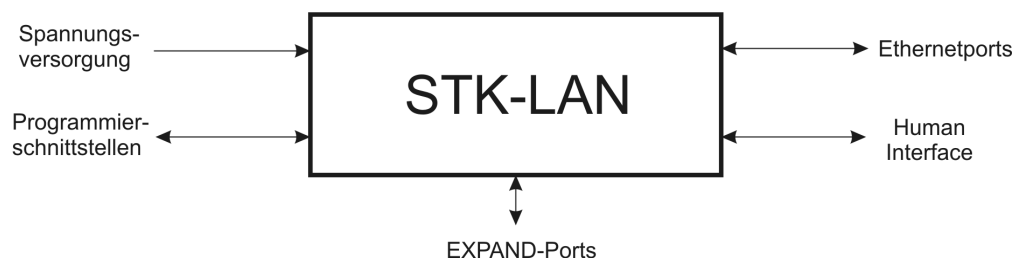


Abbildung 2.3.: Schnittstellen des STK-LAN

KAPITEL 2. HARDWARE DES STK-LAN

Pin	Signal	Beschreibung	Adernfarbe
1	TX+	Sendedaten	weiss/grün
2	TX-		grün
3	RX+	Empfangsdaten	weiss/orange
4			blau
5			weiss/blau
6	RX-	Empfangsdaten	orange
7			weiss/braun
8			braun

Tabelle 2.1.: Steckerbelegung von 10Base-T

- Programmierschnittstellen

Zur Programmierung des *AVR-Mikrocontrollers* auf dem *STK500* stehen durch das *STK-LAN* drei Programmierschnittstellen zur Verfügung. Diese sind durch Wannenstecker realisiert und können durch die Pinbelegungen nach den Tabellen 2.2(a), 2.2(b) und 2.3 direkt mit den gängigen Programmieradaptoren wie beispielsweise *JTAGICE mkII*, *JTAG ICE* oder *AVR-JTAG-USB* verbunden werden.

- EXPAND-Ports

Durch die beiden 40-poligen Stiftleisten *EXPAND0* und *EXPAND1* wird das *STK-LAN* elektrisch und mechanisch mit dem *STK500* verbunden. Die Belegung der beiden Stiftleisten kann Abbildung F.4 in Anhang F entnommen werden.

- Human Interface

Die Interaktion zwischen *STK-LAN* und Benutzer geschieht über einen Schalter und einige *LEDs*. Die Verwendung dieser Elemente ist nachfolgend zusammengefasst:

- Schalter S1 (POWER)

Mit diesem Schalter kann die Versorgungsspannung des *STK-LAN* ein- und ausgeschaltet werden.

KAPITEL 2. HARDWARE DES STK-LAN

Tabelle 2.2.: Pinbelegungen der ISP-Anschlüsse

(a) ISP6PIN		(b) ISP10PIN	
Pin	Signal	Pin	Signal
1	MISO	1	MOSI
2	VTG	2	VTG
3	SCK	3	
4	MOSI	4	GND
5	RST	5	RST
6	GND	6	GND
		7	SCK
		8	GND
		9	MISO
		10	GND

– LED1

Diese *Light Emitting Diode (LED)* zeigt durch Leuchten das Vorhandensein der Versorgungsspannung an.

– LEDs der Ethernetports

Beide Ethernetports besitzen *LEDs* zur Visualisierung der Ereignisse. *LED2* mit der Bezeichnung «ACT» zeigt eine Aktivität auf Ethernetport zwei an. *LED3* «LINK» leuchtet beim Vorhandensein einer physikalischen Netzwerkverbindung zu einem anderen Netzwerkteilnehmer an Ethernetport zwei. Die *LEDs* von Ethernetport eins haben die gleichen Bedeutungen.

– LEDs zur Anzeige der Chipselect-Signale

Die LEDs mit den Bezeichnern «CS1» und «CS2» zeigen an, welcher Ethernetport aktuell von der Software angesprochen wird.

2.2. Funktionalität

Das *STK-LAN* erweitert das *STK500* um zwei Ethernetports. Zur elektrischen und mechanischen Verbindung mit dem *STK500* sind die zwei Stiftleisten *EXPAND0* und *EXPAND1* vorgesehen.

Pin	Signal
1	TCK
2	GND
3	TDO
4	VTG
5	TMS
6	RST
7	VTG
8	
9	TDI
10	GND

Tabelle 2.3.: Belegung von JTAG

Zum Ansteuern der beiden Ethernetports enthält die Platine eine einfache Logik, zum Anpassen der Signalpegel einen Pegelwandler. Für die Spannungsversorgung des *STK-LAN* ist eine Spannungsregelung integriert. Das Ein-/Ausschalten der Platine geschieht über einen Schalter. Die Zustände der beiden Ethernetports sowie das Vorhandensein einer Versorgungsspannung werden über Leuchtdioden visualisiert. Die Programmierung des auf dem *STK500* vorhandenen *AVR-Mikrocontrollers* erfolgt über die Programmier- und Debugschnittstellen des *STK500*, welche auf dem *STK-LAN* über drei Wannenstecker herausgeführt sind. Der *SPI-Bus* besitzt hierbei eine automatische Umschaltung zwischen einem externen Programmieradapter und den internen Ethernetcontrollern.

2.3. Realisierung

Das detaillierte Blockschaltbild des *STK-LAN* ist in Abbildung 2.4 zu finden. Der komplette Schaltplan sowie das Platinenlayout befinden sich in Anhang F.

2.3.1. Ethernetcontroller

Zur Implementierung der untersten beiden Ebenen des *OSI-Referenzmodells* sind auf dem Markt fertige *Ethernetcontroller* verfügbar. Diese bieten einen *Physical Layer* (PHY) sowie einen *Medium*

KAPITEL 2. HARDWARE DES STK-LAN

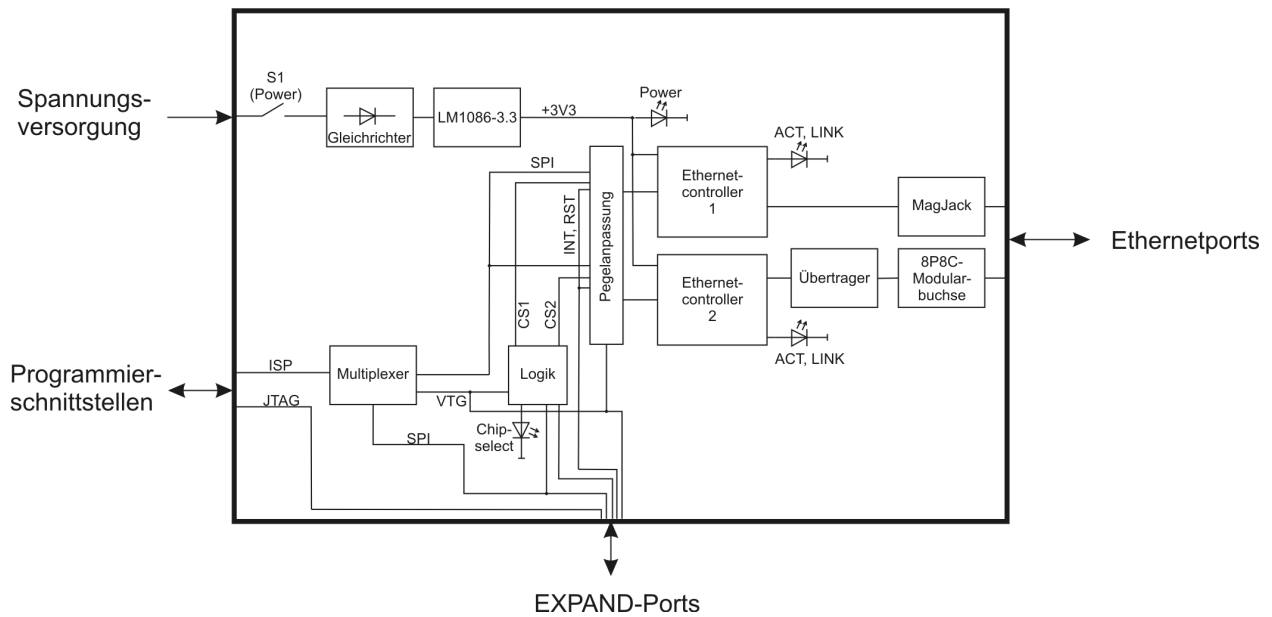


Abbildung 2.4.: Realisierung des STK-LAN

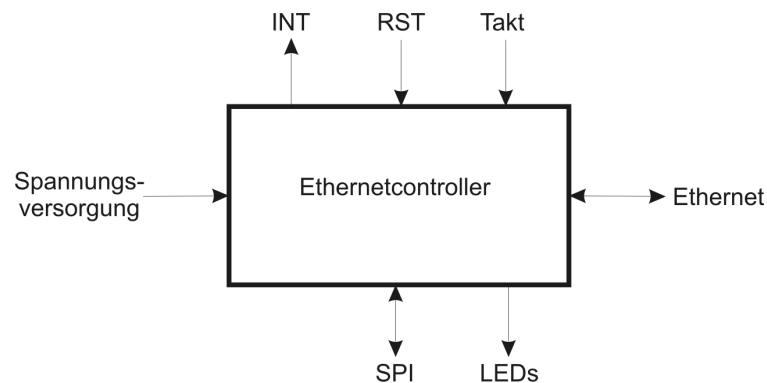


Abbildung 2.5.: Schnittstellen der Ethernetcontroller

Access Controller (MAC). Durch Verwendung eines solchen *Ethernetcontrollers* werden Schicht eins und zwei des *OSI-Modells* durch einen integrierten Schaltkreis realisiert. Der Anwender kann im Mikrocontrollerprogramm somit bei *OSI-Schicht zwei*² einsteigen.

2.3.1.1. Schnittstellen der Ethernetcontroller

Abbildung 2.5 zeigt die Schnittstellen der Ethernetcontroller. Diese sind:

²Gewisse Aufgaben, wie das Eintragen der *MAC-Adressen* in den *Ethernetframe*, müssen dennoch auf Schicht zwei erfolgen.

KAPITEL 2. HARDWARE DES STK-LAN

- Spannungsversorgung

Die Spannungsversorgung der Ethernetcontroller darf sich im Bereich von 3,1 bis 3,6 Volt bewegen und muss einen Strom von maximal 180mA je Controller liefern können.

- INT

Hierbei handelt es sich um eine low-aktive Interruptleitung, welche einstellbare Ereignisse melden kann.

- Reset (RST)

Mit der Reset-Leitung können die *Ethernetcontroller* zurückgesetzt werden. Diese Leitung ist low-aktiv, setzt die *Ethernetcontroller* also mit einem Low-Pegel zurück.

- Takt

Zur Taktversorgung besitzen die *Ethernetcontroller* einen Anschluss für einen externen Quarz.

- Serial Peripheral Interface (SPI)

Über den *SPI-Bus* können die Ethernetcontroller konfiguriert und die Nutzdaten zur Ethernetkommunikation übertragen werden.

- Ethernet

Über die Ethernet-Schnittstelle werden die *Ethernetcontroller* mit dem Netzwerk verbunden.

- LEDs

Die Leuchtdioden visualisieren eine physikalische Verbindung zum Netzwerk (LINK) sowie das Übertragen von Daten (ACT).

2.3.1.2. Funktionalität

Die beiden *Ethernetcontroller* besitzen einen *SPI-Bus* zur Kommunikation mit dem Mikrocontroller. Zusätzliche Reset- und Interruptleitungen ermöglichen das Zurücksetzen der *Ethernetcontroller* beziehungsweise das Melden von Ereignissen.

Die beiden Controller arbeiten auf den *OSI-Schichten* eins und zwei. Der *Physical Layer* realisiert die elektrische Anbindung des Systems an die Netzwerkschnittstelle. Der *Medium Access Controller* implementiert auf Schicht zwei des *OSI-Modells* das in Anhang A.1 vorgestellte *CSMA/CD-Verfahren* zur Zugriffssteuerung auf das Netzwerk.

KAPITEL 2. HARDWARE DES STK-LAN

Controller	Hersteller	Interface	Preis	Eigenschaften
RTL8019AS	Realtek Semiconductor Corp.	Paralleles Interface	~ 5,12 €	8/16-Bit Interface, am besten geeignet für Mikrocontroller mit nach außen geführtem Systembus
ENC28J60	Microchip Technology Inc.	SPI	~ 6,00 €	hohe Stromaufnahme von ca. 180 mA, hohe Verlustwärme, wenige Pins zum Anschluss nötig
CP2200	Silicon Laboratories	Paralleles Interface	~ 7,35 €	Übertrager mit 1:2,5 und 1:1 nötig
CS8900	Cirrus Logic, Inc.	Paralleles Interface	~ 7,93 €	kann im 8-Bit Modus Probleme bereiten
LAN91C96	SMSC	Paralleles Interface	~ 17,74 €	
LAN91C111	SMSC	Paralleles Interface	~ 24,37 €	unterstützt 10/100 MBit/s

Tabelle 2.4.: Vergleich verschiedener Ethernetcontroller

2.3.1.3. Realisierung

Auf dem Markt sind verschiedene Ethernetcontroller verfügbar, wie Tabelle 2.4 zeigt.

Für das *STK-LAN* wurde der Ethernetcontroller *ENC28J60* [90] von der Firma *Microchip Technology Inc.* [60] ausgewählt. Dieser hat zwar einen relativ hohen Stromverbrauch, was bei einem Laboraufbau jedoch nicht ins Gewicht fällt. Die beiden Vorteile dieses Ethernetcontrollers sind der relativ günstige Preis sowie das *SPI-Interface*. Letzteres bietet zum Einen die Möglichkeit den Ethernetcontroller auch mit einem Mikrocontroller mit nur wenigen freien Ausgangspins zu verbinden, was beispielsweise einer nachträglichen Erweiterung einer Schaltung um eine Ethernetschnittstelle entgegen kommt. Zum Anderen wird das Layouten einer Platine stark vereinfacht, da kein kompletter 8- oder 16-Bit Datenbus geroutet werden muss.

Der *ENC28J60* bietet die folgenden Features:

- 10 MBit/s Ethernet-Interface (10BASE-T)

KAPITEL 2. HARDWARE DES STK-LAN

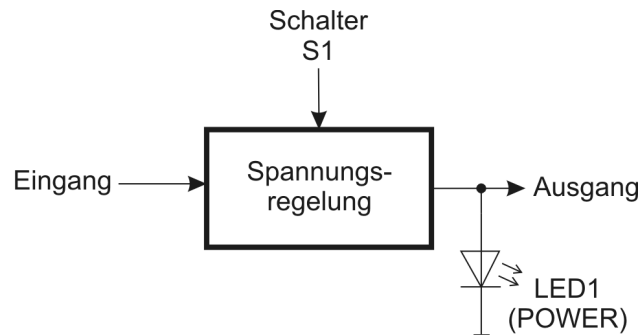


Abbildung 2.6.: Schnittstellen der Spannungsversorgung

- Voll-/Halbduplexen Datenverkehr
- 8 Kilobytes internen Buffer
- SPI-Takt bis 20 MHz

Die beiden *ENC28J60* besitzen zur Generierung ihres Systemtakts je einen 25 Megahertz (MHz) Quarz. Weiterhin sind beide Controller an dem Pin *VCAP* mit einem externen Glättungskondensator für die interne 2,5-Volt-Versorgung ausgestattet. Der *RBIAS* Pin ist über je einen 2,7 Kiloohm Widerstand mit *Ground (GND)* verbunden.

Der *AVR-Mikrocontroller* besitzt für jeden der beiden *ENC28J60* eine eigene Reset- sowie Interrupt-Leitung, wie der Schaltplan (Abbildung F.2 und F.3) veranschaulicht.

2.3.2. Spannungsversorgung

2.3.2.1. Schnittstellen der Spannungsversorgung

Die Erweiterungsplatine verfügt über eine 2,1mm Hohlbuchse zum Anschluss an ein Netzteil. Die Versorgungsspannung für das *STK-LAN* darf im Bereich von 8 bis 12 Volt³ liegen. Die Stromaufnahme beläuft sich auf maximal circa 400mA.

Auf die Polung der angeschlossenen Versorgungsspannung muss nicht geachtet werden.

Zum Ein- und Ausschalten der Platine ist Schalter S1 «POWER» (Abbildung: 2.6) integriert. Am Ausgang der Spannungsregelung stehen 3,3 Volt bei maximal 1,5 Ampere zur Verfügung. Die 3,3 Volt werden durch LED1 angezeigt.

³An dieser Stelle sind die Informationen aus Kapitel 2.1 zu beachten.

KAPITEL 2. HARDWARE DES STK-LAN

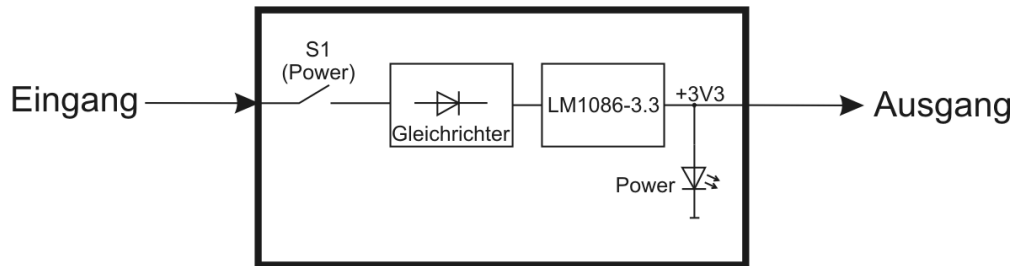


Abbildung 2.7.: Realisierung der Spannungsversorgung

2.3.2.2. Funktionalität

Da die verwendeten Ethernetcontroller vom Typ *ENC28J60* eine Versorgungsspannung von 3,3 Volt (3,1 - 3,6 Volt) benötigen und das *STK500* intern mit 5 Volt betrieben wird, muss eine eigene Spannungsversorgung für das *STK-LAN* realisiert werden. Hierzu wird ein Spannungsregler verwendet, welcher einen maximalen Ausgangsstrom von 1,5 A liefern kann und Eingangsspannungen im Bereich von 5 bis 18 Volt unterstützt.

Die beiden Ethernetcontroller können maximal je 180 mA Strom aufnehmen. Zusammen mit den Logik-Bausteinen und LEDs beläuft sich die maximale Stromaufnahme des *STK-LAN* auf circa 400 mA. Die Spannungsversorgung des *STK500* kann maximal circa 500 mA liefern, wovon sich das *STK500* selbst noch versorgt. Daher kann der Spannungsregler des *STK-LAN* nicht von der internen 5 Volt Versorgungsspannung des *STK500* gespeist werden. Es wird also für das *STK-LAN* ein komplett eigener Spannungsversorgungsanschluss benötigt.

2.3.2.3. Realisierung

Zur Generierung der 3,3-Volt-Versorgungsspannung wird ein Brückengleichrichter, gefolgt von einem Spannungsregler des Typs *LM1086IT-3.3* [93], verwendet. Abbildung 2.7 veranschaulicht diesen Aufbau. Der vollständige Schaltplan der Spannungsversorgung des *STK-LAN* ist in Abbildung F.1 in Anhang F zu finden.

Beim Aufbau der Platine sollte darauf geachtet werden, dass zur Montage des Spannungsreglers auf dem Kühlkörper etwas Wärmeleitpaste verwendet wird, um einen optimalen Wärmeübergang zu gewährleisten.

KAPITEL 2. HARDWARE DES STK-LAN

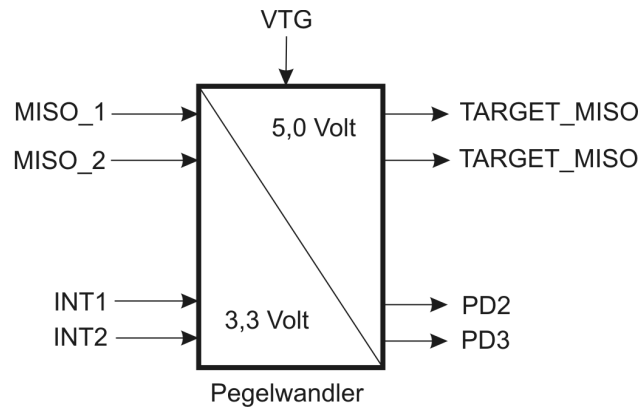


Abbildung 2.8.: Schnittstellen des Pegelwandlers

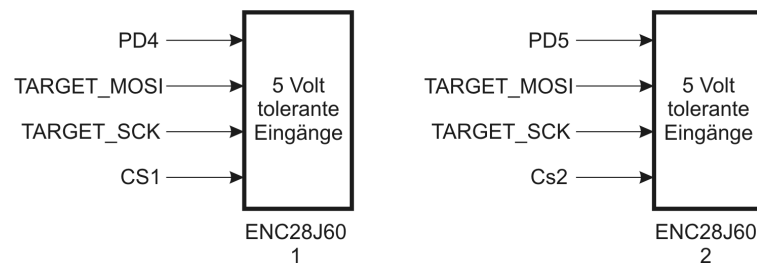


Abbildung 2.9.: 5-Volt-tolerante Ports

2.3.3. Pegelanpassung

Da das *STK500* mit 5 Volt betrieben wird und das *STK-LAN* hingegen 3,3 Volt benötigt, müssen einige Pegel der Signalleitungen zwischen den beiden Boards angepasst werden.

2.3.3.1. Schnittstellen der Pegelanpassung

Die Pegelanpassung des *STK-LAN* wird in zwei Teile unterteilt:

- Anpassung mit Pegelwandler

Die Pegelanpassung (Abbildung 2.8) besitzt vier Eingänge für die 3,3-Volt-Signale und vier Ausgänge für die entsprechenden 5-Volt-Signale. Als Versorgungsspannung erhält der Pegelwandler die *Target-Spannung* (VTG) des *STK500*.

- Anpassung mit 5-Volt-toleranten Eingängen

Die Signale vom *AVR-Mikrocontroller* zu den *Ethernetcontrollern* werden wie in Abbildung 2.9 direkt auf die Eingangspins der *ENC28J60* gegeben.

KAPITEL 2. HARDWARE DES STK-LAN

2.3.3.2. Funktionalität

Der Pegelwandler erhält an seinen Eingängen die 3,3-Volt-Pegel der *Ethernetcontroller*. An seinen Ausgängen liefert er anschließend die gleichen Signale wie an den Eingängen, allerdings mit einem Spannungspegel von 5,0 Volt.

2.3.3.3. Realisierung

Da die Eingänge der *Ethernetcontroller* 5-Volt-kompatibel sind, können diese Signale vom Mikrocontroller direkt zu den *Ethernetcontrollern* verbunden werden. Die Signale, welche von den *Ethernetcontrollern* zum Mikrocontroller übertragen werden, müssen hingegen auf 5-Volt-Pegel gebracht werden. Dies ist nötig, da bei den *Atmel AVR-Mikrocontrollern* nicht garantiert werden kann, dass sie 3,3 Volt als High-Pegel erkennen.

Die Pegelumsetzung wird auf einfachste Weise mit einem *74HCT125* [94] realisiert. Dabei handelt es sich um einen Vierfach-Treiberbaustein mit tristate Ausgängen. Es kann hierfür auch ein anderer Baustein verwendet werden. Es sollte allerdings darauf geachtet werden, dass der verwendete Baustein der *HCT-Familie* angehört, da nur diese einen logischen High-Pegel ab 2,0 Volt⁴ erkennt. Der am Ausgang vorhandene 5-Volt-Pegel kann anschließend direkt auf die Eingänge des *AVR-Mikrocontrollers* gegeben werden.

2.3.4. Umschalten des SPI-Bus

Bei den beiden verwendeten *Ethernetcontrollern* handelt es sich um Exemplare mit einer *SPI-Schnittstelle* zur Kommunikation mit dem Mikrocontroller. Da die Mikrocontroller der *AVR-Familie* typischerweise auch über *In-System-Programming (ISP)* programmiert werden können, muss die *SPI-Schnittstelle* des Mikrocontrollers zwischen den *Ethernetcontrollern* und dem *ISP-Programmieradapter* umgesteckt/umgeschaltet werden können. Da ein Umstecken zwischen den Teilnehmern auf Dauer sehr umständlich ist, wird auf dem *STK-LAN* eine automatische Umschaltung verwendet.

2.3.4.1. Schnittstellen der Umschaltung

Abbildung 2.10 zeigt die Schnittstellen der SPI-Umschaltung. Der Multiplexer erhält als Versorgungsspannung die *Target-Spannung (VTG)*. Die Anschlüsse PB5, PB6 und PB7 dienen zum Anschluss des *SPI-Bus* des Mikrocontrollers. Die Anschlüsse *ISP_SCK*, *ISP_MOSI* und *ISP_MISO*

⁴Diese Angabe gilt für eine Versorgungsspannung von 5,0 Volt.

KAPITEL 2. HARDWARE DES STK-LAN

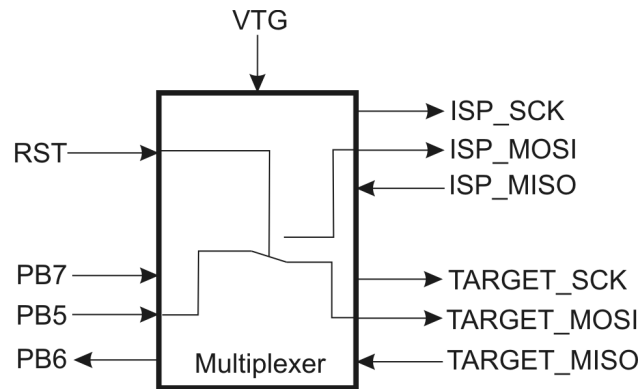


Abbildung 2.10.: Schnittstellen der SPI-Umschaltung

führen zum Programmieradapter; die Anschlüsse TARGET_SCK, TARGET_MOSI und TARGET_MISO zu den *Ethernetcontrollern*. Mit der Reset-Leitung (RST) wird der Multiplexer geschaltet.

2.3.4.2. Funktionalität

Zur Umschaltung des *SPI-Bus* zwischen den beiden *Ethernetcontrollern* und dem Programmieradapter wird ein Multiplexer verwendet. Dieser wird über die Reset-Leitung des Mikrocontrollers gesteuert.

Solange die Reset-Leitung vom Programmieradapter nicht auf *GND* gezogen wird, ist der *AVR-Mikrocontroller* mit den *Ethernetcontrollern* verbunden. Sobald jedoch der Programmiervorgang beginnt und der Programmieradapter die Reset-Leitung betätigt, schaltet der Multiplexer die *SPI-Schnittstelle* von den *Ethernetcontrollern* ab und verbindet diese mit dem Programmieradapter.

2.3.4.3. Realisierung

Zur Realisierung der Umschaltung wird ein Analogmultiplexer vom Typ 4053 [131] verwendet, welcher laut Schaltplan (Abbildung F.4) angeschlossen ist.

2.3.5. Chipselect-Generierung

Da beide *Ethernetcontroller* am selben *SPI-Bus* angeschlossen sind, muss der jeweils aktive durch ein *Chipselect-Signal* ausgewählt werden. Damit der Treiber für die *Ethernetcontroller* nicht modifiziert werden muss, ist auf dem *STK-LAN* eine einfache Logik implementiert. Durch diese Logik wird

KAPITEL 2. HARDWARE DES STK-LAN

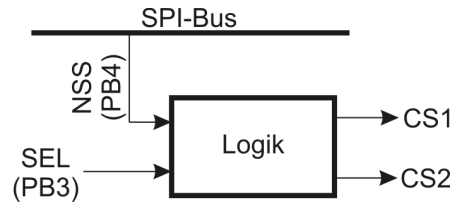


Abbildung 2.11.: Schnittstellen der Chipselect-Generierung

das *Chipselect-Signal* des Treibers mit einem Signal zur Auswahl des *Ethernetcontrollers* verknüpft. Somit kann an jeder Stelle im Mikrocontrollerprogramm durch das Schalten eines bestimmten Ausgangspins der zu verwendende *Ethernetcontroller* ausgewählt werden.

2.3.5.1. Schnittstellen der Chipselect-Generierung

Die Logik zur Generierung der beiden *Chipselect-Signale* hat, wie Abbildung 2.11 zeigt, zwei Eingänge für die Signale vom Mikrocontroller. Dies sind das *Chipselect-Signal* vom Treiber (*NSS*) und das Signal zur Auswahl des gewünschten *Ethernetcontrollers* (*SEL*). Am Ausgang liefert die Logik zwei getrennte Signale zum Ansteuern der *Ethernetcontroller*.

2.3.5.2. Funktionalität

Die Logik muss aus den beiden Eingangssignalen die zugehörigen Ausgangssignale generieren. Hierbei ist zu beachten, dass die Signale «NSS», «CS1» und «CS2» low-aktiv sind. Die *Ethernetcontroller* werden also mit einer Null ausgewählt.

Zur Auswahl des gewünschten Controllers steht das Signal «SEL» (0 = Ethernetcontroller 1, 1 = Ethernetcontroller 2) zur Verfügung.

Das Zusammenfassen dieser Anforderungen in einer Logiktafel resultiert in dem Ergebnis nach Tabelle 2.5.

2.3.5.3. Realisierung

Zur Bestimmung der benötigten Schaltung wird Tabelle 2.5 als Gleichung ausgedrückt:

$$CS1 = (\overline{NSS} \wedge \overline{SEL}) \vee (NSS \wedge \overline{SEL}) \vee (NSS \wedge SEL)$$

Aus den ersten beiden Termen wird \overline{SEL} ausgeklammert:

KAPITEL 2. HARDWARE DES STK-LAN

NSS	SEL	CS1	CS2
0	0	1	0
0	1	0	1
1	0	1	1
1	1	1	1

Tabelle 2.5.: Logiktablelle zur Generierung der Chipselect-Signale

$$CS1 = \overline{SEL} \wedge (\overline{NSS} \vee NSS) \vee (NSS \wedge SEL)$$

Dieser Term ergibt vereinfacht:

$$CS1 = \overline{SEL} \wedge 1 \vee (NSS \wedge SEL)$$

$$CS1 = \overline{SEL} \vee (NSS \wedge SEL)$$

Bei diesem Ausdruck wird \overline{SEL} in die Klammer hineingezogen und der Term anschließend vereinfacht.

$$CS1 = (\overline{SEL} \vee NSS) \wedge (\overline{SEL} \vee SEL)$$

$$CS1 = (\overline{SEL} \vee NSS) \wedge 1$$

$$CS1 = \overline{SEL} \vee NSS$$

Da zur Realisierung *NAND-Gatter* verwendet werden sollen, wird der Term für *Not AND (NAND)* angepasst:

$$CS1 = \overline{\overline{\overline{SEL} \vee NSS}}$$

$$CS1 = \overline{\overline{SEL} \wedge \overline{NSS}}$$

$$CS1 = \overline{SEL \wedge \overline{NSS}}$$

Gleichermaßen ergibt sich für das Signal CS2 die folgende Gleichung:

$$CS2 = (\overline{NSS} \wedge SEL) \vee (NSS \wedge \overline{SEL}) \vee (NSS \wedge SEL)$$

Diese Gleichung wird durch boolsche Algebra umgeformt und ergibt:

$$CS2 = \overline{\overline{SEL} \wedge \overline{NSS}}$$

Das Signal «NSS» steht für «Not Slave Select» und kann auch als \overline{SS} ausgedrückt werden.

KAPITEL 2. HARDWARE DES STK-LAN

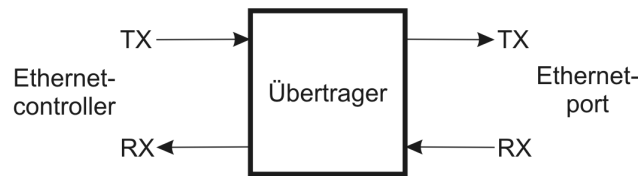


Abbildung 2.12.: Schnittstellen der Übertrager

Aus diesen Zusammenhängen ergibt sich die Ansteuerlogik, welche in Abbildung F.4 des Schaltplans dargestellt ist.

2.3.6. Übertrager

2.3.6.1. Schnittstellen der Übertrager

Die *Übertrager* besitzen zwei Kanäle: den einen für die Sendedaten (TX), den anderen für die Empfangsdaten (RX). Jeder Ein- und Ausgang eines Kanals besitzt zwei Anschlüsse für die Datenleitungen sowie einen dritten Anschluss für eine Verbindung zu *GND*.

2.3.6.2. Funktionalität

Zur galvanischen Entkopplung des *STK-LAN* vom restlichen Netzwerk ist an jedem Ethernetport ein *Übertrager* vorhanden. Durch die *Übertrager* werden beispielsweise *Brummschleifen* oder ein eventuell vorhandener *DC-Anteil* auf der Netzwerkleitung unterdrückt.

2.3.6.3. Realisierung

Bei der Realisierung der Entkopplung wurden bei der Entwicklung des *STK-LAN* zwei Methoden gewählt:

- Entkopplung mit externem Übertrager

Bei Ethernetport zwei ist der *Übertrager* durch einen externen *Übertrager* vom Typ *TG42-1406N1* [49] der Firma *HALO Electronic Inc.* [59] realisiert. Dem Übertrager folgt anschließend eine einfache *8P8C-Modularbuchse* zum Anschluss des Netzkabels.

KAPITEL 2. HARDWARE DES STK-LAN

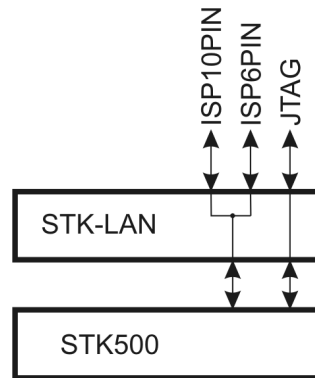


Abbildung 2.13.: Schnittstellen für die Programmieradapter

- Entkopplung mit einem *MagJack*

Ethernetport eins hingegen verwendet einen sogenannten *MagJack* vom Typ *SI-40141* [10] der Firma *Bel Fuse Inc.* [56]. Dieser *MagJack* besteht aus einer *8P8C-Modularbuchse* mit integriertem *Übertrager* und *LEDs*.

Bei der Entwicklung des *STK-LAN* wurden gezielt beide Versionen von Übertragern, also ein externer und ein integrierter, verwendet, um den Studierenden im Rahmen des Mikrocomputerlabors beide Möglichkeiten aufzeigen zu können.

Die genaue Beschaltung des Übertragers beziehungsweise des *MagJack* kann dem Schaltplan in Abbildung F.2 und F.3 entnommen werden.

2.3.7. Programmierschnittstellen

2.3.7.1. Schnittstellen zur Programmierung

Zum Anschluss der Programmieradapter stehen drei Schnittstellen zur Verfügung. Diese haben die Pinbelegungen nach den Tabellen 2.2(a), 2.2(b) und 2.3.

2.3.7.2. Funktionalität

Über die drei Steckverbinder in Abbildung 2.13 sind die folgenden Programmierschnittstellen zugänglich:

KAPITEL 2. HARDWARE DES STK-LAN

- SPI

Zur *SPI-Programmierung* stehen die beiden Wannenstecker *ISP6PIN* und *ISP10PIN* zur Verfügung. Durch diese ist eine geeignete Schnittstelle für die gängigen Programmieradapter verfügbar. Soll der verwendete Mikrocontroller direkt über das *STK500* programmiert werden, kann die Stiftleiste mit der Bezeichnung *ISP6PIN* auf dem *STK500* über das dem *STK500* beiliegende 6-adrige Kabel direkt mit dem Wannenstecker *ISP6PIN* auf dem *STK-LAN* verbunden werden. Es ist hierbei darauf zu achten, dass Pin eins des *STK500* mit Pin eins des *STK-LAN* verbunden wird.

- Joint Test Action Group (JTAG)

Die eventuell vorhandene *JTAG-Schnittstelle* des *AVRs* ist über den Wannenstecker *JTAG* erreichbar. Mit Hilfe dieses Steckers kann beispielsweise das *Atmel JTAGICE mkII* an das *STK-LAN* angeschlossen und zum Programmieren und Debuggen des Zielsystems verwendet werden.

2.3.7.3. Realisierung

Durch die Programmierschnittstellen auf dem *STK-LAN* werden die Signalleitungen des *SPI*- und *JTAG-Anschlusses* des *STK500* herausgeführt. Die beiden Anschlüsse *ISP6PIN* und *ISP10PIN* bieten die gleiche Funktionalität, da deren Signalleitungen parallel geschaltet sind. An diesen beiden Anschlüssen kann zu einem Zeitpunkt also jeweils nur ein Programmiergerät angeschlossen werden. Für weiterführende Details zur Beschaltung der Programmieranschlüsse wird an dieser Stelle auf den Schaltplan in Abbildung F.4 verwiesen.

2.3.8. Sonstiges

Das *STK-LAN* besitzt den Kippschalter *S1* zum Ein-/Ausschalten der Versorgungsspannung. *LED1* dient zur Anzeige einer vorhandenen 3,3-Volt-Versorgungsspannung. *LED4* und *LED5* visualisieren das *Chipselect-Signal* des jeweiligen Ethernetports. Dadurch ist ersichtlich, welcher Ethernetcontroller vom Mikrocontroller angesprochen wird. Beide Ethernetports besitzen weiterhin zwei *LEDs* mit den Bezeichnungen *ACT* und *LINK*, welche dazu dienen, eine bestehende Netzwerkverbindung (*LINK*) sowie eine aktive Datenübertragung (*ACT*) anzuzeigen.

KAPITEL 2. HARDWARE DES STK-LAN

3. Software des Mikrocontrollerprogramms

Zu Beginn dieser Diplomarbeit wurde mit dem *uIP-Stack* von *Adam Dunkels* [30] gearbeitet. Der Einstieg in diesen Stack erwies sich allerdings als relativ schwer und da das Ergebnis dieser Diplomarbeit für Ausbildungszwecke zum Einsatz kommen soll, musste ein leicht zu verstehender Netzwerkstack gefunden werden. Die Entscheidung fiel auf den Netzwerkstack von *Simon Schulz*, welcher unter der *GPL* frei verfügbar ist. Dieser Programmcode ist weitestgehend selbsterklärend. Das Projekt zu diesem Stack ist auf der Website von *Simon Schulz* [115] unter dem Namen *avrETH1* zu finden. Dieser Stack unterstützt die folgenden Protokolle: *Internet Protocol (IP)*, *Address Resolution Protocol (ARP)*, *Internet Control Message Protocol (ICMP)*, *Transmission Control Protocol (TCP)*, *User Datagram Protocol (UDP)* sowie das *Hypertext Transfer Protocol (HTTP)*. Einige Teile des Stacks stammen von *Ulrich Radig* [104] oder aus der *Procyon AVRlib* [121] von *Pascal Stang*. Bei den Implementierungen der Netzwerkprotokolle wurden weitestgehend nur die wesentlichen Funktionen implementiert, da es nicht möglich ist, die Protokolle mit allen Details in einen 8-Bit Mikrocontroller zu integrieren. Im *Sourcecode* sind die wichtigsten Einschränkungen mit Kommentaren versehen. Die in diesem Kapitel verwendeten Struktogramme sollen eine Übersicht über den Aufbau der einzelnen Teile des Netzwerkstacks bieten. Sie enthalten daher nur die wichtigsten Abläufe¹. Für genauere Informationen zur Implementierung der Funktionen wird auf den Quellcode verwiesen. Abbildung 3.1 zeigt den schematischen Aufbau des Mikrocontrollerprogramms.

¹Würden die Struktogramme alle Details der jeweiligen Routine und alle verwendeten Variablen enthalten, würde dies den Rahmen der Struktogramme sprengen.

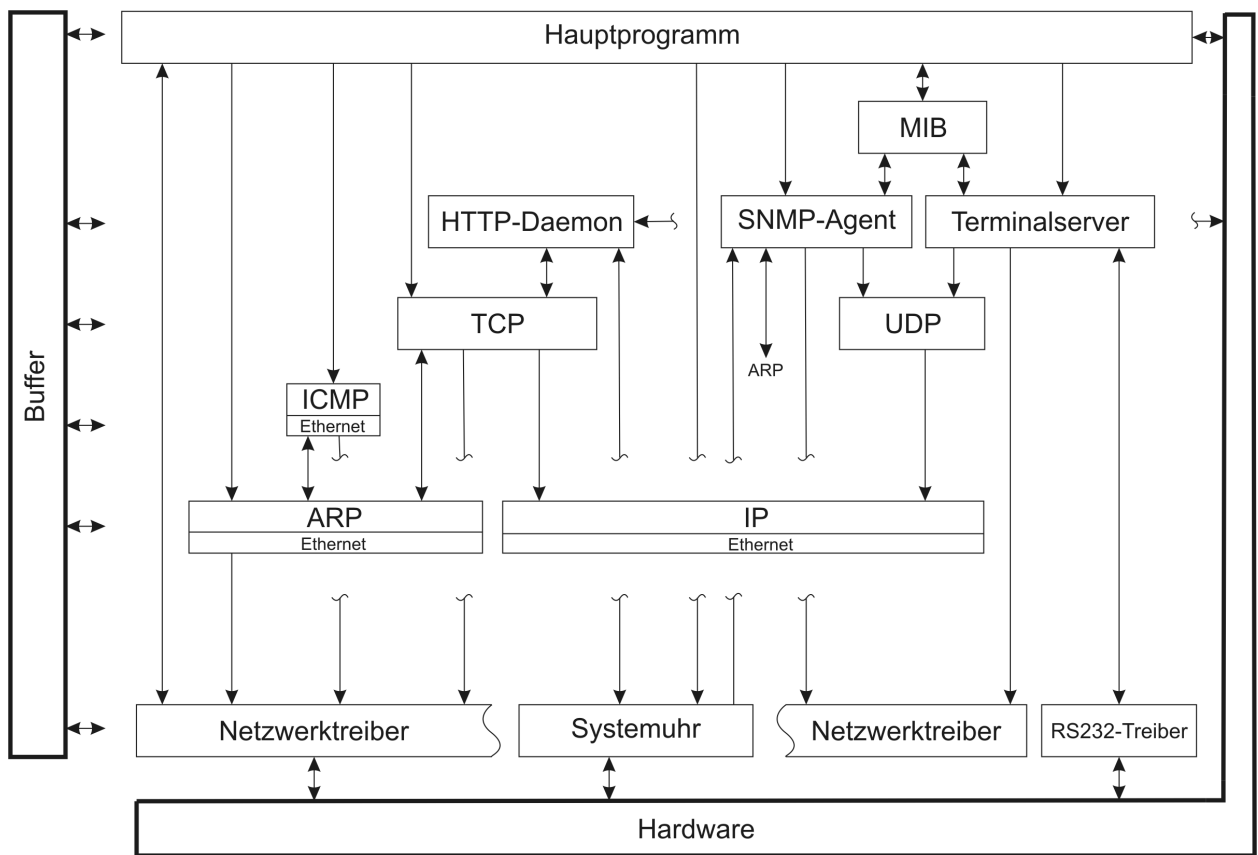


Abbildung 3.1.: Blockschaltbild des Mikrocontrollerprogramms

3.1. Schnittstellen

Die Schnittstellen nach Abbildung 3.2 sind wie folgt definiert:

- Terminalserver

Der Terminalserver wird über die Funktionen *void terminalserver_receive_udp(unsigned char *buffer, unsigned int len)* und *void terminalserver_receive_uart(void)* aufgerufen.

- TCP

Über den Aufruf *void tcp_packet_in(unsigned char *buffer, unsigned int len)* wird die Verarbeitung eines *TCP-Pakets* gestartet und über *void tcp_ttl_cleanup(void)* werden die *TTLs* der *TCP-Verbindungen* überprüft.

- SNMP-Agent

Über *void snmp_agent(unsigned char *buffer, unsigned int len)* werden die empfangenen Daten dem *SNMP-Agent* übergeben. Durch einen Aufruf der Funktion *void snmp_add_to_trap_out(unsigned char Generic_Trap, unsigned char Enterprise_Trap, unsigned long int Dest_IP, unsigned int Timestamp, unsigned char Managed_Object_Name)* können *TRAP-Nachrichten* zum Senden bereitgestellt werden. Ein anschließender Aufruf der Funktion *void snmp_agent_send(unsigned char *buffer)* sendet ausstehende Pakete.

Zum Abgleich der *Managed Information Base* mit den Variablen des Systems wird die Funktion *void MIB_Sync(void)* verwendet.

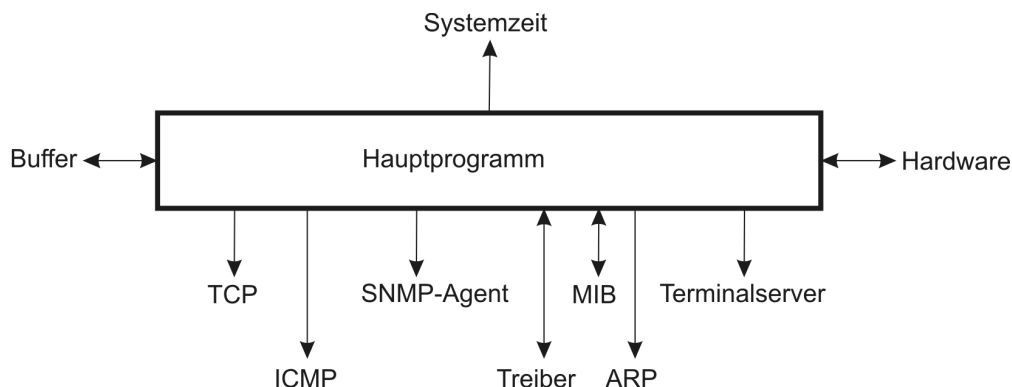


Abbildung 3.2.: Schnittstellen des Hauptprogramms

KAPITEL 3. SOFTWARE DES MIKROCONTROLLERPROGRAMMS

- ARP

Der Funktionsaufruf *void arp_packet_in(unsigned char *buffer, unsigned int len)* startet die Verarbeitung einer *ARP-Anfrage*.

- ICMP

ICMP wird über einen Aufruf von *void icmp_packet_in(unsigned char *buffer, unsigned int len)* ausgeführt.

- Netzwerktreiber

Ein direkter Zugriff auf den Treiber der *Ethernetcontroller* findet im Hauptprogramm über den Aufruf der Funktion *inline unsigned int nic_receive_packet(unsigned char *buffer, unsigned int maxlen)* statt.

- Buffer

Das Ansprechen des *Buffers* erfolgt über Zugriffe auf dessen *Array*. Ein Beispiel hierzu ist: `«type = (buffer[12]«8) + buffer[13];»`.

- Systemuhr

Um die aktuelle Systemzeit zu bestimmen, wird im Hauptprogramm regelmäßig die Funktion *void clock_do(void)* aufgerufen. Zusätzlich aktualisiert diese Routine den *Timestamp* für den *SNMP-Agent*.

- Hardware

Für direkte Hardwarezugriffe spricht das Hauptprogramm die Mikrocontrollerports direkt an, wie beispielsweise über `«DDRD = (1«4) | (1«5) | (1«6);»`.

3.2. Funktionalität

Das Hauptprogramm hat die Aufgabe, das System nach einem Start zu initialisieren, ankommende Datenpakete entgegenzunehmen und diese anschließend an die dafür bestimmte Anwendung weiterzureichen. Zusätzlich aktualisiert das Hauptprogramm regelmäßig die Systemzeit, sendet ausstehende *TRAP-Nachrichten* und synchronisiert die *MIB* mit dem System.

3.3. Realisierung

Das Hauptprogramm ist in den Dateien *main.c* und *main.h* implementiert. Wie das Struktogramm «main.c» verdeutlicht, besteht das Mikrocontrollerprogramm aus einer Endlosschleife, in welcher regelmäßig der Empfangspuffer auf neu eingetroffene Pakete überprüft wird. Für den Fall, dass ein neues Paket angekommen ist, wird dieses in den *Buffer* des Netzwerkstacks übertragen und anschließend kontrolliert, ob dieses ausgewertet werden kann/muss. Nach der Auswertung des Empfangspuffers wird die interne Systemzeit neu ermittelt. Nach jeder Sekunde werden die *TCP-Verbindungen* auf abgelaufene *TTLs* überprüft und eventuell geschlossen. Zusätzlich werden jede Sekunde anstehende *TRAP-Nachrichten* versendet sowie die *Managed Objects* aktualisiert. In der Datei *main.c* kann über die beiden Zeilen

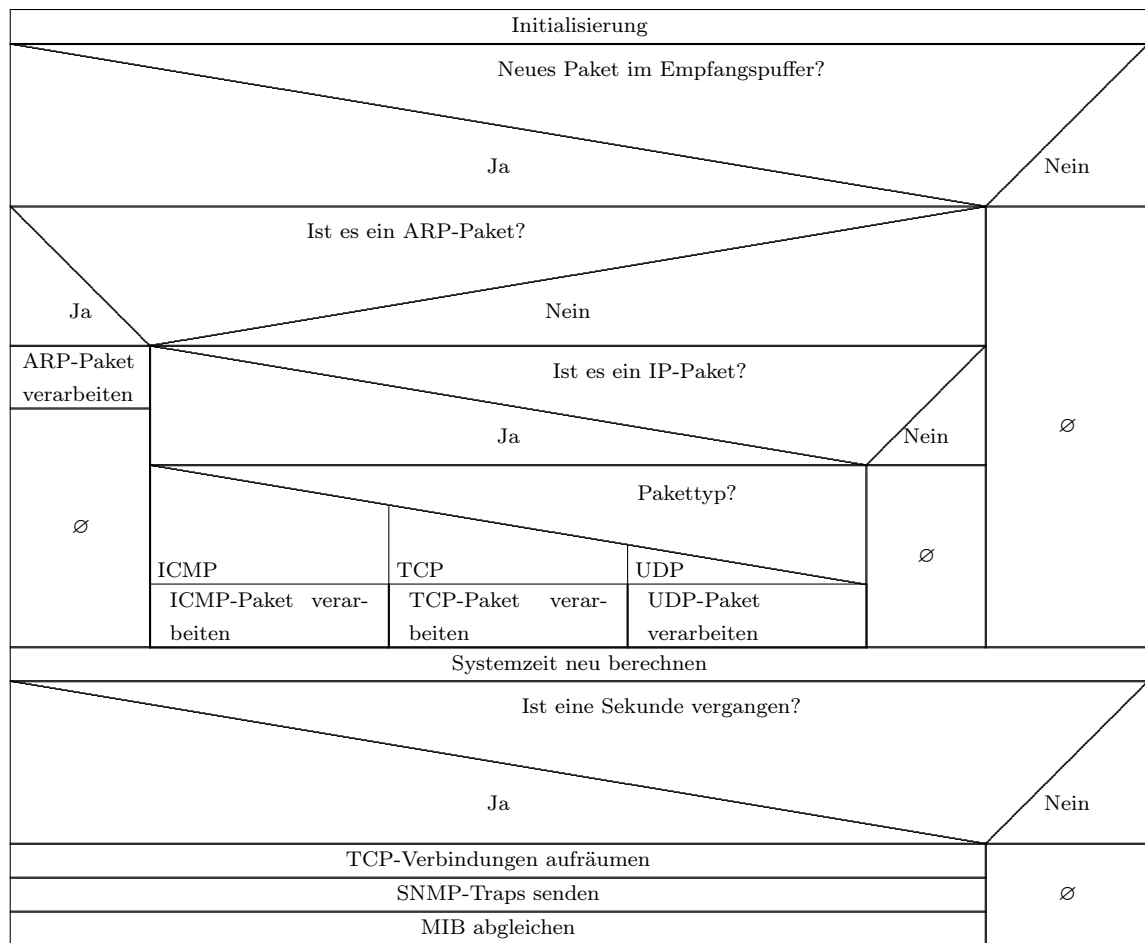
```
PORTB |= (1<<PB3); //Select Ethernet_1 on STK_LAN
//PORTB |= (0<<PB3); //Select Ethernet_2 on STK_LAN
```

der zu verwendende Ethernetcontroller ausgewählt werden.

Durch die Betätigung des Tasters *SW3* auf dem *STK500* kann dem *TRAP-Buffer* eine Nachricht hinzugefügt werden.

KAPITEL 3. SOFTWARE DES MIKROCONTROLLERPROGRAMMS

main.c



3.3.1. Netzwerktreiber

3.3.1.1. Schnittstellen des Netzwerktreibers

Das Empfangen von Daten geschieht über einen Aufruf der Funktion *inline unsigned int nic_receive_packet(unsigned char *buffer, unsigned int maxlen)*, das Senden von Daten über die Funktion *inline void nic_send_packet(unsigned char *buffer, unsigned int len)*. Die Funktion zum Senden von Daten erhält als Parameter einen Zeiger auf den Anfang der zu sendenden Daten und die Anzahl der Bytes, welche zu senden sind. Die Funktion zum Empfangen der Ethernetdaten erhält hingegen einen Zeiger auf den Beginn des Speichers, in welchen die empfangenen Daten kopiert werden sollen und eine Angabe zur maximalen Länge des zu kopierenden Datenblocks. Die Empfangsfunktion gibt einen Wert zurück, welcher die Länge des enthaltenen Ethernetpakets angibt.

Zum Zugriff auf die Hardware werden direkte Registerzugriffe verwendet. Die empfangenen beziehungsweise zu sendenden Daten werden über den *Buffer* des Mikrocontrollerprogramms verarbeitet.

3.3.1.2. Funktionalität

Der Treiber für die *Ethernetcontroller* übernimmt die komplette Ansteuerung der *ENC28J60-Controller*, inklusive der *SPI-Kommunikation*. Im restlichen Mikrocontrollerprogramm müssen somit nur die Funktionen zum Senden und Empfangen von Daten aufgerufen werden.

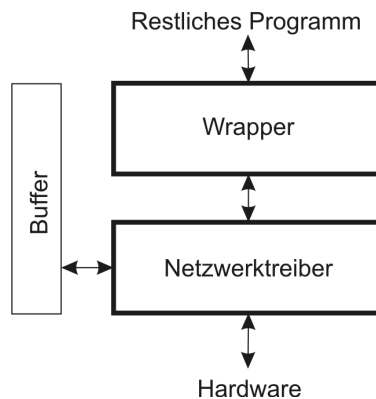


Abbildung 3.3.: Schnittstellen des Treibers

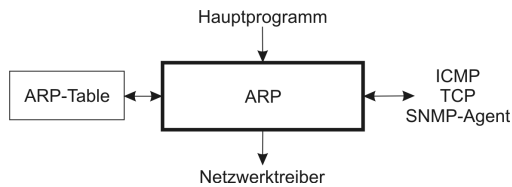


Abbildung 3.4.: Schnittstellen von ARP

3.3.1.3. Realisierung

Zur Realisierung des Treibers wird die *Procyon AVRlib* [121] von *Pascal Stang* verwendet. Der Treiber ist in den beiden Dateien *enc28j60.c* und *enc28j60.h* implementiert. Diese Dateien enthalten mehrere C-Funktionen, von denen allerdings nur zwei außerhalb dieser Dateien interessant sind. Dies ist die Funktion `void enc28j60_send_packet(unsigned char *buffer, unsigned int len)` zum Senden und die Funktion `unsigned int enc28j60_receive_packet(unsigned char *buffer, unsigned int maxlen)` zum Empfangen von Ethernetpaketen. Die Portpins zur Kommunikation mit den *Ethernetcontrollern* werden in der Datei *config.h* angegeben.

Um einen einfachen Austausch des Treibers zu ermöglichen, ist, wie Abbildung 3.3 zeigt, zwischen dem Treiber und dem Mikrocontrollerprogramm ein *Wrapper* eingefügt. Dieser dient zur Adaptierung der Treiberschnittstelle an das Mikrocontrollerprogramm und ist in den Dateien *mynic.c* und *mynic.h* implementiert.

Die Konfiguration der Netzwerkschnittstelle, also das Einstellen von *IP*, *Gateway* und *MAC-Adresse*, ist in Anhang J erläutert.

3.3.2. ARP

3.3.2.1. Schnittstellen von ARP

Zur Verwendung der ARP-Implementierung stehen die folgenden vier Funktionen zur Verfügung:

- `void arp_packet_in(unsigned char *buffer, unsigned int len)`

Zum Empfangen einer Anfrage und Generierung der zugehörigen Antwort.

- `void arp_send_request(unsigned char *buffer, uint32_t *dest_ip)`

Zum Senden einer Anfrage und Empfangen der zugehörigen Antwort.

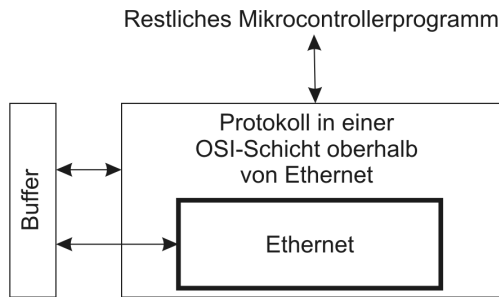


Abbildung 3.5.: Schnittstellen des Ethernet Protocols

- `int arp_search_by_ip(uint32_t ip)`

Zum Suchen einer *MAC-Adresse* anhand der zugehörigen *IP-Adresse*.

- `int arp_add_mac2ip(unsigned char *buffer, unsigned long ip)`

Für das Hinzufügen einer neuen *MAC-IP-Kombination* zur *ARP-Table*.

3.3.2.2. Funktionalität

Die ARP-Funktionalität des Netzwerkstacks dient zum Finden einer *MAC-Adresse* zu einer bekannten *IP-Adresse*. Hierzu werden bekannte *MAC-IP-Kombinationen* in einer Tabelle (Abbildung 3.4) gespeichert. Zum Hinzufügen neuer Kombinationen können die *MAC-IP-Kombinationen* aus empfangenen Paketen ausgelesen oder eigene *ARP-Requests* gestellt werden.

3.3.2.3. Realisierung

Die Funktionen für das *Address Resolution Protocol* sind in den Dateien `arp.c` und `arp.h` des *avrETH1-Netzwerkstacks* [115] implementiert. Die *MAC-IP-Adresspaare* werden über eine Tabelle verwaltet, deren Anzahl von möglichen Einträgen über die Konstante `ARP_TABLE_SIZE` in der Datei `arp.h` angepasst werden kann.

3.3.2.3.1. Ethernet Protocol Die Generierung der *Ethernet-Frames* ist im *avrETH1-Netzwerkstack* nicht explizit durch eine Funktion gelöst. Stattdessen generiert jede Funktion, die *Ethernet* benötigt, die Daten selbst und schreibt diese in den *Buffer* (Abbildung 3.5).

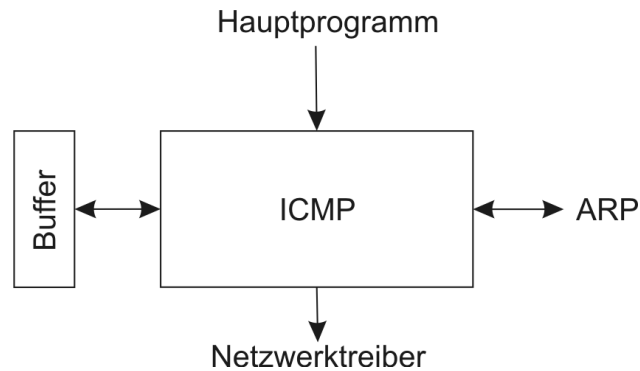


Abbildung 3.6.: Schnittstellen von ICMP

3.3.3. ICMP

3.3.3.1. Schnittstellen von ICMP

ICMP wird vom Hauptprogramm über die Funktion `void icmp_packet_in(unsigned char *buffer, unsigned int len)` aufgerufen. Der *Pointer* mit der Bezeichnung «*buffer» zeigt auf das *Array* mit den empfangenen Netzwerkdaten. Die Angabe «len» gibt die Länge der empfangenen Daten an.

Zum Auflösen der *IP-Adressen* greift *ICMP* über den Funktionsaufruf `int arp_search_by_ip(uint32_t ip)` auf die Dienste von *ARP* zu (Abbildung 3.6).

Die erzeugten Daten werden in den *Buffer* geschrieben und durch den Aufruf des Treibers mit `inline void nic_send_packet(unsigned char *buffer, unsigned int len)` gesendet. Die Angabe «*buffer» enthält hierbei einen *Zeiger* auf die zu sendenden Daten und «len» gibt die Länge der zu sendenden Daten an.

3.3.3.2. Funktionalität

Die Funktionalität des *ICMP-Teils* liegt in der Beantwortung von *Pings*, also dem Entgegennehmen von *ICMP Echo Requests* und dem Generieren von *ICMP Echo Replies*.

3.3.3.3. Realisierung

Zur Beantwortung eines *ICMP Echo Requests* werden in der Datei *icmp.c* die benötigten Stellen eines empfangenen Netzwerkpakets angepasst und dieses anschließend wieder gesendet. Die hierbei

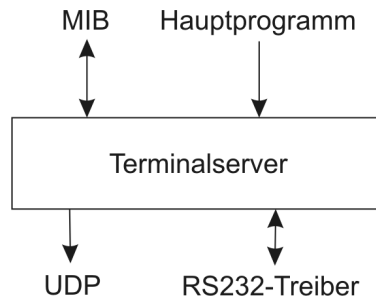


Abbildung 3.7.: Schnittstellen des Terminalservers

nötige Ethernetfunktionalität wird wie in Kapitel 3.3.2.3.1 beschrieben innerhalb der *ICMP-Routine* realisiert. In der Datei *icmp.h* werden hierzu einige Konstanten definiert.

3.3.4. Terminalserver

Damit die Studierenden im Labor langsam an die Netzwerkkommunikation herangeführt werden können, ist im Mikrocontrollerprogramm zusätzlich ein kleiner Terminalserver implementiert. Dieser kann über die serielle Schnittstelle mit der Bezeichnung *RS232 SPARE* des *STK500* und über das Netzwerk per *UDP* angesprochen werden.

Auf diese Weise wird ermöglicht, im Labor zuerst über eine serielle Verbindung eine Terminalanwendung vorzustellen. Anschließend kann auf den gleichen Befehlen basierend die Kommunikation anstatt über die serielle Schnittstelle, über eine *UDP-Verbindung* realisiert werden. Nachdem den Studierenden auf diese Weise der Übergang einer einfachen seriellen Verbindung zu einer Netzwerkverbindung veranschaulicht wurde, kann gezeigt werden, dass für diese Kommunikation nun noch zusätzlich standardkonforme Protokolle, wie beispielsweise das *Simple Network Management Protocol (SNMP)*, verwendet werden können.

3.3.4.1. Schnittstellen des Terminalservers

Der Terminalserver verwendet im Mikrocontrollerprogramm die Schnittstellen nach Abbildung 3.7:

- `void terminalserver_receive_uart(void)`

Durch Aufruf dieser Funktion überprüft der Terminalserver den Empfangspuffer der seriellen Schnittstelle auf neue Daten und verarbeitet diese.

KAPITEL 3. SOFTWARE DES MIKROCONTROLLERPROGRAMMS

- *void terminalserver_receive_udp(unsigned char *buffer, unsigned int len)*

Mit einem Aufruf dieser Funktion werden dem Terminalserver ein *Zeiger* auf den *Buffer* sowie die Länge der Daten im *Buffer* übergeben. Anschließend verarbeitet der Terminalserver eventuell vorhandene Daten.

- Zugriff auf die *Managed Information Base*

Durch Zugriff auf das Array *extern struct mib_var mib_variable[SNMP_MAXIMUM_NUMBER_OF_OID]*; greift der Terminalserver lesend und schreibend auf die *MIB* zu.

- *extern void uart_puts_p(const char *s)*

Mit Hilfe dieser Funktion sendet der Terminalserver einen *String* an die serielle Schnittstelle. Dem Funktionsaufruf wird ein mit «\0» terminierter String übergeben.

- *void udp_generate_packet(unsigned char *buffer, uint32_t *dest_ip, unsigned char *dest_mac, unsigned int source_port, unsigned int dest_port, unsigned int data_length)*

Durch Verwendung dieser Funktion übergibt der Terminalserver seine zu sendenden Daten an den *UDP-Teil* des Netzwerkstacks. Die Angabe «*buffer» enthält den *Zeiger* auf die zu sendenden Daten. «*dest_ip», «*dest_mac» und «dest_port» übergeben die Daten zur Adressierung des Zielsystems. Mit der Angabe «source_port» wird die Portnummer des Senders übergeben. Der Wert dieser Angabe ist nicht relevant, da es sich bei der *UDP-Verbindung* um eine unidirektionale Verbindung handelt. Der Parameter «data_length» übergibt dem *UDP-Teil* die Länge der zu sendenden Daten.

Beide Kommunikationswege - also die serielle Schnittstelle und *UDP* - nehmen die beiden folgenden Kommandos entgegen. Bei der Kommunikation über die serielle Schnittstelle müssen die Befehlsstrings mit einem *Carriage Return (CR)* (ASCII-Wert 13) abgeschlossen werden. Bei Verwendung von *UDP* ist dieses Zeichen nicht nötig, da der Befehlsstring hierbei durch das *UDP-Paket* begrenzt wird.

- *get LED x*

Dieser Befehl steht zum Auslesen der Werte der *LEDs* auf dem *STK500* zur Verfügung. Die Angabe «x» kann hierbei die beiden Werte «0» (LED0) und «1» (LED1) annehmen. Der Befehl «get LED 0» liest beispielsweise den aktuellen Wert von LED0 aus.

KAPITEL 3. SOFTWARE DES MIKROCONTROLLERPROGRAMMS

- *set LED x y*

Durch Verwendung dieses Befehls können die Werte der *LEDs* des *STK500* gesetzt werden. Die Angabe «x» adressiert hierbei die *LED* und die Angabe «y» übergibt den gewünschten Wert. LED1 kann beispielsweise über «set LED 1 0» auf den Wert null gesetzt werden.

Die vom Terminalserver verwendeten Statusmeldungen, welche über die serielle Schnittstelle oder *UDP* zurückgesendet werden, lauten wie folgt:

- «ok\r\n»
- «unknown parameter\r\n»
- «Value = 1\r\n»
- «Value = 0\r\n»
- «unknown command\r\n»

3.3.4.2. Funktionalität

Der Terminalserver stellt eine einfache Möglichkeit zur Kommunikation mit dem Mikrocontrollerprogramm zur Verfügung. Diese Kommunikation kann über die serielle Schnittstelle des *STK500* und über eine Netzwerkverbindung mittels *UDP* erfolgen. Der Terminalserver verwendet für beide Kommunikationswege, mit Ausnahme der Funktionen zum Ansprechen der Schnittstellen, die selben Routinen. Hierdurch kann der Übergang von der *RS232*- zur *UDP-Kommunikation* leicht nachvollzogen werden.

3.3.4.3. Realisierung

Der Terminalserver besteht aus vier Funktionen. Die beiden Funktionen *void terminalserver_receive_uart(void)* und *void terminalserver_receive_udp(unsigned char *buffer, unsigned int len)* empfangen den Befehlsstring über die *RS232-Schnittstelle* beziehungsweise über *UDP* und rufen anschließend die Funktion *void terminalserver(void)* auf. Diese Funktion analysiert den empfangenen Befehlsstring und führt diesen gegebenenfalls aus. Wie in Tabelle 3.1 veranschaulicht, greift der Terminalserver hierbei nicht direkt auf die Hardware zu. Stattdessen werden die Daten zuerst in die *Managed Information Base* des *SNMP-Agent* eingetragen, von welchem diese anschließend mit dem System synchronisiert werden. Nach dem Aktualisieren der *MIB* wird über die Funktion *void Send_Terminal_Response(const char *string, unsigned char Command_Receiver, unsigned short*

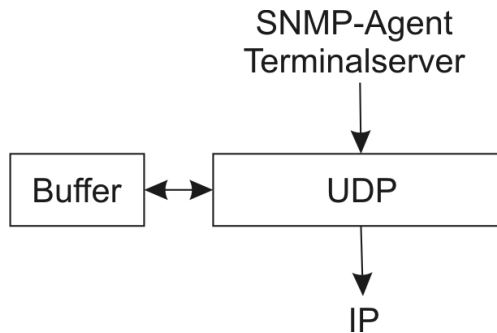


Abbildung 3.8.: Schnittstellen von UDP

*int Length, unsigned long int Destination_IP, unsigned int Destination_Port, unsigned char *buffer*) eine *Response* an die *RS232*- oder *Ethernet*-Schnittstelle gesendet.

	Managed Information Base	
Anwendungsschicht	SNMP	Terminalserver
Transportschicht	UDP	
Internetschicht	IP	
Netzzugangsschicht	Ethernet	RS232

Tabelle 3.1.: Position des Terminalserver im TCP/IP-Referenzmodell

Die Konfiguration des *Terminalservers* geschieht über die Datei *config.h*, in welcher die *IP-Adresse* des Terminal-Clients sowie die *Baudrate* der *RS232-Schnittstelle* angegeben werden. In der Datei *ip.h* wird über die Variable *IP_PORT_TERMINALSERVER* der *UDP-Port* definiert, den der Terminalserver verwendet. In der Datei *terminalserver.h* werden die maximal empfangbare Befehlslänge und die *Response-Nachrichten* des Terminalservers definiert.

3.3.4.3.1. UDP

3.3.4.3.1.1. Schnittstellen von UDP Der *UDP-Teil* des Netzwerkstacks wird von darüberliegenden Schichten des *OSI-Modells* über die Funktion *void udp_generate_packet(unsigned char *buffer, uint32_t *dest_ip, unsigned char *dest_mac, unsigned int source_port, unsigned int dest_port, unsigned int data_length)* aufgerufen. Die hierbei übergebenen Parameter sind der Reihenfolge nach:

- Ein *Zeiger* auf die zu sendenden Daten

KAPITEL 3. SOFTWARE DES MIKROCONTROLLERPROGRAMMS

- Die *IP-Adresse* des Empfängers
- Die *MAC-Adresse* des Empfängers
- Der *Netzwerkport*, von dem aus gesendet wird
- Der *Netzwerkport* auf der Seite des Empfängers
- Die Länge der zu sendenden Daten

Nachdem der *UDP-Teil* seine Daten dem *Buffer* über die Schnittstelle (Abbildung 3.8) hinzugefügt hat, gibt er über die Funktion `void ip_generate_packet(unsigned char *buffer, uint32_t *dest_ip, unsigned char *dest_mac, unsigned int source_port, unsigned int dest_port, unsigned char ip_packettype, unsigned int data_length)` die Daten an die darunterliegende *IP-Schicht* weiter. Die übergebenen Parameter sind hierbei die gleichen wie beim Aufruf der *UDP-Routine*, mit dem Unterschied, dass diesmal noch der Parameter «*ip_packettype*» übergeben wird. Dieser zusätzliche Parameter spezifiziert den verwendeten Pakettyp im *Ethernetframe*.

3.3.4.3.1.2. Funktionalität Der *UDP-Teil* des Netzwerkstacks fügt den Daten einen *UDP-Header* hinzu.

3.3.4.3.1.3. Realisierung Der *UDP-Teil* besteht aus einer einzelnen Funktion `void udp_generate_packet(unsigned char *buffer, uint32_t *dest_ip, unsigned char *dest_mac, unsigned int source_port, unsigned int dest_port, unsigned int data_length)`, welche dazu dient, im *Buffer* den *UDP-Header* einzufügen. Innerhalb dieser Funktion wird aus der Datei *ip.c* die Funktion zur Generierung eines *IP-Pakets* aufgerufen. Bevor die Funktion zum Erzeugen eines *UDP-Pakets* aufgerufen wird, müssen die Nutzdaten an die richtige Stelle² im *Buffer* geschrieben werden, da sonst die Prüfsummen in den *Headern* nicht korrekt gebildet werden. Hierbei ist anzumerken, dass die Prüfsummenberechnung im Stack *avrETH1* deaktiviert ist und daher in das *Checksum-Feld* lediglich der Wert null eingetragen wird.

In der Datei *udp.h* ist die Variable *UDP_POS_DATA* definiert, welche den Index auf den Beginn der Nutzdaten im *Buffer* enthält.

²Der Beginn der Nutzdaten eines *UDP-Pakets* ist an der Stelle im *Buffer*, welche durch die Variable *UDP_POS_DATA* indiziert wird.

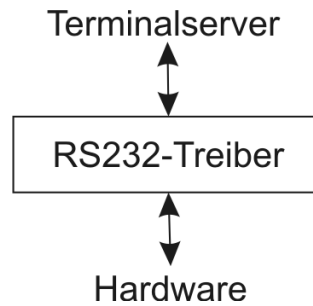


Abbildung 3.9.: Schnittstellen des RS232-Treibers

3.3.4.3.2. RS232-Treiber

3.3.4.3.2.1. Schnittstellen des RS232-Treibers Der *RS232-Treiber* bietet dem Terminalserver zwei Kommunikationsschnittstellen an. Dies sind die Funktion *unsigned int uart_getc(void)* zum Lesen von Daten der seriellen Schnittstelle und die Funktion *void uart_puts_p(const char *progmem_s)* zum Schreiben von Daten. Die erstgenannte Funktion übergibt der aufrufenden Funktion das empfangene Zeichen zusammen mit einigen Statusbits. Die Funktion zum Senden von Daten erhält als Parameter einen *Zeiger* auf den zu sendenden String, welcher mit «\0» terminiert ist.

Auf der Hardwareseite (Abbildung 3.9) greift der Treiber direkt auf die *USART-Register* des *AVR-Mikrocontrollers* zu.

3.3.4.3.2.2. Funktionalität Der Treiber hat die Aufgabe, den *Universal Synchronous and Asynchronous Receiver and Transmitter (USART)* des Mikrocontrollers zu konfigurieren und die Daten des Mikrocontrollerprogramms über den *USART* zu senden beziehungsweise zu empfangen.

3.3.4.3.2.3. Realisierung Zur Kommunikation mit der *RS232-Schnittstelle* wird die *UART-Library* von Peter Fleury [38] verwendet, welche zum Puffern der Daten einen *Ringbuffer* verwendet. Die Größe des *Ringbuffers* kann in der Datei *uart.h* über die beiden Variablen *UART_RX_BUFFER_SIZE* und *UART_TX_BUFFER_SIZE* angepasst werden. Die komplette Library ist in den beiden Dateien *uart.c* und *uart.h* implementiert.

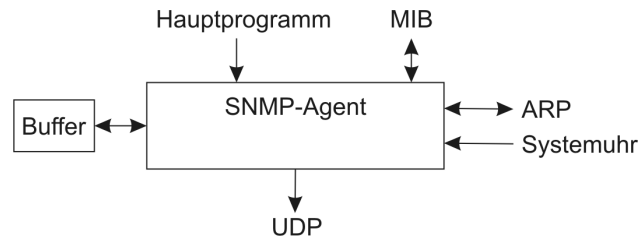


Abbildung 3.10.: Schnittstellen des SNMP-Agent

3.3.5. SNMP-Agent

3.3.5.1. Schnittstellen des SNMP-Agent

Auf den *SNMP-Agent* kann über die folgenden Schnittstellen (Abbildung 3.10) zugegriffen werden:

- `void snmp_add_to_trap_out(unsigned char Generic_Trap, unsigned char Enterprise_Trap, unsigned long int Dest_IP, unsigned int Timestamp, unsigned char Managed_Object_Name)`

Durch Aufruf dieser Funktion kann dem *TRAP-Buffer* ein Eintrag hinzugefügt werden.

- `void snmp_agent(unsigned char *buffer, unsigned int len)`

Der Aufruf dieser Funktion startet das Auswerten des empfangenen Datenpakets durch den *SNMP-Agent*.

- `void snmp_agent_send(unsigned char *buffer)`

Mit dieser Funktion werden alle anstehenden Nachrichten des *SNMP-Agent* gesendet.

- `int arp_search_by_ip(uint32_t ip)`

Diese Funktion sucht die *MAC-Adresse* zu einer bekannten *IP-Adresse* in der *ARP-Tabelle*. Wird in der Tabelle ein passender Eintrag gefunden, gibt die Funktion den Index des Eintrags in der *ARP-Tabelle* zurück.

- `int arp_add_mac2ip(unsigned char *buffer, unsigned long ip)`

Diese Funktion fügt der *ARP-Tabelle* eine *MAC-IP-Kombination* hinzu. Der Rückgabewert indiziert die Stelle, an welcher der neue Eintrag in der Tabelle hinzugefügt wurde.

- `extern struct arp_entry arp_table[ARP_TABLE_SIZE]`

Über einen direkten Zugriff auf die *ARP-Tabelle* können Einträge ausgelesen werden.

KAPITEL 3. SOFTWARE DES MIKROCONTROLLERPROGRAMMS

- *extern unsigned long int Timestamp*

Über lesenden Zugriff auf die Variable *Timestamp* erhält der *SNMP-Agent* einen aktuellen Zeitstempel.

Der *SNMP-Agent* greift während der Verarbeitung der Daten direkt auf den *Buffer* sowie lesend und schreibend auf die *MIB* zu.

Zum Senden der generierten Daten werden diese über die Funktion *void udp_generate_packet(unsigned char *buffer, uint32_t *dest_ip, unsigned char *dest_mac, unsigned int source_port, unsigned int dest_port, unsigned int data_length)* an *UDP* übergeben.

3.3.5.2. Funktionalität

Der *SNMP-Agent* unterstützt die Nachrichtentypen *GET*, *GETNEXT*, *SET* und *TRAP*. Zur Einordnung der verwendeten *Managed Objects* in den *MIB-Tree* wird die offizielle *Private Enterprise Number* der *Fachhochschule Heilbronn* verwendet, welche die Adresse «1.3.6.1.4.1.4766» besitzt. Laut Frau *Heidrun Schmidt* [114] vom Rechenzentrum der *Hochschule Heilbronn* sind an der Hochschule die folgenden *OIDs* bereits durch das *Lightweight Directory Access Protocol (LDAP)* belegt und können somit nicht für das *STK-LAN* verwendet werden:

- 1.3.6.1.4.1.4766.1.1
- 1.3.6.1.4.1.4766.1.2
- 1.3.6.1.4.1.4766.1.3
- 1.3.6.1.4.1.4766.1.4
- 1.3.6.1.4.1.4766.1.5
- 1.3.6.1.4.1.4766.1.6
- 1.3.6.1.4.1.4766.1.7
- 1.3.6.1.4.1.4766.1.8
- 1.3.6.1.4.1.4766.1.9
- 1.3.6.1.4.1.4766.1.10
- 1.3.6.1.4.1.4766.2.1

	Juri Andruschenko	STK-LAN
Flash	46 Kilobyte (KB) (SNMP-Agent)	23 KB (Stack + HTTPD + SNMP-Agent)
RAM	4,6 KB	1,9 KB
Anzahl der Managed Objects	15	4
CPU	40 MHz / 16 Bit	8 MHz / 8 Bit
Zeit für einen GETNEXT-Request	4ms	2ms

Tabelle 3.2.: Vergleich der SNMP-Agent-Implementierungen

- 1.3.6.1.4.1.4766.2.2

Die *Managed Objects* des *STK-LAN* sind daher unter der *OID* «1.3.6.1.4.1.4766.3» zu finden.

An dieser Stelle soll die Implementierung des *SNMP-Agent* dieser Diplomarbeit mit dem *SNMP-Agent* aus der Diplomarbeit von *Juri Andruschenko* [7] verglichen werden, um eine grobe Abschätzung über die Qualität des implementierten *Agent* zu erlangen. In Tabelle 3.2 sind der benötigte Speicher in *Flash* und *Random Access Memory (RAM)*, die Anzahl der implementierten *Managed Objects* sowie die verwendete *Central Processing Unit (CPU)* und die Zeit für eine *GETNEXT* Anfrage gegenübergestellt.

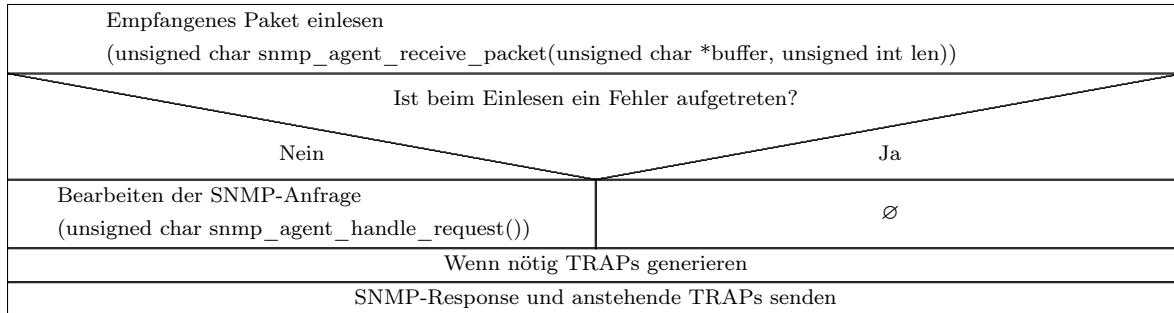
3.3.5.3. Realisierung

Bei der Realisierung wurde besonders darauf geachtet, dass der *SNMP-Agent* auf ein bestehendes Programm aufgesetzt werden kann. Während der Verarbeitung einer *SNMP-Anfrage* durch die Funktion `void snmp_agent(unsigned char *buffer, unsigned int len)` greift der *Agent* nur auf die *Managed Objects* zu, nicht aber direkt auf die Variablen und Hardware des Zielsystems. Erst durch den periodischen Aufruf der Funktion `void MIB_Sync(void)` werden die Daten zwischen Hauptprogramm und *SNMP-Agent* synchronisiert.

Die Hauptfunktionen des *SNMP-Agent* befinden sich in den beiden Dateien `snmp_agent.c` und `snmp_agent.h`. Wird ein neues Paket empfangen und als *SNMP-Paket* erkannt, wird in der `main.c` die Funktion `void snmp_agent(unsigned char *buffer, unsigned int len)` aus der `snmp_agent.c`

KAPITEL 3. SOFTWARE DES MIKROCONTROLLERPROGRAMMS

snmp_agent.c



aufgerufen. In dieser Funktion wird das empfangene Paket über *unsigned char snmp_agent_receive_packet(unsigned char *buffer, unsigned int len)* eingelesen und analysiert. Nach erfolgreichem Einlesen wird die Bearbeitung der *SNMP-Requests* durch den Aufruf der Funktion *void snmp_agent_handle_request()* eingeleitet. Anschließend kontrolliert die Funktion *void snmp_agent_generate_trap(unsigned char State)*, ob während der kompletten Verarbeitung des Netzwerkpakets ein Fehler aufgetreten ist und es wird bei Bedarf eine geeignete *TRAP-Nachricht* generiert. Zum Schluss werden eventuell anstehende *TRAP-Nachrichten* sowie die *SNMP-Response* über die Funktion *void snmp_agent_send(unsigned char *buffer)* gesendet. Der Aufbau der SNMP-Abarbeitung kann dem Struktogramm «snmp_agent.c» entnommen werden.

In den Dateien *snmp_datastructure.c* und *snmp_datastructure.h* werden die SNMP-Statusvariablen, der *TRAP-Buffer* sowie das Datenformat der *Managed Information Base* definiert. Zusätzlich sind in der *snmp_datastructure.h* die bei *SNMP* verwendeten *Daten-* sowie *TRAP-Typen* deklariert.

Die Daten einer SNMP-Verarbeitung werden in den folgenden vier Variablen verwaltet:

- *extern struct aktueller_status psw*

Diese Variable enthält Informationen zur Bearbeitung der SNMP-Anfrage

- *extern struct pdu_data PDU_Daten[SNMP_MAX_NUMBER_OF_VARIABLES_PER_PDU]*

In *pdu_data* werden die beiden zu einem *SNMP-Request* gehörenden Variablenbindungen zwischengespeichert.

- *extern struct mib_var mib_variable[SNMP_MAXIMUM_NUMBER_OF_OID]*

Die Variable *mib_var* enthält den kompletten *MIB-Tree* mit allen *Managed Objects*.

KAPITEL 3. SOFTWARE DES MIKROCONTROLLERPROGRAMMS

- *extern struct trap_out trap_out_buffer[MAX_TRAP_OUT_BUFFER]*

Hierbei handelt es sich um den Buffer für anstehende *TRAP-Nachrichten*. Jede zu sendende *TRAP-Nachricht* wird zuerst diesem Buffer hinzugefügt und anschließend aus dem Buffer heraus versendet.

In der Datei *snmp_datastructure.h* sind unter der Rubrik «CONFIGURATION» einige wichtige Variablen zur Konfiguration des *SNMP-Agent* vorhanden. An dieser Stelle wird beispielsweise die Variable *SNMP_COMMUNITY_STRING*, welche den Community-String enthält, definiert.

Die *Managed Objects* sind in der Datei *mib.c* nach folgendem Muster definiert:

```
mib_variable[3].OID[0] = 43;    //{1,3,6,1,4,1,4766,3,2,2,0};
mib_variable[3].OID[1] = 6;    //{43,6,1,4,1,(165,30),3,2,2,0
mib_variable[3].OID[2] = 1;
mib_variable[3].OID[3] = 4;
mib_variable[3].OID[4] = 1;
mib_variable[3].OID[5] = 165;
mib_variable[3].OID[6] = 30;
mib_variable[3].OID[7] = 3;
mib_variable[3].OID[8] = 2;
mib_variable[3].OID[9] = 2;
mib_variable[3].OID[10] = 0;
mib_variable[3].oid_len = 11;
mib_variable[3].name = SNMP_HID_Leds;
mib_variable[3].access = SNMP_READ_WRITE;
mib_variable[3].type = SNMP_INTEGER;
mib_variable[3].wert_len = 1;
mib_variable[3].wertarray[0] = 0x00;
mib_variable[3].Value_changed = 1;
```

Die *OID* wird dabei nach den *Basic Encoding Rules (BER)* kodiert angegeben. Die Adresse «1.-3.6.1.4.1.4766.3.2.2.0» wird also, wie in Anhang A.8.3 beschrieben, als «43.6.1.4.1.165.30.3.2.2.0» angegeben. Beim Namen des *Managed Objects*, in diesem Fall «SNMP_HID_Leds», handelt es sich um einen programminternen Bezeichner, welcher in der Datei *mib.h* definiert wird. Mit der Variable «access» wird der Zugriff auf das Objekt geregelt. Die hierbei möglichen Werte sind *SNMP_READ_WRITE* für lesenden und schreibenden Zugriff, *SNMP_READ_ONLY* für nur lesenden Zugriff

KAPITEL 3. SOFTWARE DES MIKROCONTROLLERPROGRAMMS

und *SNMP_NO_ACCESS*, falls überhaupt kein Zugriff auf dieses *Managed Object* erlaubt ist. Mit «type» wird der Datentyp des Objekts angegeben. Die hierbei erlaubten Werte sind:

- *SNMP_INTEGER*
- *SNMP_OCTET_STRING*
- *SNMP_NULL*
- *SNMP_OID*
- *SNMP_SEQUENZ*
- *SNMP_IP_ADRESSE*
- *SNMP_TIME_STAMP*

Mit «wert_len» muss die verwendete Länge des «wertarray» angegeben werden. Im gezeigten Beispiel beträgt dieser Wert «1», da das Array nur an Stelle «0» einen Wert enthält.

Die Variable «Value_changed» wird verwendet, um in der Funktion zum Synchronisieren der *MIB* mit dem Rest des Programms zu erkennen, wann eine Synchronisation seitens *SNMP* nötig ist. Nach einem Reset enthält diese Variable den Wert «1», damit die zugehörige Variable im Mikrocontrollerprogramm auf den Wert des *Managed Objects* gesetzt wird.

In der Datei *mib.h* wird außerdem über die Variable *ENTERPRISE_OID* die *Private Enterprise Number* des Systems angegeben.

Durch die beiden Dateien *mib_sync.c* und *mib_sync.h* wird die Funktion *void MIB_Sync(void)* implementiert. Diese Funktion wird in der *main.c* sekundlich aufgerufen und synchronisiert die *Managed Objects* mit dem Rest des Programms.

Informationen zum Hinzufügen weiterer *Managed Objects* sind Anhang I zu entnehmen.

3.3.6. TCP

3.3.6.1. Schnittstellen von TCP

TCP verwendet zur Kommunikation die folgenden drei Schnittstellen (Abbildung 3.11):

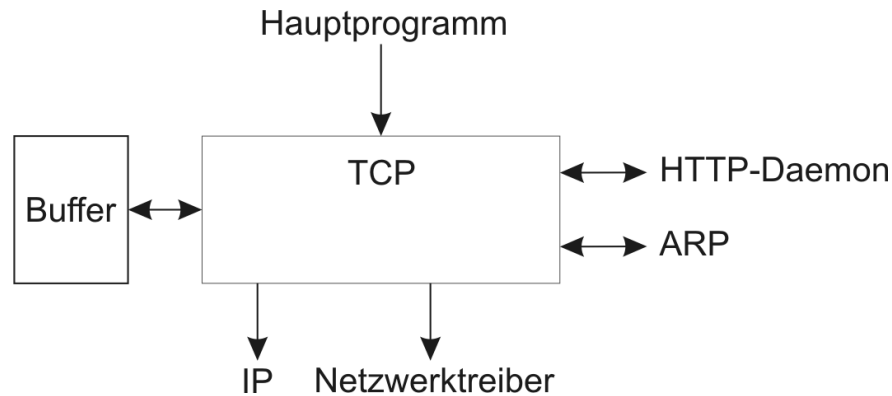


Abbildung 3.11.: Schnittstellen von TCP

- `void tcp_packet_in(unsigned char *buffer, unsigned int len)`

Über diese Funktion wird der *TCP-Teil* des *Stacks* aufgerufen. Die Parameter sind ein Zeiger auf den *Buffer* sowie die Länge der Daten im *Buffer*.

- `unsigned int httpd_data_in(unsigned char *buffer, unsigned int datapos, unsigned int datalen, unsigned char socketnum, unsigned char *appstate)`

Der *HTTP-Daemon* wird vom *TCP-Teil* über diesen Funktionsaufruf gestartet. Nach der Bearbeitung des *HTTP-Teils* wird dem *TCP-Teil* die Länge der zu sendenden Daten übergeben.

- `void ip_generate_packet(unsigned char *buffer, uint32_t *dest_ip, unsigned char *dest_mac, unsigned int source_port, unsigned int dest_port, unsigned char ip_packettype, unsigned int data_length)`

Nachdem *TCP* die zu verwendenden Daten in den *Buffer* übertragen hat, wird die weitere Generierung des Netzwerkpakets über diese Funktion an die darunterliegende Schicht *IP* übergeben.

- `int arp_search_by_ip(uint32_t ip)`

Mit Hilfe dieser Funktion kann die zu einer *IP-Adresse* gehörende *MAC-Adresse* in der *ARP-Tabelle* gesucht werden. Wird der gesuchte Eintrag in der Tabelle gefunden, gibt die Funktion den Index des Eintrags in der *ARP-Tabelle* zurück.

- `int arp_add_mac2ip(unsigned char *buffer, unsigned long ip)`

Durch diese Funktion kann der *ARP-Tabelle* eine *MAC-IP-Kombination* hinzugefügt werden. Der Rückgabewert indiziert die Stelle, an welcher der neue Eintrag in der Tabelle hinzugefügt wurde.

KAPITEL 3. SOFTWARE DES MIKROCONTROLLERPROGRAMMS

- `extern struct arp_entry arp_table[ARP_TABLE_SIZE]`

Über einen direkten Zugriff auf die *ARP-Tabelle* können Einträge ausgelesen werden.

3.3.6.2. Funktionalität

Der *TCP-Teil* übernimmt das Generieren des *TCP-Pakets* und das Kommunizieren mit dem *HTTP-Daemon*.

3.3.6.3. Realisierung

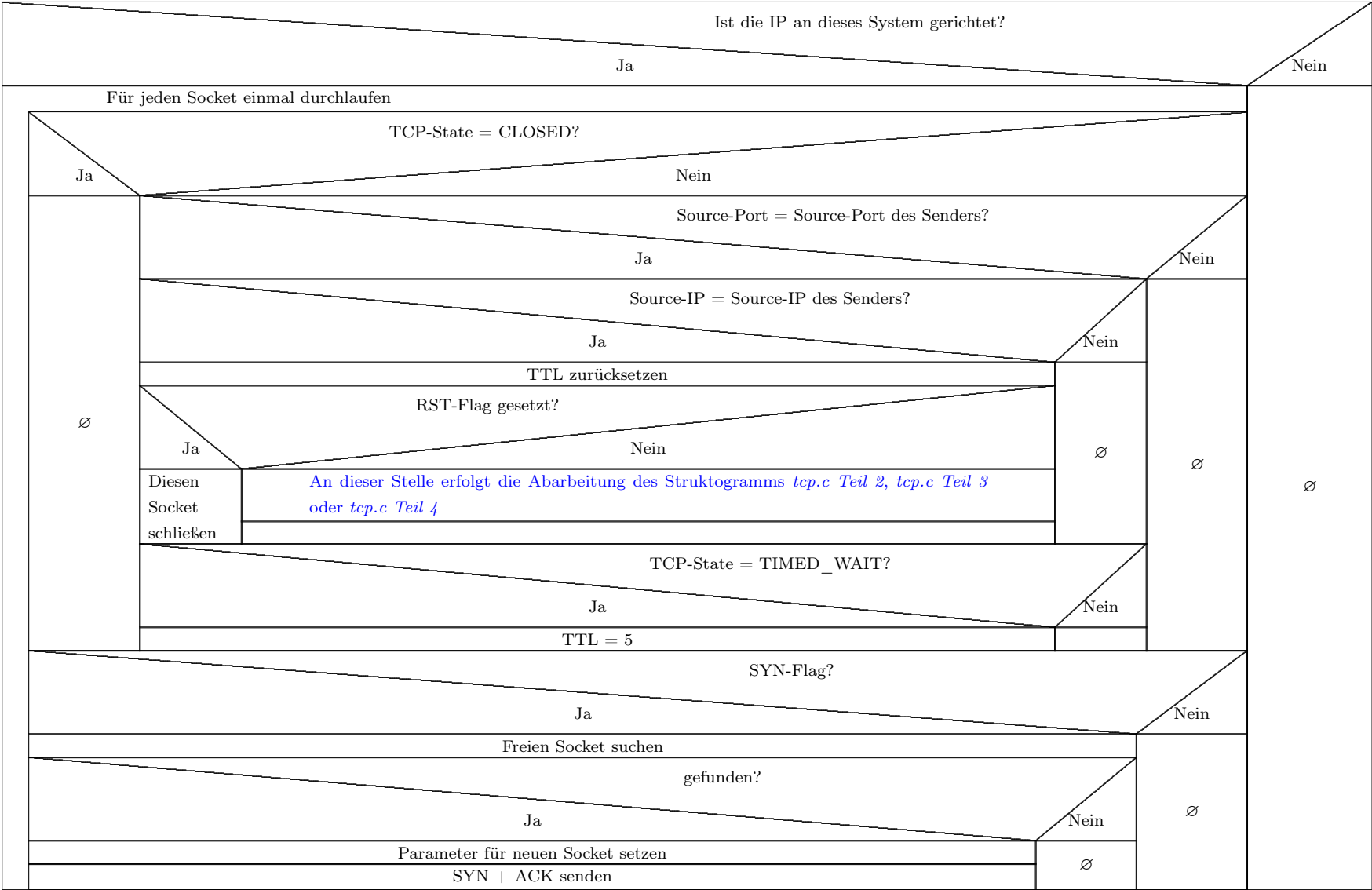
In diesem Teil des *Stacks* ist das *Transmission Control Protocol* realisiert. Sobald ein empfangenes Paket in der *main.c* an die Funktion `void tcp_packet_in(unsigned char *buffer, unsigned int len)` weitergereicht wird, wird dieses durch das Struktogramm «tcp.c» verarbeitet. Die einzelnen *TCP-Verbindungen* werden über das Array «tcp_sockets» verwaltet. Ein einzelner Eintrag dieses Arrays hat den folgenden Aufbau:

```
struct tcp_socket {
    unsigned int  source_port;
    unsigned int  dest_port;
    unsigned long source_ip;
    unsigned long seq;
    unsigned long ack;
    unsigned char state;
    unsigned char ttl;
    unsigned char misc_state; //used by application above tcp!
};

extern struct tcp_socket tcp_sockets[];
```

Zum Verwalten der *TTLs* der *TCP-Sockets* wird sekundlich die Funktion `void tcp_ttl_cleanup()` aufgerufen, welche die *TTL* jedes *Sockets* dekrementiert und überprüft, ob diese schon abgelaufen ist. In der *tcp.h* können durch die beiden Variablen *TCP_SOCKET_COUNT* und *TCP_TTL_TIMEOUT* die maximale Anzahl der *Sockets* und die maximale *TTL* einer Verbindung festgelegt werden.

tcp.c Teil 1



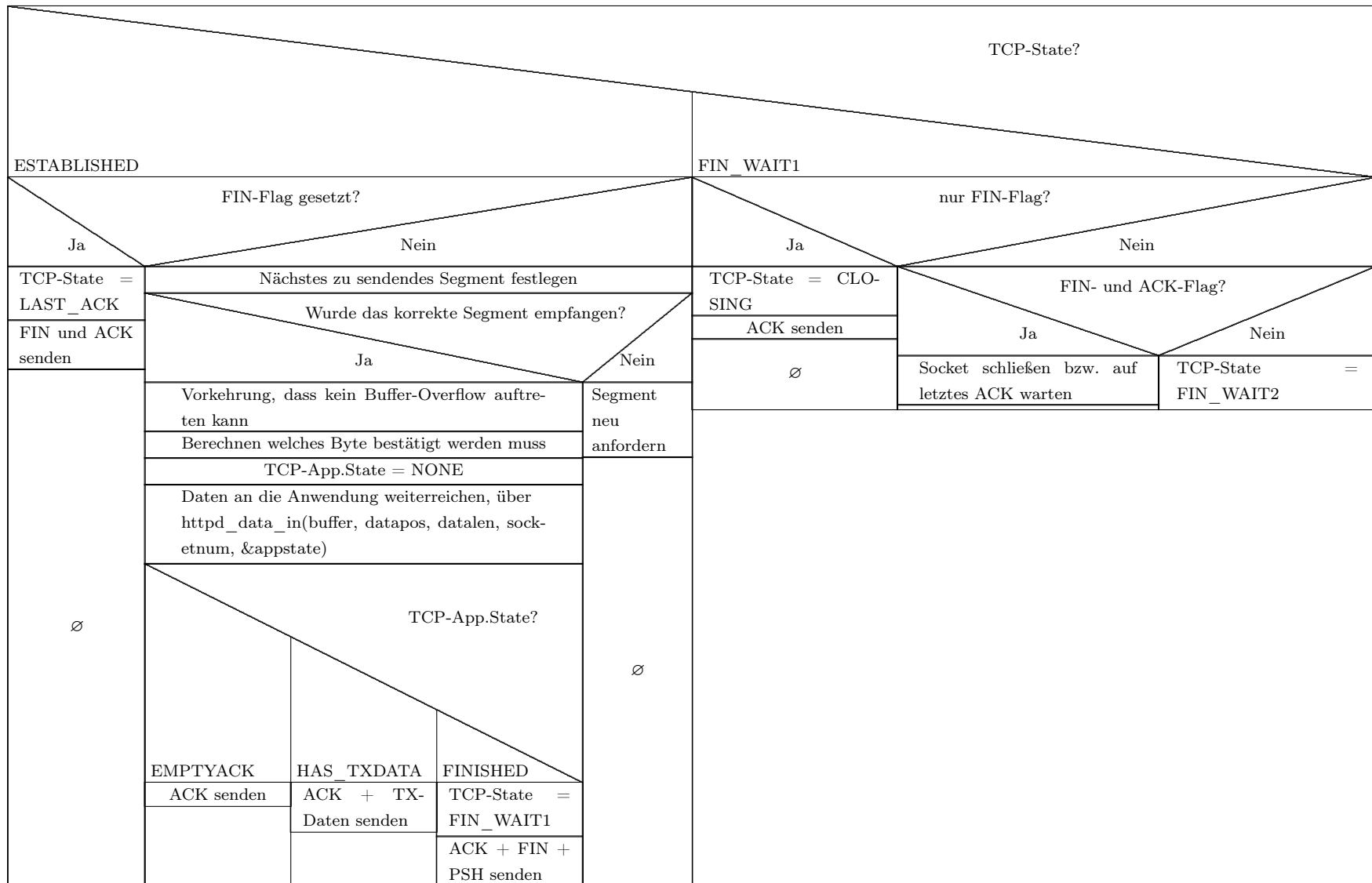
tcp.c Teil 2

TCP-State?			
CLOSE_WAIT	TIMED_WAIT	LAST_ACK	SYN_RECEIVED
TCP-State = LAST_ACK		Socket schließen	ACK-Flag gesetzt?
FIN senden			Ja
			Nein
			TCP-State = ESTABLISHED
			Socket schließen

tcp.c Teil 3

TCP-State?			
FIN_WAIT2		CLOSING	
FIN-Flag?		ACK-Flag?	
Ja		Ja	
Nein		Nein	
TCP-State = TIMED_WAIT		TCP-State = TIMED_WAIT	
ACK senden			
		Ø	

tcp.c Teil 4



3.3.6.3.1. Internet Protocol

3.3.6.3.1.1. Schnittstellen von IP Der *IP-Teil* des Netzwerkstacks besitzt zur Kommunikation die beiden Schnittstellen aus Abbildung 3.12. Die eine Schnittstelle ist die Funktion *void ip_generate_packet(unsigned char *buffer, uint32_t *dest_ip, unsigned char *dest_mac, unsigned int source_port, unsigned int dest_port, unsigned char ip_packettype, unsigned int data_length)*, durch welche der *IP-Teil* aufgerufen wird. Die andere Schnittstelle besteht aus direkten Schreib-/Lesezugriffen auf das *Array* des *Buffers*.

Die Funktion zur Generierung des *IP-Pakets* erhält als Parameter einen Zeiger auf den Anfang des *Buffers*, die *Ziel-IP-Adresse*, die *Ziel-MAC-Adresse*, den *Quell-* und *Ziel-Port* sowie den im Datenbereich verwendeten *Pakettyp* und die Länge der Nutzdaten.

3.3.6.3.1.2. Funktionalität Nach der Ausführung dieser Funktion enthält der *Buffer* einen *Ethernetframe* mit darin enthaltenem *IP-Paket* und den Nutzdaten.

3.3.6.3.1.3. Realisierung Die Datei *ip.h* enthält einige wichtige Angaben zum Aufbau eines *IP-Pakets*. Außerdem werden in dieser Datei alle *Netzwerkports* des Systems definiert. Die Datei *ip.c* enthält die Funktion namens *void ip_generate_packet(unsigned char *buffer, uint32_t *dest_ip, unsigned char *dest_mac, unsigned int source_port, unsigned int dest_port, unsigned char ip_packettype, unsigned int data_length)*, welche zur Erzeugung des *IP-Pakets* verwendet wird. Neben der Generierung des *IP-Pakets* erzeugt die Funktion zusätzlich den zugehörigen *Ethernetframe*, wie in Kapitel 3.3.2.3.1 beschrieben.

3.3.6.3.2. HTTP-Daemon

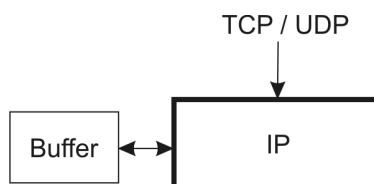


Abbildung 3.12.: Schnittstellen von IP

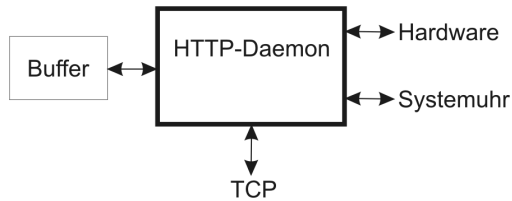


Abbildung 3.13.: Schnittstellen des HTTP-Daemon

3.3.6.3.2.1. Schnittstellen des HTTP-Daemon Der *HTTP-Daemon* besitzt Schnittstellen zu den folgenden Komponenten aus Abbildung 3.13:

- TCP

Der *Daemon* wird von *TCP* über die Funktion *unsigned int http_data_in(unsigned char *buffer, unsigned int datapos, unsigned int datalen, unsigned char socketnum, unsigned char *appstate)* aufgerufen. Dabei werden ein Zeiger auf den *Buffer*, ein Index auf den Anfang der übermittelten Daten, die Länge der Daten, die Socketnummer sowie der aktuelle Status der *TCP-Verbindung* übergeben.

- Hardware

Der Webserver verwendet zum Auslesen und Setzen von Portpins des *AVR-Mikrocontrollers* die folgenden drei Funktionen:

- *void port_set_portbit(unsigned char portchar, unsigned char pin, unsigned char val)*

Mit Hilfe dieser Funktion kann ein einzelnes Bit eines Ports gesetzt werden. Der Aufruf «*port_set_portbit('B', '3', 1);*» setzt beispielsweise Bit drei von Port B auf den Wert eins.

- *unsigned char port_get_portbit(unsigned char portchar, unsigned char pin)*

Diese Funktion ermöglicht das Auslesen des Ausgangsregisters eines Ports. Der Aufruf «*port_get_portbit('A', '3');*» gibt beispielsweise den Wert von Portpin A3 zurück.

- *unsigned char port_get_pinbit(unsigned char portchar, unsigned char pin)*

Durch Aufruf dieser Funktion wird das Eingangsregister eines Ports gelesen. Die Parameter sind hierbei die selben wie bei der Funktion zum Lesen des Ausgangsregisters.

- Buffer

Der Zugriff auf den *Buffer* geschieht über direkte *Array-Zugriffe*.

KAPITEL 3. SOFTWARE DES MIKROCONTROLLERPROGRAMMS

- Systemuhr

Der *HTTP-Daemon* aktualisiert die Systemzeit über einen Aufruf der Funktion *void clock_do(void)*. Kurz danach greift er über das *Array* «extern unsigned char clock[3]» lesend auf die Systemzeit zu.

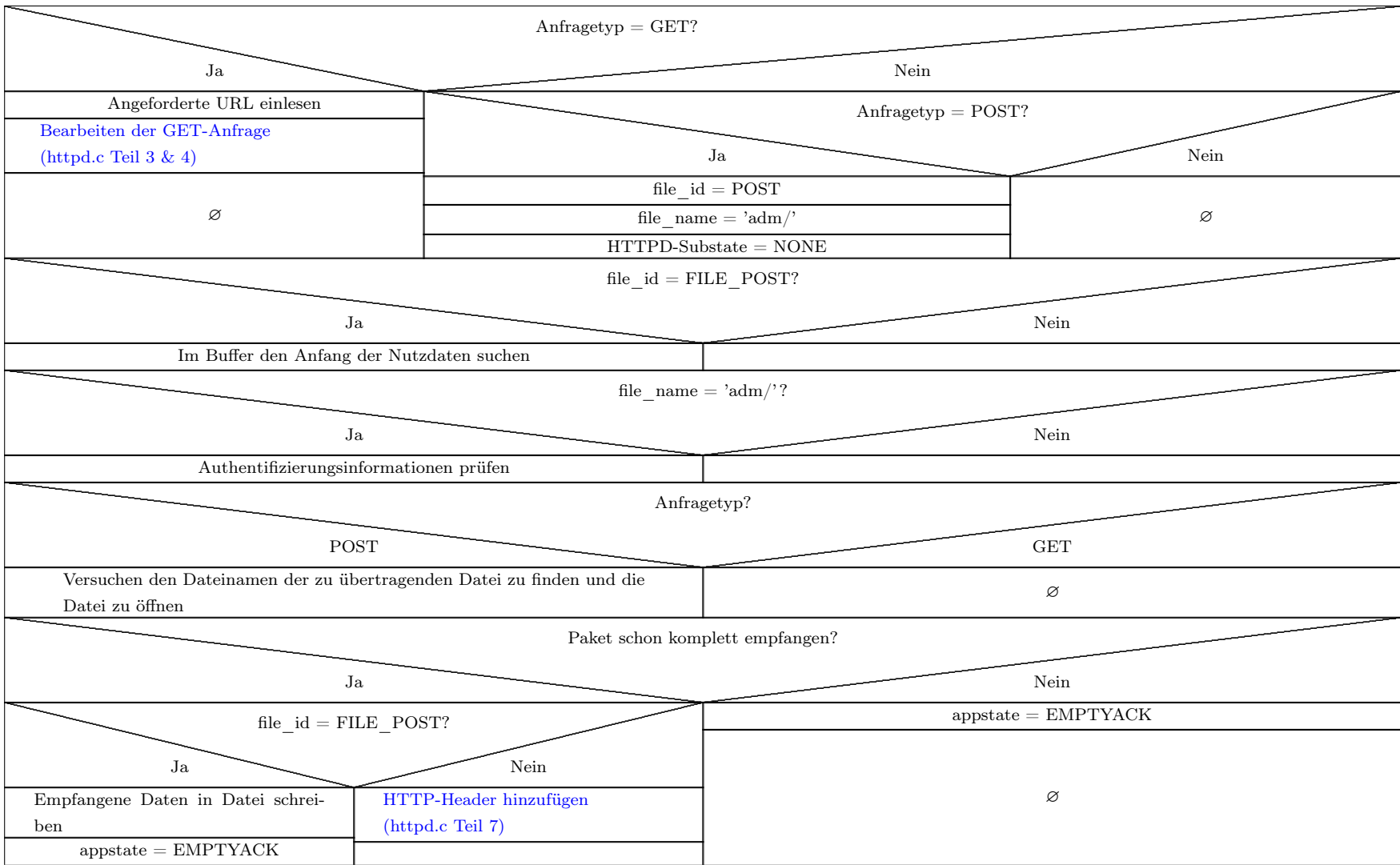
3.3.6.3.2.2. Funktionalität Der *HTTP-Daemon* unterstützt *GET*- sowie *POST*-Anfragen. Bei *POST*-Anfragen werden jedoch nur grundlegende Funktionen, wie beispielsweise *Authentifizierung*, unterstützt. Die anzuzeigenden Websites können im *Flash* in Form von *Website-Templates* abgelegt werden. Weiterhin werden Binärdateien unterstützt. Durch deren Verwendung können Bilder angezeigt oder Dateien zum Download angeboten werden.

3.3.6.3.2.3. Realisierung Die wichtigsten Funktionen des *HTTP-Daemon* befinden sich in den beiden Dateien *httpd.c* und *httpd.h*. Nachdem in der *main.c* entschieden wurde, dass es sich bei dem empfangenen Paket um ein *TCP-Paket* handelt, wird daraufhin die Funktion *void tcp_packet_in(unsigned char *buffer, unsigned int len)* aufgerufen. In dieser Funktion wird überprüft, an welchen Port die Anfrage gerichtet ist und falls es sich hierbei um den Port des *Hypertext Transfer Protocol Daemon (HTTPD)* handelt, wird dieser über die Funktion *unsigned int httpd_data_in(unsigned char *buffer, unsigned int datapos, unsigned int datalen, unsigned char socketnum, unsigned char *appstate)* aufgerufen. Der darauf folgende Ablauf des *HTTPD* ist in den Teilen des Struktogramms «httpd.c» dargestellt.

httpd.c Teil 1

Initialisierung				
HTTPD-State?				
IDLE	ACTIVE	FINISHED	POST	sonst
Bearbeiten der Anfrage (httpd.c Teil 2)	appstate = HAS_TXDATA	HTTPD-State = IDLE	Datei komplett empfangen? Ja Nein	appstate = EMPTYACK
	Generierung der HTTP-Antwort (httpd.c Teil 5 & 6)	appstate = FINISHED		
			Antwort erzeugen	
			file_id = UP_DONE oder UP_ERR	
			appstate = HAS_TXDATA	
			nächsten Teil speichern	
			appstate = EMPTYACK	
			∅	

httpd.c Teil 2



httpd.c Teil 3

URL?			
index.*	site/temp	site/io	logo.*
file_id = INDEX	file_id = SITE_TEMP	file_id = SITE_IO	file_id = LOGO
file_ext[0] = 'h'			

httpd.c Teil 4

						URL?	
set/				get/		sonst	
file_name[7] = '1'?				file_name[7] = 'x'?		file_id = IDLE	
Ja		Nein		Ja		Nein	
file_name = D[4,5,6]?		file_name = D[4,5,6]?		Portbit gesetzt?		Ø	
Ja		Ja		Ja			
Nein		Nein		Nein			
file_id = SET_IO_0_PIC		file_id = SET_IO_1_PIC		file_id = SET_IO_1_PIC			
Portbit setzen		Portbit löschen					

httpd.c Teil 5

file_id?		
INDEX	SITE_TEMP	SITE_IO
INDEX_HTML zum Buffer hinzufügen	INDEX_HTML zum Buffer hinzufügen	INDEX_HTML zum Buffer hinzufügen
START_HTML zum Buffer hinzufügen	TEMP_HTML zum Buffer hinzufügen	IO_HTML zum Buffer hinzufügen

httpd.c Teil 6

file_id?			
SET_IO_0_PIC	SET_IO_1_PIC	LOGO	sonst
BUTTON_RED zum Buffer hinzufügen	BUTTON_GREEN zum Buffer hinzufügen	LOGO_GIF zum Buffer hinzufügen	FILE_NOT_FOUND zum Buffer hinzufügen

httpd.c Teil 7

HTTPD_RESPONSE_OK zum Buffer hinzufügen				
file_ext[0]?				
'j'	'p'	'b'	'g'	sonst
HTTPD_CTYPE_JPG hinzufügen	HTTPD_CTYPE_PNG hinzufügen	HTTPD_CTYPE_BMP hinzufügen	HTTPD_CTYPE_GIF hinzufügen	HTTPD_CTYPE_HTML hinzufügen
\r\n\r\n hinzufügen				
appstate = HAS_TXDATA				
HTTPD-State = ACTIVE				

KAPITEL 3. SOFTWARE DES MIKROCONTROLLERPROGRAMMS

Eine wichtige Funktion des *Daemons* ist die *unsigned int httpd_add_progmem_data(PGM_P pointer, PGM_P include, unsigned char *buffer, unsigned int pos, unsigned int offset, unsigned int len)*. In dieser Funktion ist das Parsen der Daten und das Hinzufügen der Daten zum *Buffer* realisiert. Dadurch ist es möglich, bestimmte Teile einer Ressource zur Laufzeit durch andere Daten zu ersetzen beziehungsweise weitere Ressourcen einzubinden.

In *httpd_data.c* und *httpd_data.h* werden die Dateien definiert, die der Webserver anzeigen kann, also beispielsweise HTML-Dateien oder Bilddateien. Jede Ressource ist hierbei in einem eigenen Array definiert, welches für eine HTML-Datei zum Beispiel wie folgt aussehen kann:

```
PROGMEM unsigned char HTTPD_FILE_SITE_START_HTML[] = {
"<CENTER>"
"<BR><BR>"
"<H1>AVR-Webserver</H1><BR>v1.1<br>(based on avrETH1
<a href=\"http://avr.auctionant.de\">http://avr.auctionant.de</a>)"
"<BR><BR>Uptime: $$CK"
"</CENTER>"
};
```

Eine Grafik wird dagegen im Binärformat angegeben und kann folgende Form aufweisen:

```
PROGMEM unsigned char HTTPD_BUTTON_GREEN[] = {
0x47, 0x49, 0x46, 0x38, 0x37, 0x61, 0x0F, 0x00, 0x0F, 0x00, 0xC2, 0x08,
0x00, 0xB5, 0x05, 0x2B, 0xCC, 0x01, 0x32, 0xA7, 0x1B, 0x38, 0xB1, 0x47,
0x5C, 0xBB, 0x6F, 0x7E, 0xD5, 0x67, 0x7E, 0xE6, 0x9A, 0xAB, 0xFC, 0xF6,
0xF8, 0x2C, 0x00, 0x00, 0x00, 0x00, 0x0F, 0x00, 0x0F, 0x00, 0x00, 0x03,
0x50, 0x78, 0xAA, 0x56, 0x6E, 0x2B, 0x1E, 0x13, 0x88, 0x25, 0x02, 0x14,
0x59, 0xC4, 0xBD, 0x86, 0xB0, 0x31, 0xC0, 0x67, 0x8A, 0x0A, 0x20, 0x78,
0x2C, 0xB6, 0x66, 0x13, 0x20, 0xAB, 0xAB, 0x9C, 0xC9, 0xCE, 0xAC, 0xEF,
0xB4, 0x1E, 0x00, 0x3F, 0xA0, 0xEC, 0x57, 0x08, 0x18, 0x8D, 0xC0, 0xE3,
0xB1, 0xA4, 0x6C, 0x2A, 0x09, 0x87, 0xE6, 0xC0, 0x19, 0x50, 0x10, 0x02,
0x83, 0x2C, 0x36, 0xCB, 0x15, 0x2C, 0x04, 0xDC, 0x70, 0xD6, 0x1B, 0xB9,
0x8A, 0x03, 0x64, 0xC9, 0x01, 0xA3, 0x81, 0x46, 0x12, 0x00, 0x3B
};
```

KAPITEL 3. SOFTWARE DES MIKROCONTROLLERPROGRAMMS

Das Erstellen einer solchen binären Ressource ist relativ einfach. Hierzu muss die gewünschte Datei mit einem beliebigen Hex-Editor geöffnet und der angezeigte Inhalt in das jeweilige Array der *httpd_data.c* kopiert werden.

Zum Erstellen eines *Website-Templates* genügt es, wenn der gewünschte Grundkörper der HTML-Datei erstellt und an den Stellen, an welchen der externe Inhalt eingebunden werden soll, ein String der Form «*\$\$xx*» hinzugefügt wird. Dieser String wird beim Verarbeiten der Seite vom Parser durch den gewünschten Inhalt ersetzt. Das «*xx*» definiert dabei, durch was der String ersetzt werden soll. So fügt «*\$\$CK*» beispielsweise die aktuelle Systemzeit ein.

Zur Verarbeitung der Authentifizierung bei *HTTP-POST-Requests* wird die Funktion *void base64_decode(unsigned char *buf, unsigned char len)* aus den Dateien *base64.c* und *base64.h* verwendet, welche den empfangenen *BASE64-Datenstrom*, wie in RFC3548 [65], RFC4648 [66] und dem Artikel «Base64» [2] beschrieben, dekodiert.

In den Dateien *string.c* und *string.h* des *Daemons* werden einige kleine Funktionen definiert, welche zum Durchsuchen und Kopieren von Strings verwendet werden. Das Ziel ist hierbei sehr kleine und effektive Funktionen zur Verfügung zu stellen, um den Mikrocontroller nicht unnötig zu belasten.

Der Zugriff auf die Portpins des Mikrocontrollers ist über die Funktionen in den Dateien *port.c* und *port.h* realisiert.

Zum Anzeigen des Analogwerts an *Port A0* wird die *Analog-to-Digital converter function library* aus der *Procyon AVRlib* von *Pascal Stang* verwendet. Diese Library ermöglicht die Verwendung des *Analog-Digital-Converter (ADC)* des *AVR-Mikrocontrollers*. Die *A/D-Wandlung* wird über den Aufruf der Funktion *unsigned char a2dConvert8bit(unsigned char ch)* gestartet.

3.3.7. Systemuhr

3.3.7.1. Schnittstellen der Systemuhr

Die Systemuhr besitzt Schnittstellen zu den Komponenten aus Abbildung 3.14, welche wie folgt definiert sind:

- *void clock_do(void)*

Durch Aufruf dieser Funktion wird die Systemzeit in dem *Array* «extern unsigned char clock[3]» aktualisiert.

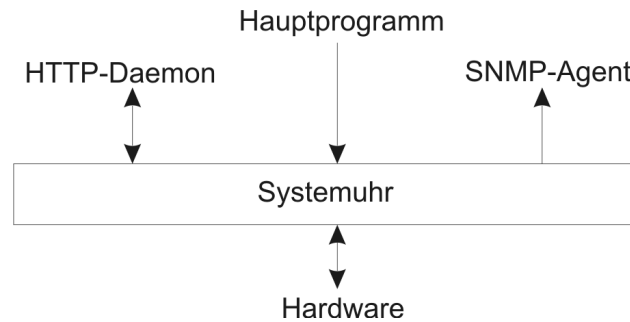


Abbildung 3.14.: Schnittstellen der Systemuhr

- *extern unsigned char clock[3]*

Über dieses *Array* kann die aktuelle Systemzeit ausgelesen werden. «clock[0]» enthält hierbei die Stunden, «clock[1]» die Minuten und «clock[0]» die Sekunden.

- *extern unsigned long int Timestamp;*

Der *Timestamp* steht für die *SNMP-Funktionalität* des Mikrocontrollerprogramms zur Verfügung und kann über die Variable «extern unsigned long int Timestamp» ausgelesen werden.

- Registerzugriffe

Für das Ansprechen der Hardware des Systems werden direkte Zugriffe auf die Portregister verwendet.

3.3.7.2. Funktionalität

Die Systemuhr stellt die seit dem Systemstart vergangene Zeit im Format «HH:MM:SS» dar. Zusätzlich bietet sie einen *Timestamp* an, der alle zehn Millisekunden erhöht wird.

3.3.7.3. Realisierung

Die Systemuhr ist in den Dateien *clock.c* und *clock.h* implementiert. Durch einen Timer-Interrupt, dessen Zählerstand über die Taktfrequenzangabe im *Makefile* (Kapitel E) berechnet wird, ist eine relativ genaue Systemzeit gewährleistet.

Das Aktualisieren der Systemzeit geschieht jede Sekunde durch einen Aufruf der Interruptroutine *SIGNAL (SIG_OVERFLOW1)*, welche die Variable *clock_intcount* um eins inkrementiert und das *clock_new_flag* setzt. Der Anwender kann anschließend an jeder Stelle im Mikrocontrollerprogramm

KAPITEL 3. SOFTWARE DES MIKROCONTROLLERPROGRAMMS

die Funktion *void clock_do(void)* aufrufen, um die aktuelle Systemzeit zu bestimmen. Dieser Aufruf wird beispielsweise in jeder Runde des Hauptprogramms einmal ausgeführt.

Zusätzlich zur Systemuhr befindet sich in der *clock.c* noch eine Erweiterung für die *SNMP-Funktionalität*. Dabei handelt es sich um das Inkrementieren des *SNMP-Timestamps* alle zehn Millisekunden. Die Zeit kann über die Variable *TIMER_0_CLOCK* in der Datei *clock.h* angepasst werden.

3.3.8. Buffer

Der *Buffer* für den Netzwerkstack hat die Aufgabe, empfangene Daten zwischenspeichern und die zu sendenden Daten aufzunehmen. Er bietet Zugriff auf die Daten des aktuellen Netzwerkpakets, um mit diesen zu arbeiten. Im Mikrocontrollerprogramm können die Daten im *Buffer* jederzeit ausgelesen und verändert werden.

Wird ein neues Netzwerkpaket über einen der *Ethernetcontroller* empfangen, wird dieses komplett in den *Buffer* übertragen. Beim Senden eines Pakets werden die Daten im *Buffer* an den entsprechenden *Ethernetcontroller* übergeben.

Der *Buffer* ist durch das Array «*unsigned char buffer[NIC_BUFFERSIZE];*» realisiert und wird in der *main.c* definiert. Die mit «*NIC_BUFFERSIZE*» angegebene Größe des *Buffers* wird in der Datei *mynic.h* festgelegt. Die Größe muss hierbei so gewählt werden, dass ein zu empfangendes Netzwerkpaket komplett in den *Buffer* passt.

4. Test und Inbetriebnahme

In diesem Kapitel werden der Hardwareaufbau und die verwendeten Softwaretools zum Analysieren und Testen der Netzwerkprotokolle vorgestellt. Grundlegende Informationen zu den verwendeten Hardwaretools sowie zur Installation der Software werden in Anhang C näher erläutert.

An dieser Stelle soll auch erwähnt werden, dass es unter Umständen vorkommt, dass sich ein Netzwerkgerät wie beispielsweise ein *Hub*, *Router* oder *Switch* aufhängt, sobald ein ungültiges Netzwerkpaket übertragen wird. Im Zeitraum dieser Diplomarbeit hat ein verwendeter *8-Port-Switch* zweimal seinen Dienst verweigert, nachdem er ein ungültiges Netzwerkpaket erhalten hatte. Erst nach einer kurzen Unterbrechung der Stromversorgung nahm das Gerät seinen Dienst wieder auf.

4.1. Hardware

4.1.1. Verbinden des STK-LAN mit dem STK500 und Konfiguration der Boards

Zum Testen der Hard- und Software müssen das *STK-LAN* und das *STK500* zuerst miteinander verbunden werden. Hierzu werden zuerst alle Jumpereinstellungen des *STK500* auf die aufgedruckten Werte des *STK-LAN* gebracht. Der gewünschte *AVR-Mikrocontroller* muss vor dem Aufstecken des *STK-LAN* in den Sockel *SCKT3100A3* des *STK500* gesteckt werden. Anschließend wird die Erweiterungsplatine auf das *STK500* aufgesteckt. Hierbei ist darauf zu achten, dass die Platine in der richtigen Position auf das *STK500* gesteckt wird. Dies ist gewährleistet, wenn die Stiftleiste *EXPAND0* des *STK-LAN* mit der Buchsenleiste *EXPAND0* des *STK500* und die beiden Leisten *EXPAND1* miteinander verbunden werden. In dieser Position zeigen die Stromversorgungsbuchsen der beiden Boards in die gleiche Richtung.

Da die beiden Boards über zwei Steckverbinder mit jeweils 40 Pins verbunden werden, erfordert das Aufstecken etwas Kraftaufwand. Um die beiden Boards hierbei nicht zu beschädigen, wird

KAPITEL 4. TEST UND INBETRIEBNAHME

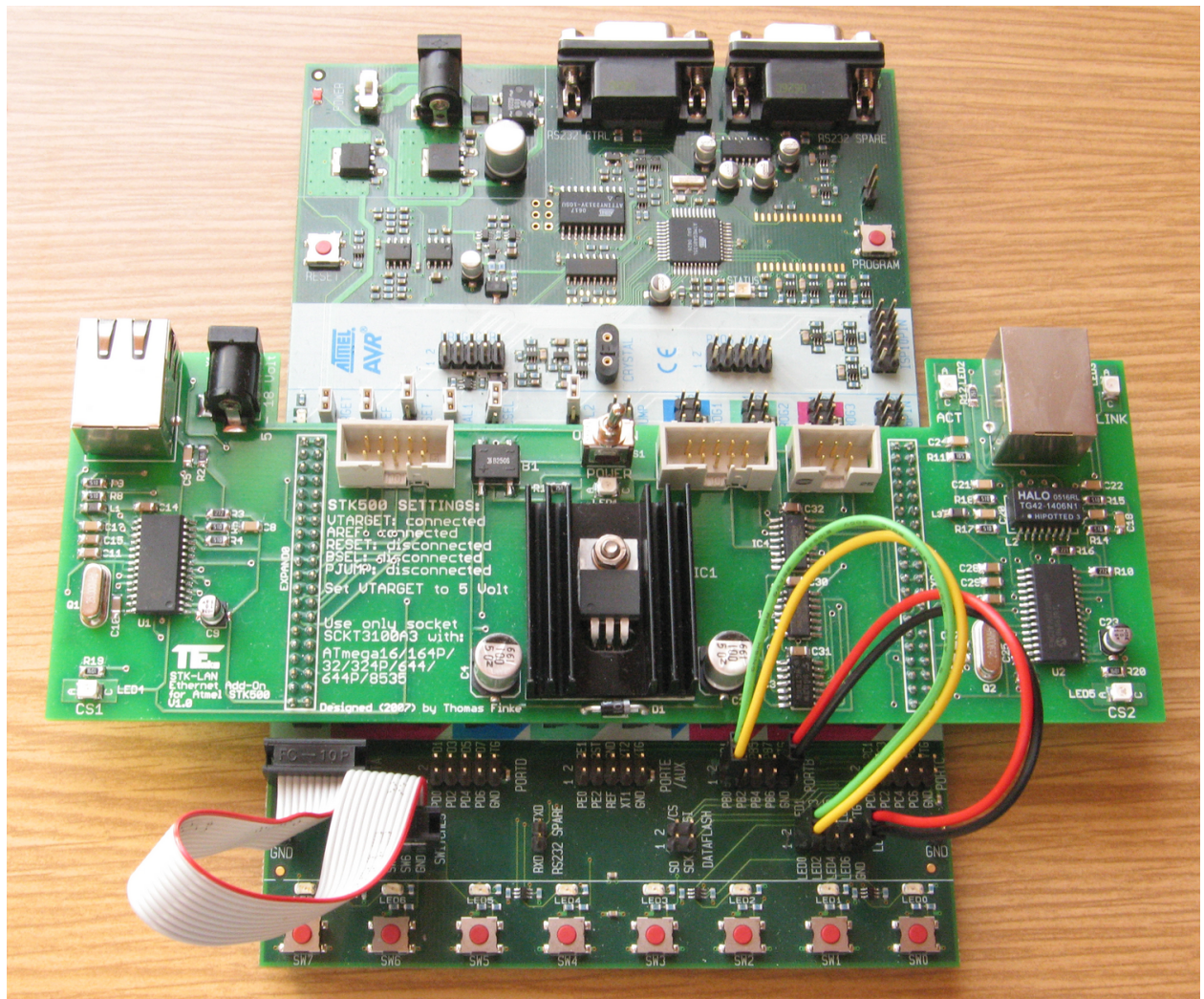


Abbildung 4.1.: STK500 mit aufgestecktem STK-LAN

KAPITEL 4. TEST UND INBETRIEBNAHME

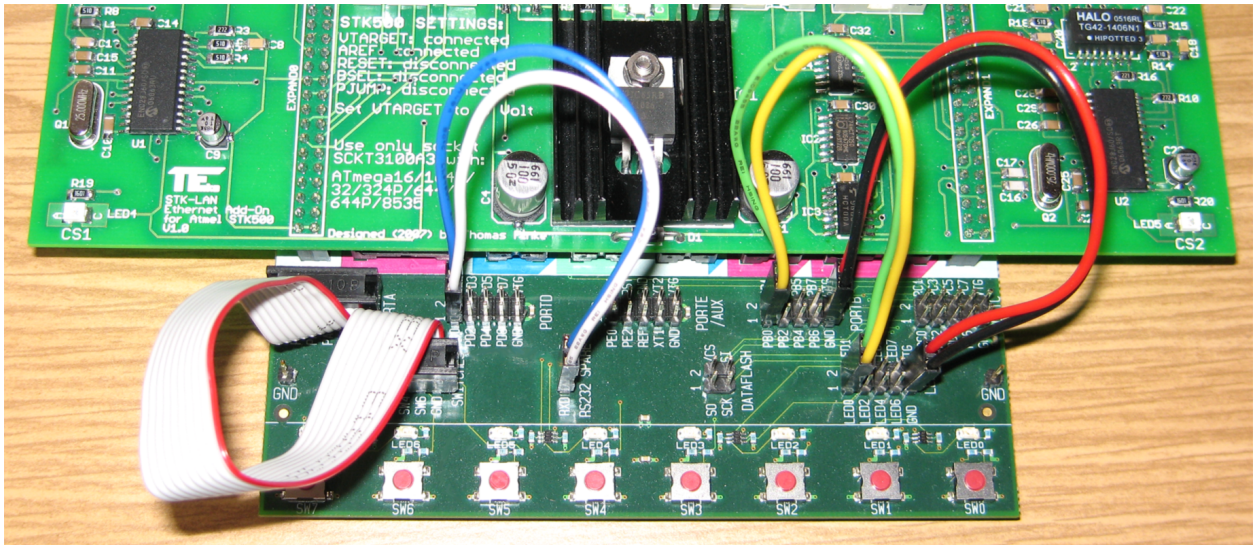


Abbildung 4.2.: Benötigte Kabelverbindungen auf dem STK500

empfohlen, die Boards zuerst passend aufeinander zu legen und danach die beiden 40-poligen Steckverbinder nacheinander zusammenzustecken. Abbildung 4.1 zeigt die beiden zusammengesteckten Platinen. Anschließend muss die Stiftleiste «PORTA» des *STK500* über das 10-adrige Flachbandkabel zu der Stiftleiste «SWITCHES» geführt werden. Zusätzlich sind die beiden Pins «PB0» und «PB1» der Stiftleiste «PORTB» mit den Pins «LED0» und «LED1» der Stiftleiste «LEDS» zu verbinden. Zwischen diesen beiden Stiftleisten sollten zusätzlich die beiden «GND»-Pins sowie die beiden «VTG»-Pins miteinander verbunden werden¹. Für diese Verbindungen eignen sich die zweiadrigen Adapterkabel des *STK500* sehr gut. Um die «RS232 SPARE»-Schnittstelle des *STK500* verwenden zu können, müssen die beiden Pins *Received Data (RXD)* und *Transmitted Data (TXD)* der Stiftleiste «RS232 SPARE» mit den beiden Pins «PD0» und «PD1» der Stiftleiste «PORTD» zusammengeführt werden. Die erforderlichen Kabelverbindungen sind aus Abbildung 4.2 nochmals ersichtlich.

Zusätzlich zu den bisher getätigten Verbindungen müssen die beiden Platinen noch an die Versorgungsspannung im Bereich von 8 bis 12 Volt angeschlossen werden. Hierzu sind die entsprechenden Anschlussleitungen vom Netzteil zu den beiden Hohlbuchsen auf den Platinen zu führen (Abbildung 4.3). Zum Programmieren und Debuggen kann noch ein geeignetes Programmiergerät, wie beispielsweise das *Atmel JTAGICE mkII*, an die dafür vorgesehenen Wannenstecker *JTAG*, *ISP10PIN* oder *ISP6PIN* angeschlossen werden. Zum Schluss wird noch ein Netzkabel mit

¹Laut dem Schaltplan [1] des *STK500* ist diese Verbindung der Versorgungspins nicht nötig, da diese bereits auf dem *STK500* fest miteinander verbunden sind. Um Problemen bei einer eventuell neueren Version des *STK500* vorzubeugen, wird dennoch empfohlen, diese Verbindung zu realisieren.

KAPITEL 4. TEST UND INBETRIEBNAHME

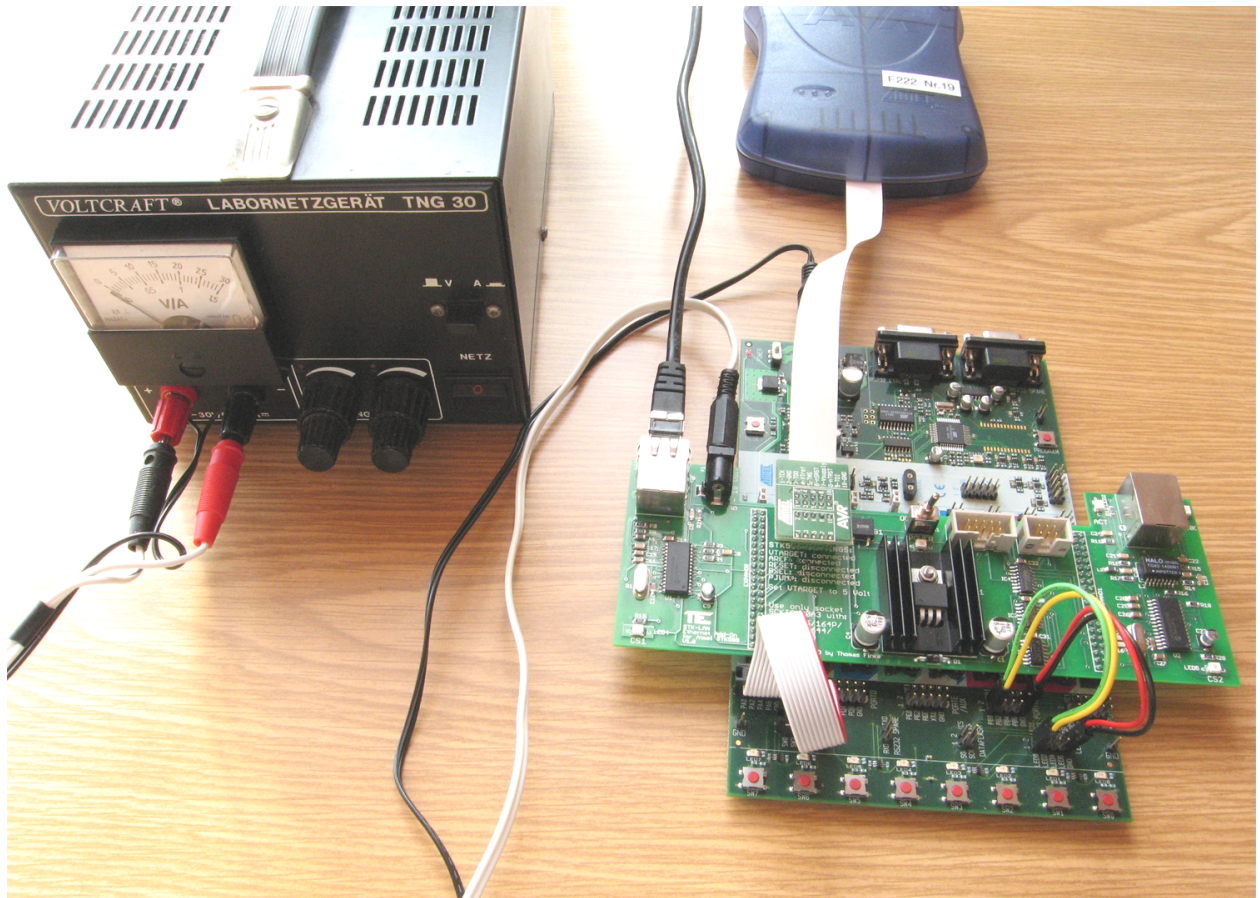


Abbildung 4.3.: Kompletter Hardwareaufbau

dem in der Mikrocontroller-Software verwendeten Ethernetport verbunden. Soll zusätzlich über die serielle Schnittstelle des *STK500* mit dem PC kommuniziert werden, ist ein serielles Kabel zwischen der neunpoligen Sub-D-Buchse des *STK500* mit der Bezeichnung *RS232 SPARE* und einer seriellen Schnittstelle des PCs nötig. Das hierzu zu verwendende Kabel sollte eine 1:1-Verbindung besitzen, also kein Nullmodemkabel sein.

Wird das *STK-LAN* über einen *Hub*, *Switch* oder *Router* mit dem Netzwerk verbunden, wird hierzu ein *Patchkabel* benötigt. Soll das *STK-LAN* hingegen ohne zwischengeschaltetes Netzwerkgerät an einen PC angeschlossen werden, muss ein sogenanntes *Crossover-Kabel* verwendet werden.

Da die Ethernetcontroller auf dem *STK-LAN* nur eine Übertragungsgeschwindigkeit von *10 MBit/s* unterstützen, müssen alle verwendeten *Hubs*, *Switches*, *Router* und PCs einen der Geschwindigkeitsmodi *10 MBit/s*, *10/100 MBit/s* oder *10/100/1000 MBit/s* unterstützen. Andernfalls ist eine Kommunikation zwischen *STK-LAN* und PC nicht möglich.

4.1.2. Einschalten der Boards und Überprüfen der LEDs

Nachdem der komplette Hardwareaufbau angeschlossen ist, können das *STK500* und das *STK-LAN* über die beiden Schalter mit der Bezeichnung «Power» eingeschaltet werden. Anschließend leuchten auf den beiden Boards die *LEDs* mit den Bezeichnungen «POWER». Direkt nach dem Einschalten blinkt die *STATUS-LED* des *STK500* einige Male und zeigt hierdurch den Initialisierungsvorgang an. Nach der Initialisierung geht diese *LED* in ein grünes Dauerleuchten über.

Auf dem *STK-LAN* leuchtet beziehungsweise blinkt zu diesem Zeitpunkt mindestens eine der beiden *Chipselect-LEDs* «CS1» und «CS2». Diese beiden *LEDs* visualisieren einen Zugriff des *AVR-Mikrocontrollers* auf den jeweiligen *Ethernetcontroller*.

Ist am *STK-LAN* ein Netzkabel angeschlossen², so wird dies durch die *LINK-LED* an der jeweiligen *8P8C-Modularbuchse* angezeigt. Werden zusätzlich Daten über dieses Netzkabel übertragen, so blinkt zusätzlich die *ACT-LED* des Anschlusses.

An dieser Stelle angekommen, sind die beiden Boards für die Anwendung bereit.

4.1.3. Abnehmen des STK-LAN vom STK500

Wird der Hardwareaufbau nicht mehr benötigt, können die beiden Boards wieder voneinander getrennt werden. Da die beiden Platinen jedoch durch insgesamt 80 Pins der Stiftleisten sehr fest miteinander verbunden sind, lassen sich die Platinen nicht auf einfache Weise trennen. Durch Versuche, die beiden Platinen mit einem Schraubendreher auseinander zu hebeln, könnten die Platinen beschädigt werden.

Um die beiden Boards schonend voneinander zu trennen, wurde im Rahmen dieser Diplomarbeit zusätzlich eine Aushebelvorrichtung (Abbildung 4.4) entwickelt. Diese kann, wie in Abbildung 4.5 gezeigt, angesetzt werden und anschließend können die beiden Platinen durch Betätigung der Zange auseinandergedrückt werden. Dieser Vorgang muss für die beiden Stiftleisten *EXPAND0* und *EXPAND1* der Platinen getrennt erfolgen.

²Am anderen Ende des Kabels muss sich allerdings auch ein Netzwerkteilnehmer befinden.

KAPITEL 4. TEST UND INBETRIEBNAHME

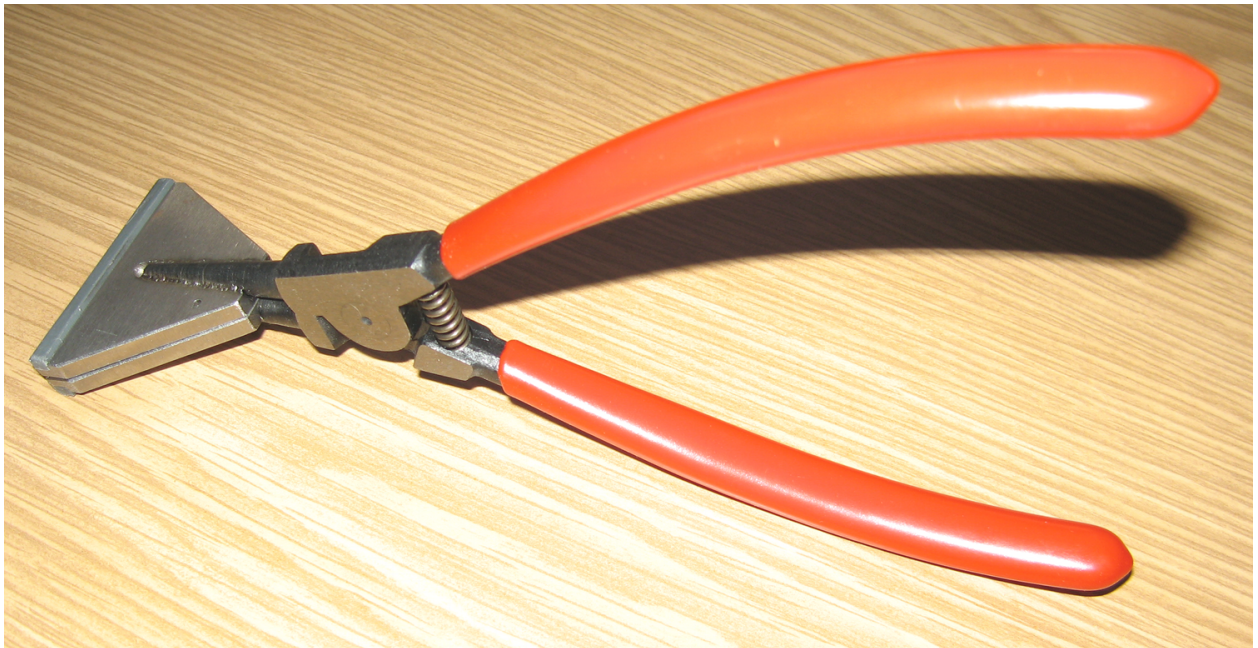


Abbildung 4.4.: Aushebelvorrichtung für das STK-LAN

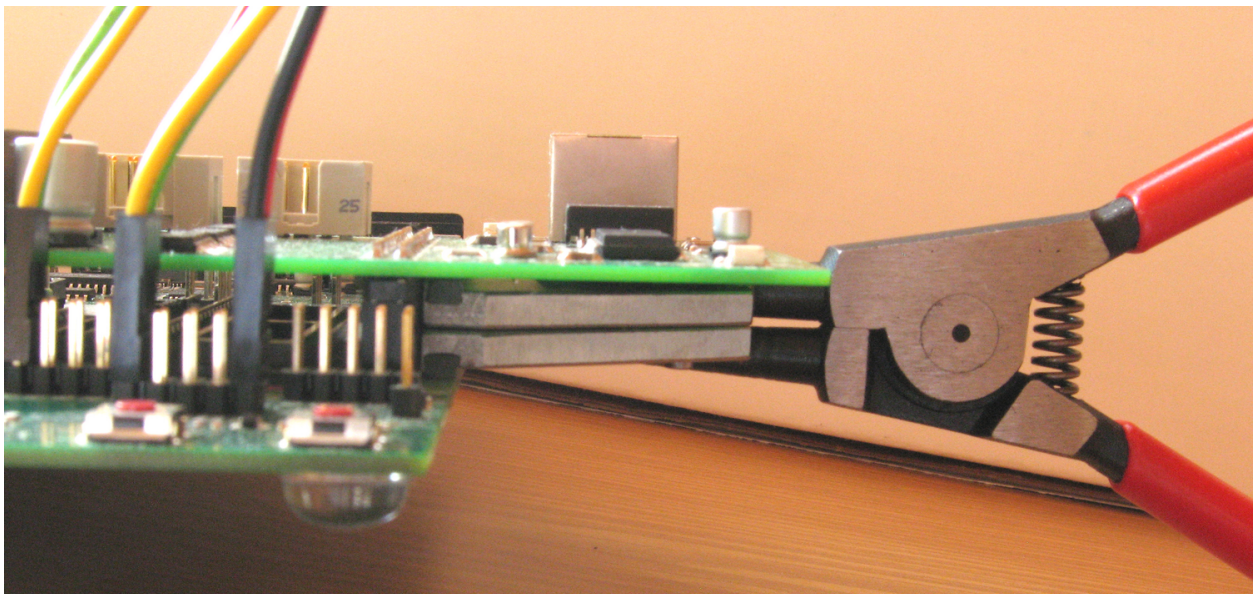


Abbildung 4.5.: Abnehmen des *STK-LAN*

4.2. Software

4.2.1. Programmieren des Mikrocontrollers

4.2.1.1. Modifizieren und Kompilieren des Programmcodes

Im Folgenden wird beschrieben, wie ein existierendes *WinAVR-Projekt* geöffnet und bearbeitet werden kann. Auf die Beschreibung zur Erstellung eines komplett neuen Projekts wird verzichtet, da es einfacher ist, ein neues Projekt als Modifikation eines alten Projekts anzulegen. Meist wird von den Studierenden im Rahmen des Labors nur auf das fertige Projekt zurückgegriffen werden, welches dann an die eigenen Anforderungen angepasst wird.

Nach der in Anhang C.2.1 beschriebenen Installation des Softwarepakets ist im Windows-Startmenü unter der Programmgruppe «WinAVR» ein Eintrag namens «Programmers Notepad [WinAVR]» zu finden. Hierbei handelt es sich um den Editor, in welchem der C-Code geschrieben wird. Nach dem Start kann über das Menü «File» und den Eintrag «Open Project(s)...» der Dialog zur Auswahl des gewünschten Projektfiles, wie in Abbildung 4.6 zu sehen, geöffnet werden. In dem erscheinenden Fenster wird die Projektdatei «avrETH.pnproj» ausgewählt und mit einem Klick auf «Öffnen» geladen.

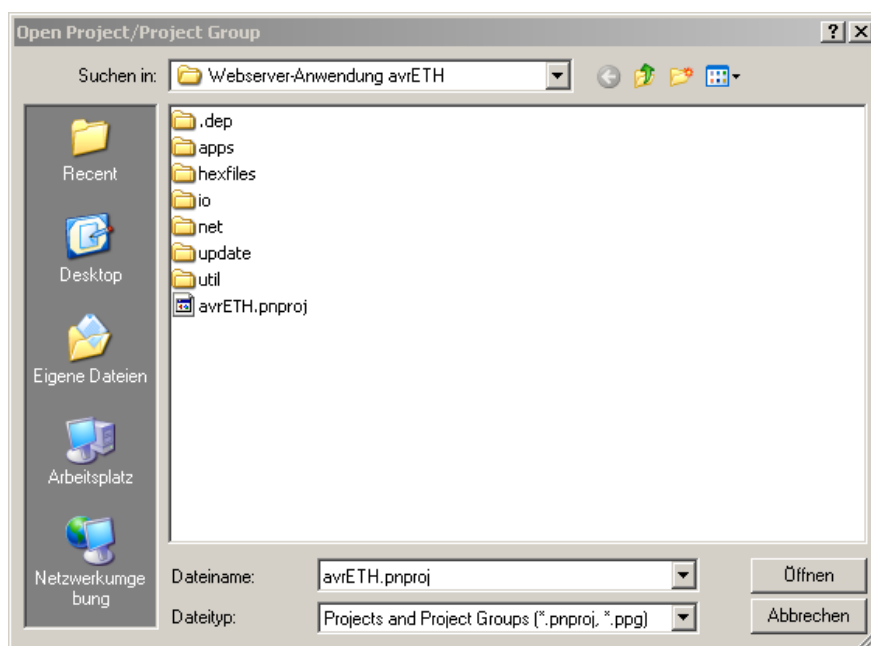


Abbildung 4.6.: Öffnen eines WinAVR-Projekts

KAPITEL 4. TEST UND INBETRIEBNAHME

Nach dem Öffnen des Projekts werden in der Projektübersicht «Projects» auf der linken Seite alle Dateien des aktuellen Projekts angezeigt. Mit einem Doppelklick können diese im rechten Teil des Fensters geöffnet werden. Dort können die Dateien schließlich modifiziert und über das Menü «File» gespeichert werden. Zum Kompilieren des Projekts muss in dem rechten Teil des Fensters die Datei «main.c» geladen und ausgewählt sein. Anschließend kann der Kompiliervorgang über den Menüpunkt «[WinAVR] Make All» im Menü «Tools» gestartet werden. Wenn der Kompiliervorgang erfolgreich abgeschlossen wurde, wird dies über die zwei Meldungen «Errors: none» und «Process Exit Code: 0» im unteren *Output-Fenster* angezeigt (Abbildung 4.7).

KAPITEL 4. TEST UND INBETRIEBNAHME

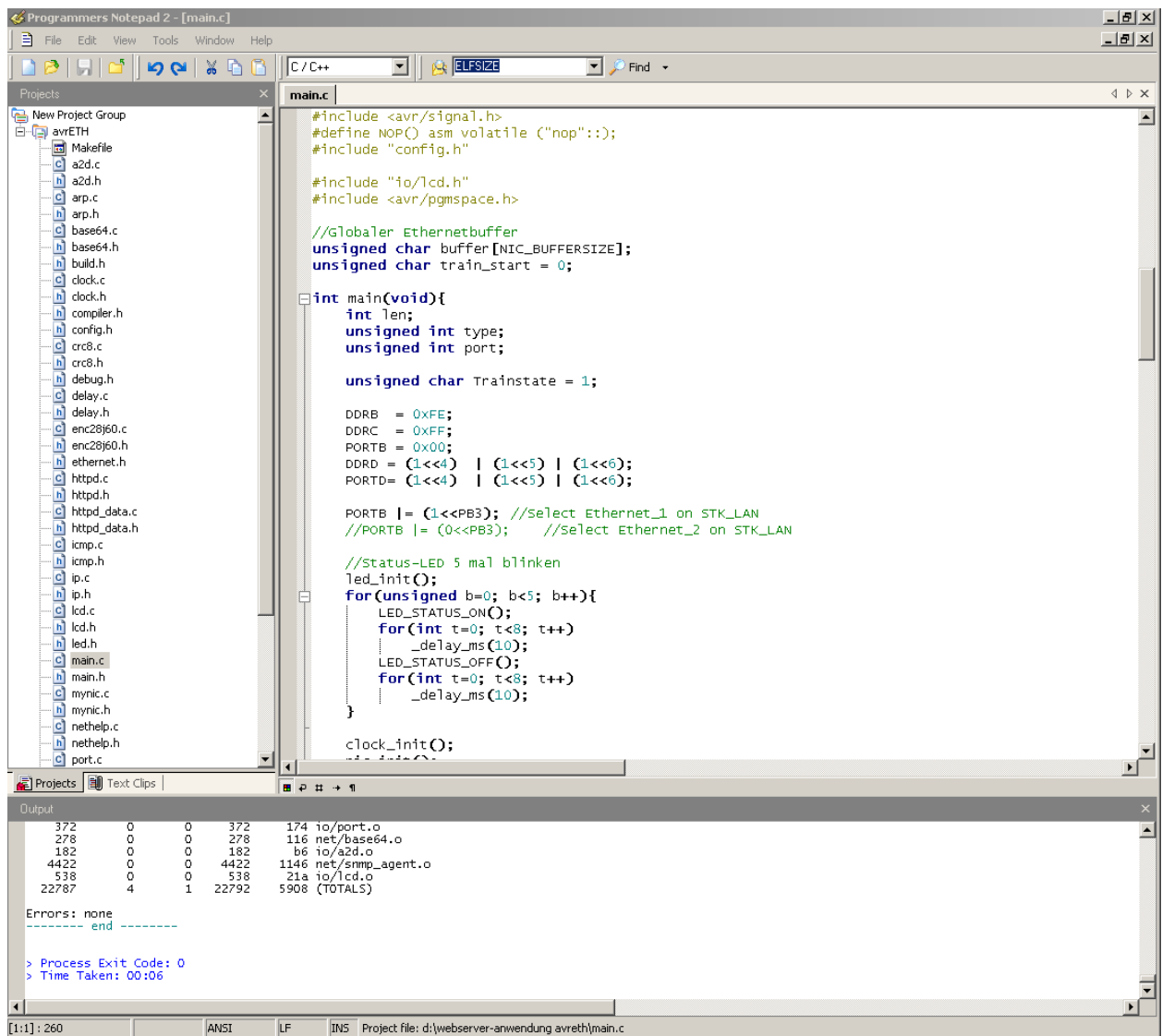


Abbildung 4.7.: Meldungen nach einem erfolgreichen Kompiliervorgang

KAPITEL 4. TEST UND INBETRIEBNAHME

4.2.1.2. Übertragen und Debuggen des Programms

Nachdem das *Atmel AVR Studio* laut Anhang C.2.2 installiert ist, kann es über den Eintrag «Start - Programme - Atmel AVR Tools - AVR Studio 4» im *Windows-Startmenü* gestartet werden. Direkt nach dem Start erscheint die in Abbildung 4.8 gezeigte Oberfläche.

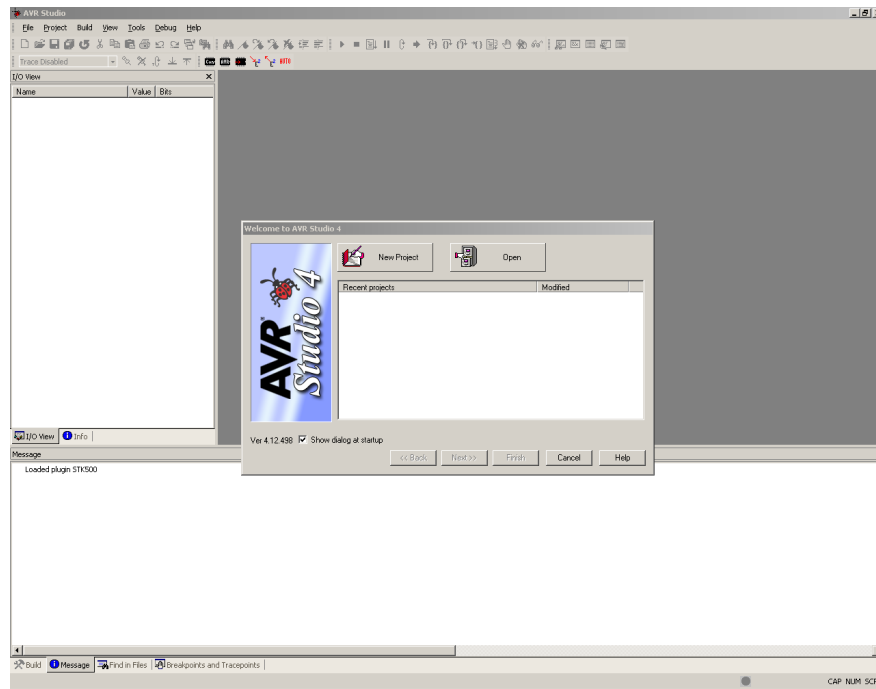


Abbildung 4.8.: Programmoberfläche nach dem Start des AVR Studios

Auf dieser Oberfläche wird mit einem Klick auf «Open» der Dialog zum Öffnen einer Datei beziehungsweise eines Projekts aufgerufen (Abbildung 4.9).

An dieser Stelle gibt es nun zwei Wege zum weiteren Vorgehen:

- Öffnen einer Datei

Mit diesem Weg können Dateien wie beispielsweise Hex- (*.hex) oder elf-Files (*.elf) geladen werden. Die Hex-Files enthalten den binären Code für den *Flash-Speicher* des Mikrocontrollers und werden im *AVR Studio* vorwiegend dazu verwendet, den *Flash-Speicher* per *ISP* zu beschreiben. Die elf-Files hingegen werden benötigt, um mit dem *AVR Studio* Programme zu debuggen. Um das in dieser Diplomarbeit erstellte Programm in den *AVR-Mikrocontroller* zu flashen und anschließend zu debuggen, wird die von *WinAVR* erstellte Datei (main.elf) geöffnet. Nach der Auswahl dieser Datei und einem Klick auf «Öffnen» erscheint das Fenster von Abbildung 4.10.

KAPITEL 4. TEST UND INBETRIEBNAHME

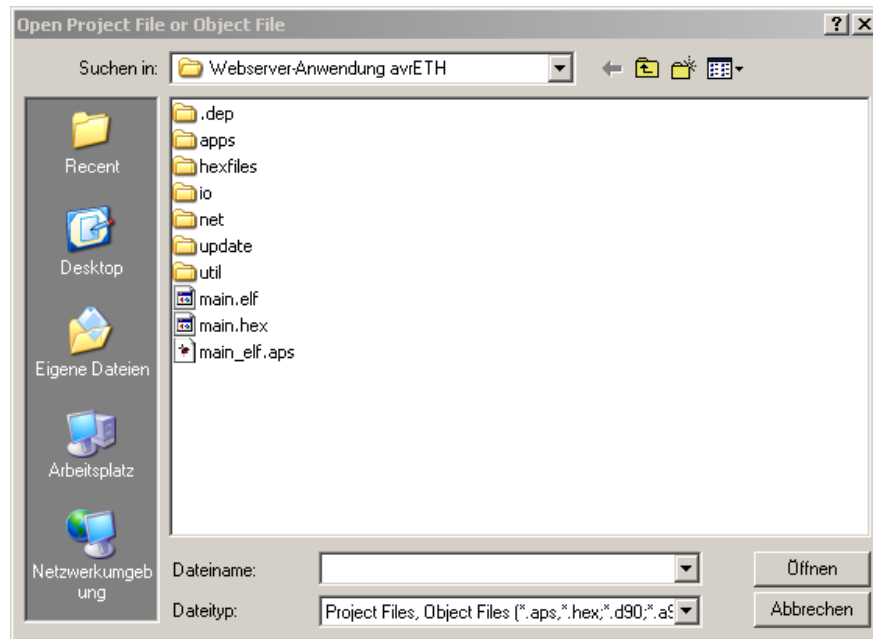


Abbildung 4.9.: Dialog zum Öffnen eines Projekts oder einer Datei

In diesem Fenster wird nachgefragt, wohin das Projekt gespeichert werden soll, welches von *AVR Studio* im folgenden erstellt wird. Im Normalfall kann der vorgeschlagene Pfad einfach mit «Speichern» übernommen werden. Sollte in dem gewählten Pfad bereits ein gleichnamiges Projektfile existieren, wird nachgefragt, ob dieses ersetzt werden soll.

- Öffnen eines Projekts

Mit dieser Möglichkeit kann ein zuvor erstelltes *AVR Studio Projekt* geöffnet werden. Hierzu ist im Dialogfenster zum Öffnen eines Projekts die Projektdatei (`main_elf.aps`) zu selektieren und zu öffnen.

Im darauf erscheinenden Fenster namens «Select device and debug platform» wird auf der linken Seite der verwendete Programmieradapter (*Debug platform*), auf der rechten Seite der verwendete Mikrocontroller (*Device*) und unten die verwendete Schnittstelle (*Port*) ausgewählt.

Zum Einsatz des in dieser Diplomarbeit verwendeten Mikrocontrollers *ATmega32* mit dem Programmieradapter *JTAGICE mkII* werden die Einstellungen wie in Abbildung 4.11 gewählt. Zu diesem Zeitpunkt muss der Programmieradapter an den Rechner und das Zielsystem angeschlossen sein.

Mit dem Bestätigen der «Finish» Schaltfläche werden die getätigten Einstellungen übernommen und die Verbindung zum Zielsystem getestet. Sollte hierbei ein Fehler auftreten und die Hardware nicht fehlerfrei angesprochen werden können, so erscheint die Fehlermeldung aus Abbildung 4.12.

KAPITEL 4. TEST UND INBETRIEBNAHME

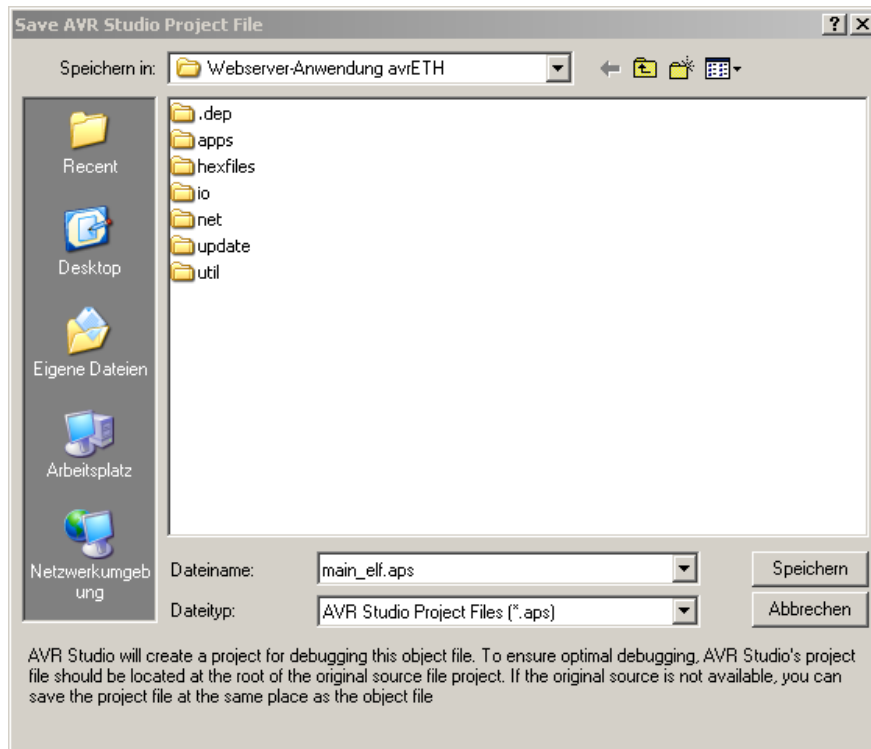


Abbildung 4.10.: Dialog zum Speichern eines Projekts

Wird der Programmieradapter gefunden und kann erfolgreich angesprochen werden, kann unter Umständen eine Meldung wie in Abbildung 4.13 erscheinen. Diese Meldung weist darauf hin, dass für das Programmiergerät eine neuere Firmwareversion verfügbar ist und fragt nach, ob diese installiert werden soll. Im Normalfall kann diese Frage mit «Nein» beantwortet werden, da ein Flashen der Firmware meist nur zu einem unnötigen Risiko führt³.

Nun sollte das *AVR Studio* das Programm in den Mikrocontroller flashen. Für den Fall, dass das Zielsystem nicht gefunden wird (es wird keine *Target-Spannung* (*VTarget*) gefunden), so erscheint die Meldung in Abbildung 4.14.

Sobald alle Hardwarekomponenten des Programmieradapters und Zielsystems korrekt angeschlossen sind, flasht das *AVR Studio* den Mikrocontroller. Anschließend wird eventuell nochmals nach dem Pfad des Projektordners (Abbildung 4.15) gefragt. Dieser kann im Normalfall einfach mit einem Klick auf «Select» übernommen werden.

Nach dieser letzten Auswahl ist das Projekt bereit, wie in Kapitel 4.2.1.2.2 beschrieben, gedebuggt zu werden.

³Nach dem Flashen kann es unter Umständen vorkommen, dass das Gerät seinen Dienst verweigert.

KAPITEL 4. TEST UND INBETRIEBNAHME

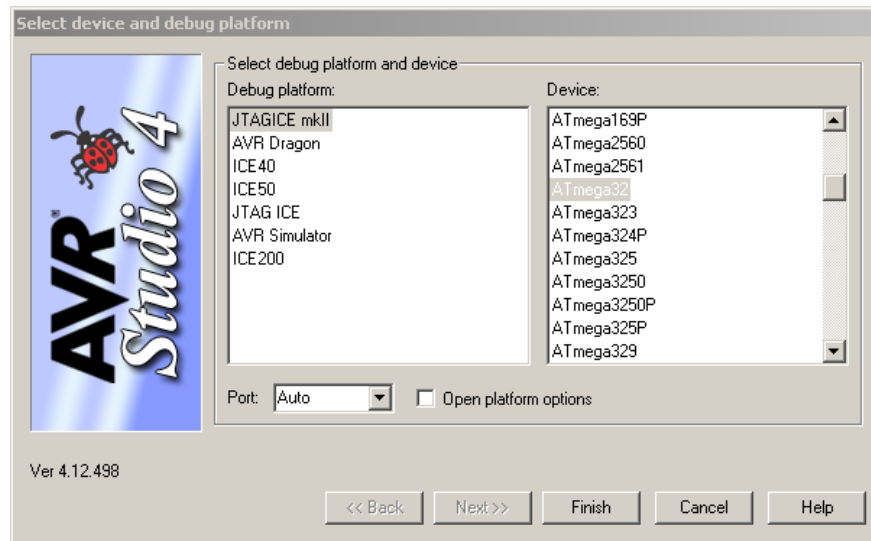


Abbildung 4.11.: Auswahl des verwendeten Bausteins und der Debug-Plattform

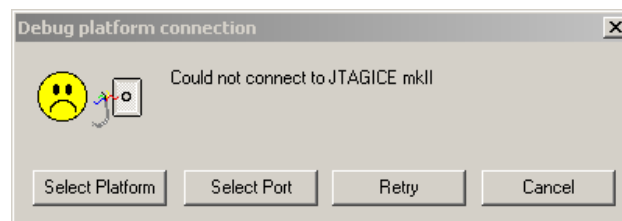


Abbildung 4.12.: Fehler in der Verbindung zum Programmieradapter

4.2.1.2.1. Programmieren und Konfigurieren des Mikrocontrollers Nachdem das Programm in den Mikrocontroller übertragen wurde, wird an dieser Stelle gezeigt, wie eine Verbindung zum Mikrocontroller hergestellt wird, um den *Flash-Speicher* gezielt zu löschen/programmieren oder die *Fuses* des *AVRs* anzupassen.

Zum gezielten Verbinden mit dem Mikrocontroller steht die Symbolleiste aus Abbildung 4.16 zur Verfügung. Von Interesse sind hierbei nur die beiden linken Symbole «Display the 'connect' dialog» und «Connect to the selected AVR programmer», welche allerdings nur zur Verfügung stehen, wenn das *Debuggen* nicht aktiv ist.

Die bereits gezeigte Auswahl des Mikrocontrollers und des Programmiergeräts kann jederzeit über die linke der beiden Schaltflächen («Display the 'connect' dialog») aufgerufen werden. Sind beispielsweise mehrere Programmieradapter am Rechner angeschlossen, kann über den *Connect Dialog* zwischen diesen gewechselt werden.

Wurden die Einstellungen im «Connect Dialog» bereits einmal getätigt, so kann die rechte der

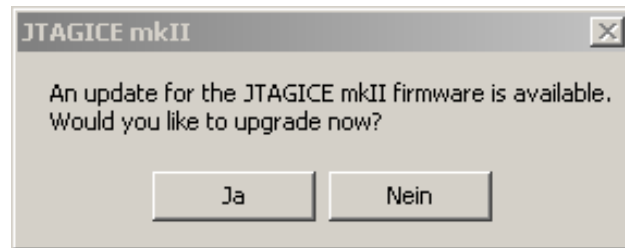


Abbildung 4.13.: Es ist eine neuere Firmwareversion für JTAGICE mkII verfügbar

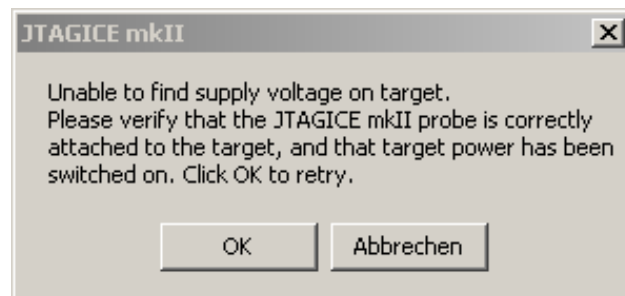


Abbildung 4.14.: Fehlende Target-Spannung

beiden Schaltflächen («Connect to the selected AVR programmer») verwendet werden.

Unabhängig davon, welche der beiden Schaltflächen verwendet wird, erscheint anschließend das Fenster mit den Einstellungen des Programmiergeräts (Abbildung 4.17⁴). Vor diesem Fenster kann eventuell nochmals die Nachfrage kommen, ob die *Firmware* im Programmiergerät aktualisiert werden soll.

Dieses Fenster besitzt die folgenden sechs Registerkarten. Während dem Verwenden der Funktionen auf diesen Registerkarten kann unter Umständen die Fehlermeldung aus Abbildung 4.18 erscheinen. Diese deutet auf einen Konfigurationsfehler hin: Beispielsweise kann die *ISP-Verbindung* fehlerhaft oder der falsche Mikrocontroller als *Device* ausgewählt sein.

- Program

In dieser Registerkarte muss mit *Device* der verwendete Mikrocontroller ausgewählt werden. Mit der Schaltfläche «Erase Device» kann dessen Flash-Speicher anschließend gelöscht werden. Mit *Programming mode* wird eingestellt, über welche Schnittstelle der Programmieradapter mit dem Zielsystem verbunden ist (beim *STK500* ist dies normalerweise *ISP mode*, beim *JTAGICE mkII* ist es standardmäßig *JTAG mode*). Mit den beiden Haken rechts neben der

⁴Die Anzeige kann von Programmiergerät zu Programmiergerät etwas differieren. In der Abbildung wird ein Beispiel für das *JTAGICE mkII* und ein *STK500* gezeigt.

KAPITEL 4. TEST UND INBETRIEBNAHME

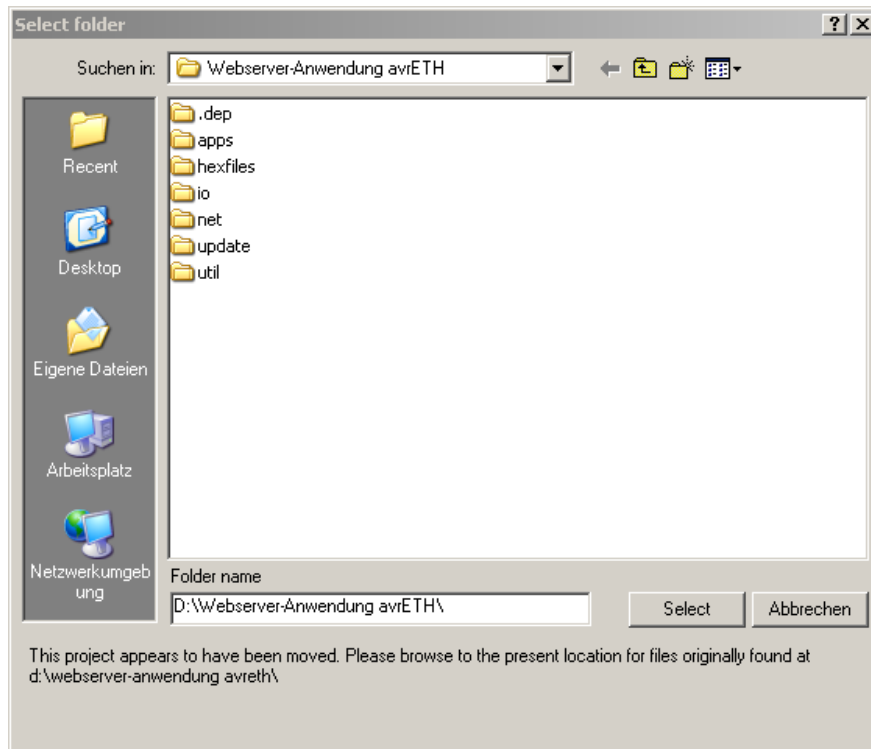


Abbildung 4.15.: Auswahl des Projektverzeichnisses



Abbildung 4.16.: Schaltflächen zum Verbinden mit dem Mikrocontroller

Auswahl des Programmiermodus kann ausgewählt werden, ob der Mikrocontroller vor jeder Programmierung gelöscht werden (*Erase Device Before Programming*) oder der Speicherinhalt nach jeder Programmierung überprüft werden soll (*Verify Device After Programming*).

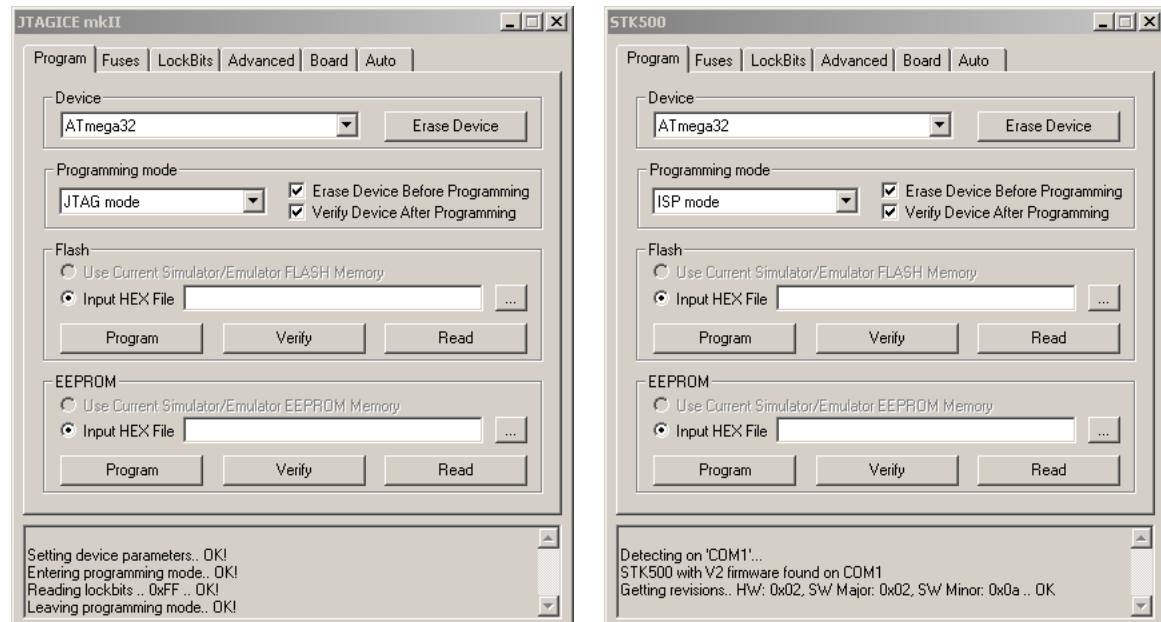
Mit dem Bereich *Flash* kann der Programmspeicher des Zielcontrollers ausgelesen und beschrieben werden. Der Bereich *EEPROM* bietet die gleichen Funktionen für das interne *EEPROM* des Mikrocontrollers.

Das unterste Feld auf dieser Registerkarte zeigt Statusinformationen zu den aktuellen Operationen an.

- Fuses

Auf der Registerkarte von Abbildung 4.19 können die Fuses des Mikrocontrollers programmiert und gelöscht werden. Laut Datenblättern der *Atmel Corporation* [23] wird eine Option im Controller durch ein gelöscht Bit aktiviert; ein gesetztes Bit deaktiviert die jeweilige

KAPITEL 4. TEST UND INBETRIEBNAHME



(a) Eigenschaften des JTAGICE mkII

(b) Eigenschaften des STK500

Abbildung 4.17.: Eigenschaften der Programmieradapter

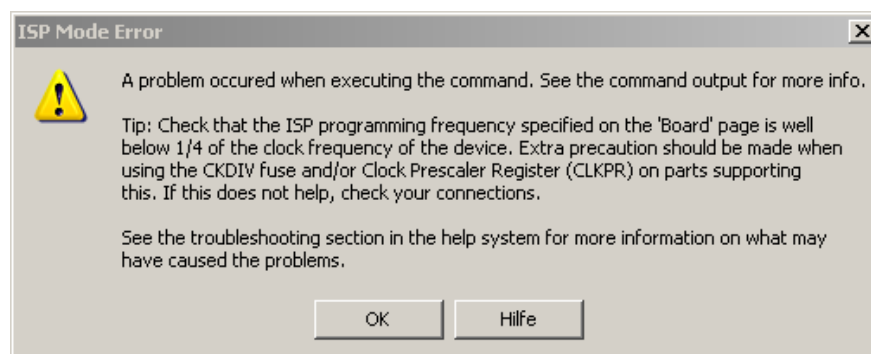


Abbildung 4.18.: ISP-Fehler

KAPITEL 4. TEST UND INBETRIEBNAHME

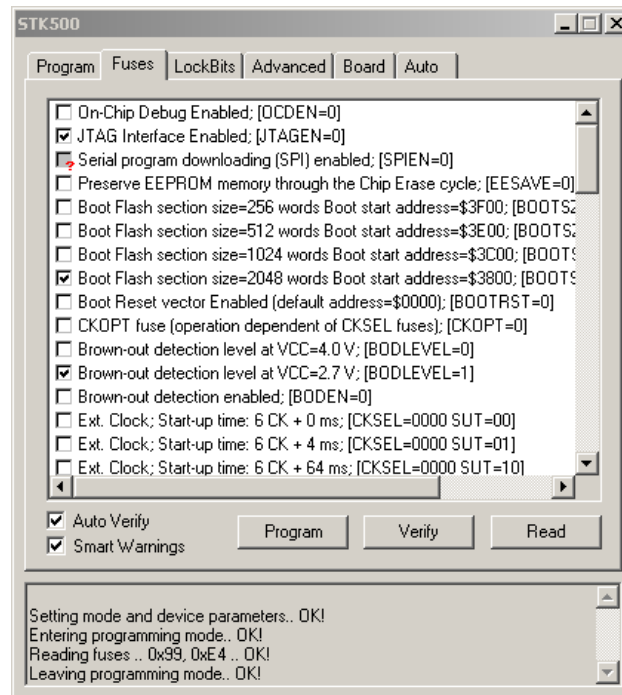


Abbildung 4.19.: Einstellen der Fuses

Funktion. In der Liste der Fuses bedeutet ein Haken, dass das jeweilige Bit gelöscht ist. Eine Funktion wird mit einem gesetzten Haken aktiviert. Mit den Schaltflächen «Program» (Fuses programmieren), «Verify» (die eingestellten Fuses mit den im Controller vorhandenen Fuses vergleichen) und «Read» (Fuses auslesen) kann anschließend die gewünschte Funktion ausgeführt werden. Die beiden Haken *Auto Verify* und *Smart Warnings* sollten beide gesetzt bleiben.

Die einzelnen *Fuses* und deren Bedeutungen variieren von Mikrocontroller zu Mikrocontroller und sollten daher im Datenblatt des jeweiligen Bausteins nachgelesen werden. Mit den Fuse-Einstellungen sollte allerdings sehr vorsichtig umgegangen werden, da der Mikrocontroller beispielsweise bei der Auswahl einer falschen Taktquelle seinen Dienst verweigert.

Die im Rahmen dieser Diplomarbeit relevanten Fuses werden nachfolgend kurz vorgestellt:

- JTAG Interface Enabled

Diese Option muss gesetzt sein, damit das *JTAG-Interface* auf dem Mikrocontroller aktiviert ist. Nur auf diese Weise kann der Mikrocontroller im Rahmen dieser Diplomarbeit gedebuggt werden.

KAPITEL 4. TEST UND INBETRIEBNAHME

– Einstellungen des Clock-Systems

Es ist eine Reihe von Fuses zum Einstellen der Taktquelle des Systemtakts vorhanden. Hierbei gibt es die Möglichkeit, einen externen Takt «Ext. Clock», den internen RC-Oszillator «Int. RC Osc.», einen externen RC-Oszillator «Ext. RC Osc.» oder einen externen Quarz «Ext. Crystal/Resonator» zu verwenden. Für weiterführende Informationen zum Einstellen des Taktsystems wird an dieser Stelle wieder auf das Datenblatt des jeweiligen *AVR-Mikrocontrollers* verwiesen.

Zur Verwendung des Mikrocontrollerprogramms dieser Diplomarbeit kann der *AVR* im einfachsten Fall mit dem internen 8-MHz-RC-Oszillator betrieben werden, eine dafür geeignete Fuse ist zum Beispiel «Int. RC Osc. 8 MHz; Start-up time: 6 CK + 64 ms».

• LockBits

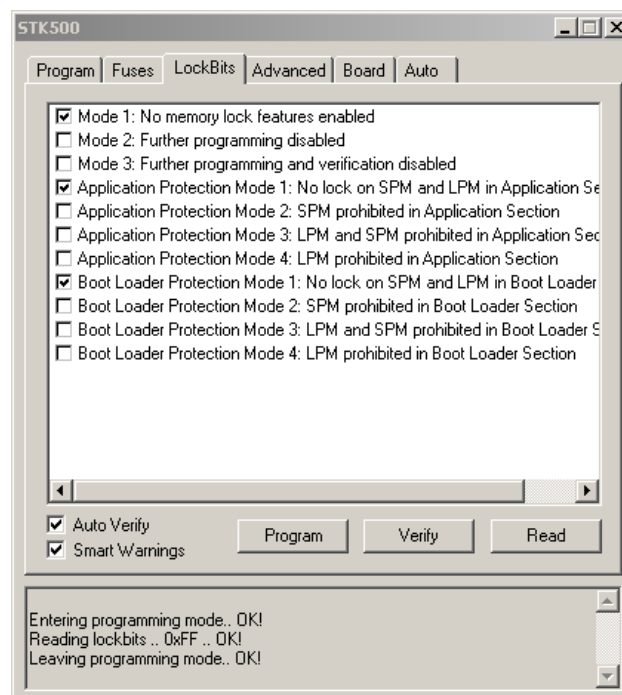


Abbildung 4.20.: Einstellung der LockBits

Mit den in Abbildung 4.20 dargestellten *LockBits* können einzelne Bereiche des Flash-Speichers gegen unberechtigtes Lesen und/oder Schreiben geschützt werden. Da diese Funktion eher selten benötigt wird, wird auch an dieser Stelle wieder auf das Datenblatt des jeweiligen Bausteins verwiesen.

KAPITEL 4. TEST UND INBETRIEBNAHME

- Advanced

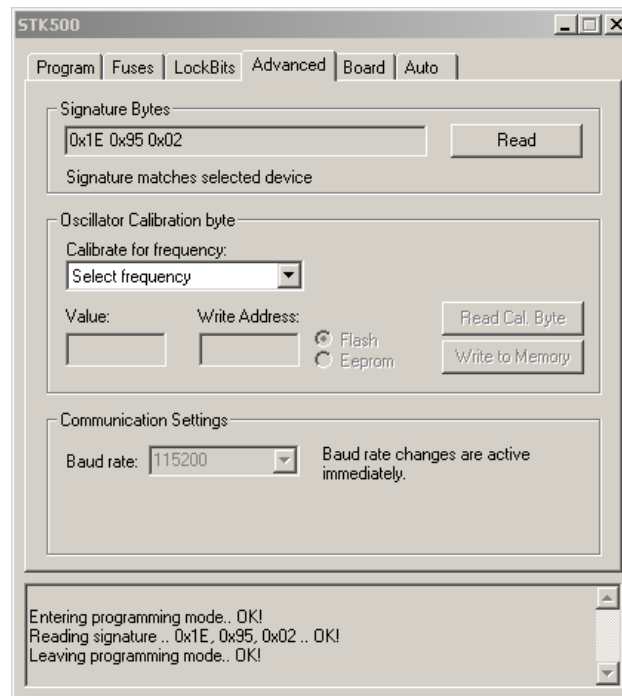


Abbildung 4.21.: Erweiterte Einstellungen

In der Registerkarte *Advanced* (Abbildung 4.21) kann die digitale Signatur des Mikrocontrollers ausgelesen werden. Jeder der *Atmel AVR Mikrocontroller* besitzt eine drei Bytes lange Signatur, die einen Controllertyp eindeutig identifiziert. Durch einen Klick auf die Schaltfläche «Read» kann diese Signatur ausgelesen und mit dem unter der Registerkarte *Program* eingestellten Controllertyp verglichen werden.

Der Abschnitt *Oscillator Calibration byte* bietet die Möglichkeit, den Oszillator des Mikrocontrollers nachzukalibrieren. Genauere Infos hierzu sind dem entsprechenden Datenblatt zu entnehmen.

- Board

Dieser Teil der Einstellungen bietet die Möglichkeit, die Parameter des Programmierboards, wie beispielsweise des *STK500*, anzupassen. Abbildung 4.22 zeigt die vorhandenen Optionen an. Mit den Schiebereglern im oberen Bereich können die Spannung des Zielsystems (*VTarget*) und die Referenzspannung (*ARef*) angepasst werden. Die somit eingestellten Spannungen können mit einem Klick auf «Write Voltages» in das *STK500* geschrieben werden. Dabei ist zu beachten, dass *ARef* niemals größer als *VTarget* sein kann. Ein Klick auf «Read Voltages» liest die aktuellen Spannungswerte des *STK500* aus und stellt diese mit den beiden Schiebereglern

KAPITEL 4. TEST UND INBETRIEBNAHME

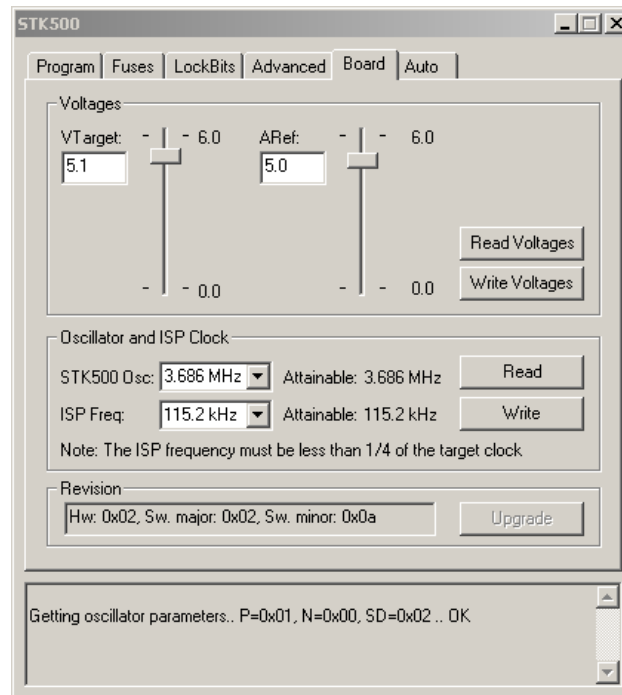


Abbildung 4.22.: Einstellungen des Programmierboards

und Wertangaben dar. Zur Verwendung des *STK-LAN* muss *VTarget* auf 5,0 Volt eingestellt werden.

Mit den Einstelloptionen im Bereich *Oscillator and ISP Clock* kann die Taktfrequenz des Mikrocontrollers sowie die Geschwindigkeit der *ISP-Kommunikation* ausgelesen und geändert werden.

Das *STK500* kann eine Taktfrequenz (*STK500 Osc*:) von 32,7 Kilohertz (kHz), 1,23 MHz, 1,84 MHz oder 3,69 MHz zur Verfügung stellen. Eine höhere Taktrate als 3,69 MHz ist auf diese Weise nicht möglich. Hierzu kann auf das *STK500* jedoch ein externer Quarz gesteckt oder der Mikrocontroller auf den internen RC-Oszillator (bis 8 MHz bei einem *ATmega32*) konfiguriert werden.

Mit «ISP Freq:» kann der *SPI-Takt* für *ISP* gewählt werden. Der Takt darf niemals höher als ein Viertel des Systemtakts sein, um eine zuverlässige Verbindung zu gewährleisten. In den meisten Fällen ist eine *ISP Frequenz* von 115,2 kHz eine gute Wahl.

- Auto

In der Registerkarte *Auto* (Abbildung 4.23) können mehrere Operationen markiert und dann in einer Liste abgearbeitet werden.

KAPITEL 4. TEST UND INBETRIEBNAHME

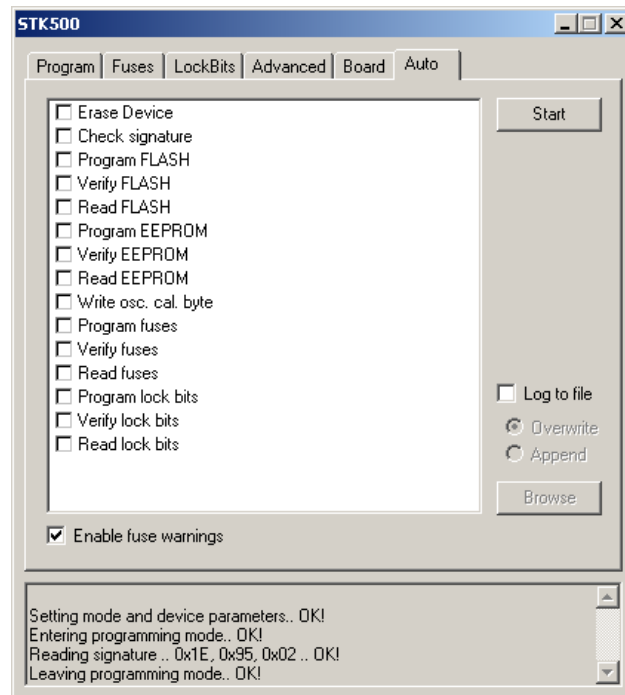


Abbildung 4.23.: Definieren einer Befehlsliste

4.2.1.2.2. Debuggen des Programms Sollte sich das in den Mikrocontroller geflashte Programm nicht wie erwartet verhalten, kann es sinnvoll sein, das Programm zu Debuggen, also nachzuschauen, was wirklich innerhalb des Systems passiert. Die Entwicklungsumgebung besitzt hierzu in der Kopfzeile eine Reihe von Schaltflächen (Abbildung 4.24).

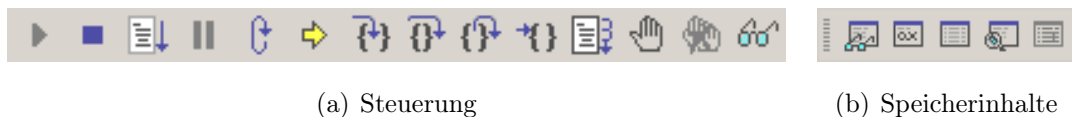


Abbildung 4.24.: Schaltflächen zum Steuern des Debuggens und Betrachten von Speicherinhalten

Diese Schaltflächen lassen sich in zwei Gruppen aufspalten:

- Steuerung des Debuggens (Abbildung 4.24(a))

Die Bedeutungen der Debug-Schaltflächen von links nach rechts:

- Start Debugging

Mit dieser Schaltfläche kann das Debuggen gestartet werden. Beim Betreten des Debugmodus verbindet sich das *AVR Studio* mit dem Mikrocontroller, resetet diesen und lädt

KAPITEL 4. TEST UND INBETRIEBNAHME

den Programmcode in dessen Speicher. Sobald sich der Mikrocontroller im Debugmodus befindet, läuft dessen Programm nicht mehr selbstständig ab, sondern wird von der Debugumgebung gesteuert.

– Stop Debugging

Hiermit kann der Debugvorgang beendet werden. Nach Betätigen dieser Schaltfläche wird das Debuggen beendet und der Mikrocontroller zurückgesetzt. Anschließend beginnt das Programm im Mikrocontroller wieder selbstständig zu laufen.

– Run

Mit *Run* kann ein kontinuierlicher Programmablauf gestartet oder fortgesetzt werden. Das Programm läuft dabei solange, bis es wieder explizit gestoppt wird oder einen *Breakpoint* erreicht. Der Mikrocontroller verhält sich in diesem Modus weitestgehend so, als wäre er nicht im Debugmodus, sondern würde sein Programm normal abarbeiten. Er kann jedoch jederzeit angehalten werden.

– Break

Mit diesem Befehl wird ein Mikrocontroller, welcher sich im *Run* Modus befindet, angehalten.

– Reset

Die Schaltfläche *Reset* setzt den Mikrocontroller zurück, das heißt sein Programm beginnt wieder vom Anfang an, ganz so, als ob der Controller einen Hardware-Reset bekommen hätte.

– Show Next Statement

Diese Funktion setzt den gelben Marker im Codefenster auf die Programmzeile, welche als nächstes ausgeführt wird.

– Step Into

Wenn sich der Mikrocontroller im *Break* Zustand befindet, kann mit dieser Schaltfläche der nächste Befehl ausgeführt werden. Wenn als nächstes eine Unterfunktion folgt, wird in diese hineingesprungen.

KAPITEL 4. TEST UND INBETRIEBNAHME

– Step Over

Mit einem Klick auf diese Schaltfläche wird wie bei *Step Into* der folgende Befehl ausgeführt. Der Unterschied zu *Step Into* liegt darin, dass eine Unterfunktion nicht betreten, sondern wie ein einziger Befehl abgearbeitet wird. Befindet sich in dieser Unterfunktion ein Breakpoint, so stoppt das Programm dennoch an diesem Punkt.

– Step Out

Mit dieser Funktion kann aus einer Unterfunktion herausgesprungen werden. Ein Klick auf die Schaltfläche führt das Programm solange aus, bis die Unterfunktion wieder verlassen wird, das Programm also in der Hierarchie eine Stufe nach oben wechselt. Wird während dem Ausführen ein Breakpoint erreicht, so wird die Programmausführung angehalten. Sollte sich das Programm bereits auf der obersten Hierarchieebene befinden, wenn diese Funktion aufgerufen wird, so wird das Programm solange ausgeführt, bis es explizit angehalten wird oder einen Breakpoint erreicht.

– Run to Cursor

Durch diese Schaltfläche kann das Programm bis zur aktuellen Cursorposition ausgeführt werden, es wird dabei nicht durch Breakpoints unterbrochen. Sollte sich der Cursor an einer Stelle befinden, welche vom Programmablauf nie erreicht wird, so läuft das Programm solange, bis es vom Benutzer angehalten wird.

– AutoStep

Die Funktion *AutoStep* führt das Programm automatisch Schritt für Schritt aus und aktualisiert nach jedem Schritt alle Fenster der Debugoberfläche. Die Programmausführung erfolgt bis zu einem Breakpoint oder kann vom Benutzer unterbrochen werden.

– Toggle Breakpoint

Mit *Toggle Breakpoint* kann ein Breakpoint im Sourcecode gesetzt werden. Der gesetzte Breakpoint wird durch einen roten Punkt links neben der entsprechenden Codezeile angezeigt. Ein erneutes Betätigen dieser Schaltfläche entfernt den Breakpoint wieder.

– Remove all Program Breakpoints

Mit Hilfe dieser Schaltfläche können alle Breakpoints entfernt werden.

KAPITEL 4. TEST UND INBETRIEBNAHME

- Quickwatch

Durch *Quickwatch* kann die im Sourcecode momentan markierte Variable in das *Watch Window* übernommen werden.

- Fenster zum Betrachten von Speicherinhalten (Abbildung 4.24(b))

Die *Watch-Windows* haben die folgenden Bedeutungen (von links nach rechts):

- Toggle Watch Windows

Mit dieser Schaltfläche kann das *Watch Window* ein- und ausgeblendet werden. Es dient zur Anzeige von Variablenwerten zur Laufzeit. Sobald der Debugvorgang angehalten wird, beispielsweise durch *Break* oder einen Breakpoint, werden die Variablenwerte im *Watch Window* aktualisiert. Wenn sich ein Variablenwert im letzten Debugschritt geändert hat, wird dieser Wert im *Watch Window* rot dargestellt.

- Toggle Register Window

Mit dem *Register Window* können die Inhalte der Arbeitsregister *R0* bis *R31* betrachtet und auch geändert werden. Wie im vorigen *Watch Window* werden auch im *Register Window* diejenigen Werte rot dargestellt, die sich im letzten Schritt des Debuggens geändert haben.

- Toggle Memory Window

Mit Hilfe des *Memory Windows* können die einzelnen Speicherbereiche des Mikrocontrollers angeschaut und auch geändert werden. Die dabei auswählbaren Optionen sind *Data* (der Arbeitsspeicher des Mikrocontrollers), *EEPROM* (der Inhalt des EEPROM), *I/O* (der Ein-/Ausgabebereich), *Program* (der Programmspeicher im Flash) und *Register* (die Arbeitsregister). Die im letzten Debugschritt veränderten Speicherinhalte werden in der Ansicht rot dargestellt.

- Toggle Disassembler Window

Das *Disassembler Window* ermöglicht das Anzeigen des Assemblercodes zu den einzelnen Anweisungen im C-Code. Dadurch kann beispielsweise nachgeschaut werden, wie der Compiler einzelne C-Anweisungen in Assembler realisiert hat.

KAPITEL 4. TEST UND INBETRIEBNAHME

– Toggle Trace Window

Die Funktion der Schaltfläche *Trace Window* ist nur in Verwendung mit der Debughardware ICE40 und ICE50 vorhanden. Bei Verwendung des *JTAGICE mkII* oder *AVR-JTAG-USB* ist diese Schaltfläche deaktiviert.

Beim *Atmel AVR Studio* tritt teilweise der Fehler auf, dass die Fenster zum Betrachten von Speichereinhalten, also beispielsweise das *Watch Window*, nicht mehr reagieren. In diesem Fall müssen im Menü über «Tools - Options...» die *Options* aufgerufen werden. Dort wird der Haken bei «Reset Desktop on restart» gesetzt und diese Angabe mit einem Klick auf «OK» übernommen. Anschließend sollten die Fenster wieder wie gewohnt funktionieren.

Als weitere nützliche Funktion können die wichtigsten Register des Mikrocontrollers auch über die *I/O View* in der linken Seitenleiste ausgelesen und verändert werden.

4.2.2. SNMP

Um die SNMP-Funktionalität des Mikrocontrollerprogramms auf Applikations- sowie Bitebene zu testen, werden die nachfolgend vorgestellten Tools verwendet. Die im Mikrocontrollerprogramm implementierten *Managed Objects* sind in Anhang K zusammengefasst.

4.2.2.1. SnmpTool

Das kostenlose Kommandozeilenprogramm *SnmpTool* steht auf der Website von *Jürgen Klein* [91] zum Herunterladen zur Verfügung und läuft ohne vorherige Installation unter *Microsoft Windows* in der *MS-DOS Eingabeaufforderung*. Das Tool unterstützt das Senden der SNMP-Nachrichtentypen *GET*, *GETNEXT*, *SET* und *TRAP*. Der komplette *MIB-Tree* kann zusätzlich durch den Parameter *walk* ausgelesen werden. Dabei werden nacheinander *GETNEXT-Anfragen* gestellt, bis die komplette Baumstruktur ausgelesen ist.

Nach der Online-Hilfe, welche durch alleinigen Aufruf von *snmptool* ohne Parameter angezeigt wird, stehen die nachfolgenden Parameter zur Verfügung:

```
SnmpTool - Simple Network Management Protocol Tool for Win32
Version 1.0 (28 Aug 1997); James D. Murray <jdm@oreilly.com>
```

Usage:

KAPITEL 4. TEST UND INBETRIEBNAHME

```
SnmpTool [-rt] {get|getnext} agent community oid [oid] [...]  
SnmpTool [-rt] set agent community oid type value [oid type value] [...]  
SnmpTool [-inrtx] poll agent community oid [oid] [...]  
SnmpTool [-lrt] walk agent community oid  
SnmpTool [-inp] trap  
SnmpTool @file
```

Options with arguments are:

```
-t <request time out in milliseconds>  
-r <request retries attempts before timing out>  
-i <polling interval in milliseconds>  
-n <number of polling requests>
```

Valid 'type' are:

1 = INTEGER	5 = Counter
2 = OCTET STRING	6 = Gauge
3 = OBJECT IDENTIFIER	7 = TimeTicks
4 = IpAddress	8 = Opaque

Es kann also beispielsweise über das Kommando «*snmptool get 192.168.10.123 public .1.3.6.1.2.1.1.0*» die *sysDescr* des Agenten 192.168.10.123 ausgelesen werden (Abbildung 4.25).

SNMPtool unterstützt absolute und relative OIDs. Die *Global ID* einer relativen Angabe lautet *.1.3.6.1.2.1*.

4.2.2.2. MG-Soft MIB Browser

Beim *MIB Browser* der *MG-SOFT Corporation* [24] handelt es sich um ein kostenpflichtiges Programmpaket für *SNMP*. Auf der Website des Herstellers kann für nicht kommerzielle Projekte eine kostenlose 30-Tage-Evaluierungslizenz erworben werden.

Im Folgenden werden diejenigen Programmteile näher beschrieben, welche im Rahmen dieser Diplomarbeit verwendet werden.

KAPITEL 4. TEST UND INBETRIEBNAHME

```
D:\SNMP>snmptool get 192.168.10.123 public .1.3.6.1.2.1.1.1.0
SnmpTool - Simple Network Management Protocol Tool for Win32
ErrorStatus: 0 (No Error)
ErrorIndex: 0
varbind 1:
  Name: system.sysDescr.0
  OID: 1.3.6.1.2.1.1.1.0
  Type: OCTET STRING
  Length: 31
  Value: ATmega32 with embedded ethernet
D:\SNMP>
```

Abbildung 4.25.: GET-Request mit SnmpTool

4.2.2.2.1. MIB Browser Dieser Programmteil dient dazu, Anfragen an den *SNMP-Agent* zu senden und *TRAP-Nachrichten* sowie *RESPONSES* zu empfangen. Das Programm wird über «Start - Programme - MG-SOFT MIB Browser - MIB Browser» gestartet und zeigt direkt nach dem Start die Oberfläche aus Abbildung 4.26.

Um alle *Managed Objects* korrekt anzeigen zu können, wird zuerst die zum *STK-LAN* gehörende *MIB*, wie in Anhang A.8.4 beschrieben, importiert. Anschließend wird unter der Registerkarte «Query» die *IP-Adresse* des *SNMP-Agent* in das Feld *Remote SNMP agent* (Abbildung 4.27) eingetragen. Rechts neben dem IP-Feld können mit der Schaltfläche *SNMP Protocol Preferences* weitere Einstellungen zum *SNMP-Agent* aufgerufen werden. In diesem Einstellungsdialog wird die «SNMP protocol version» auf *SNMPv1* eingestellt. Die «Read community» und «Set community» werden auf den Wert *public* und die «Port number» auf den Wert 161 gesetzt. Anschließend wird das Einstellungsfenster mit einem Klick auf «OK» geschlossen.

Mit einem Rechtsklick auf einen Teil des *MIB-Tree* beziehungsweise über den Menüpunkt *SNMP* werden die verschiedenen *SNMP-Requests* aufgerufen, wie die Abbildungen 4.28 und 4.29 verdeutlichen.

Der *MIB Browser* beinhaltet zusätzlich einen *Trap Receiver*. Dieser öffnet sich automatisch, sobald ein *TRAP* empfangen wird, kann aber auch manuell über den Menüpunkt «Tools» und «Trap Ringer Console» aufgerufen werden. Das daraufhin erscheinende Fenster (Abbildung 4.30) zeigt alle empfangenen *TRAP-Nachrichten* mit ihren Details an. Im abgebildeten Beispiel ist ein *coldStart-*

KAPITEL 4. TEST UND INBETRIEBNAHME

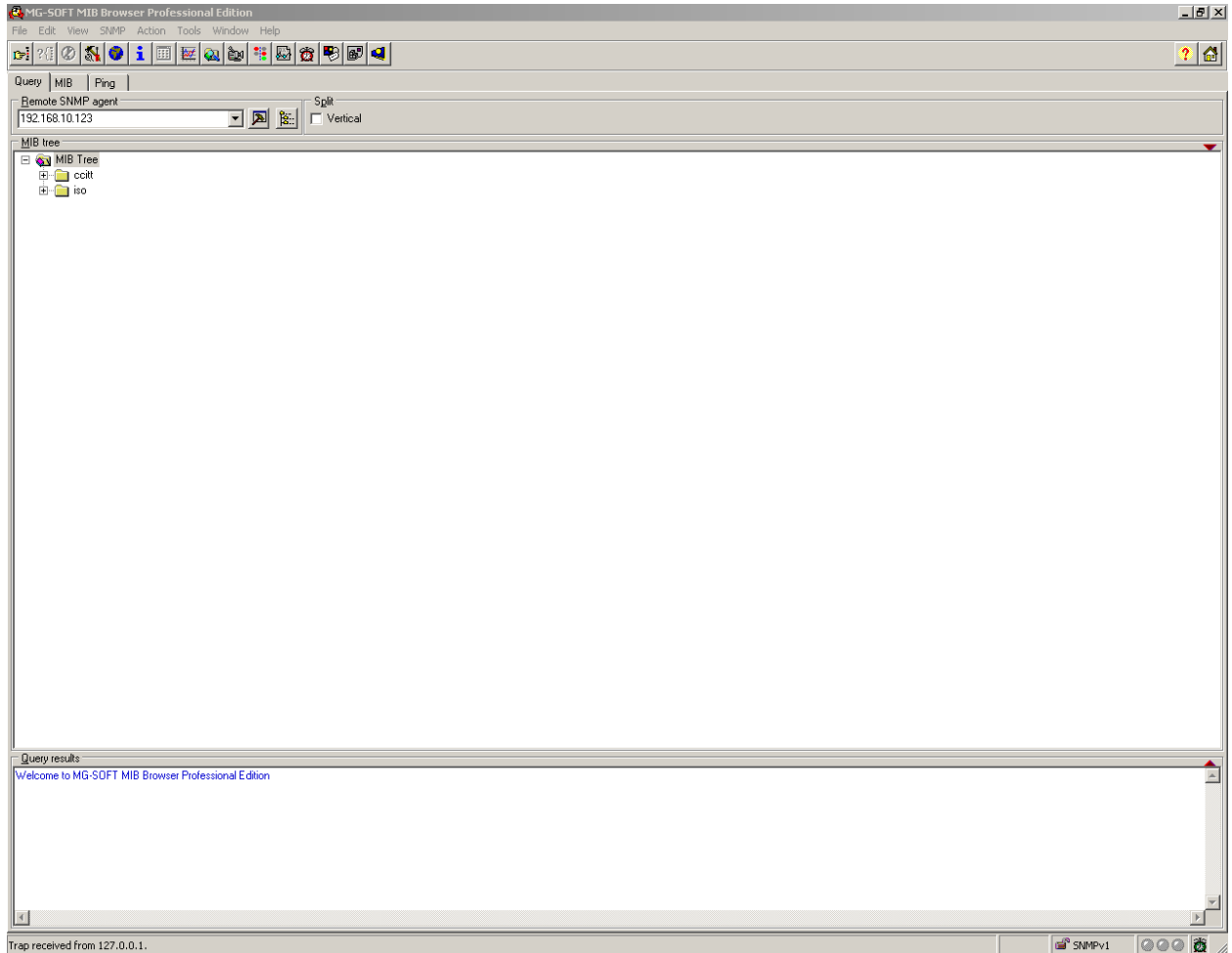


Abbildung 4.26.: Programmoberfläche des MG-SOFT MIB Browsers

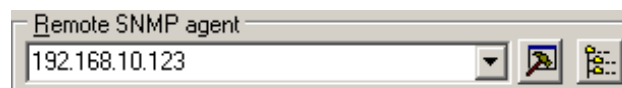


Abbildung 4.27.: Einstellungen zum Verbinden mit dem SNMP-Agent

KAPITEL 4. TEST UND INBETRIEBNAHME

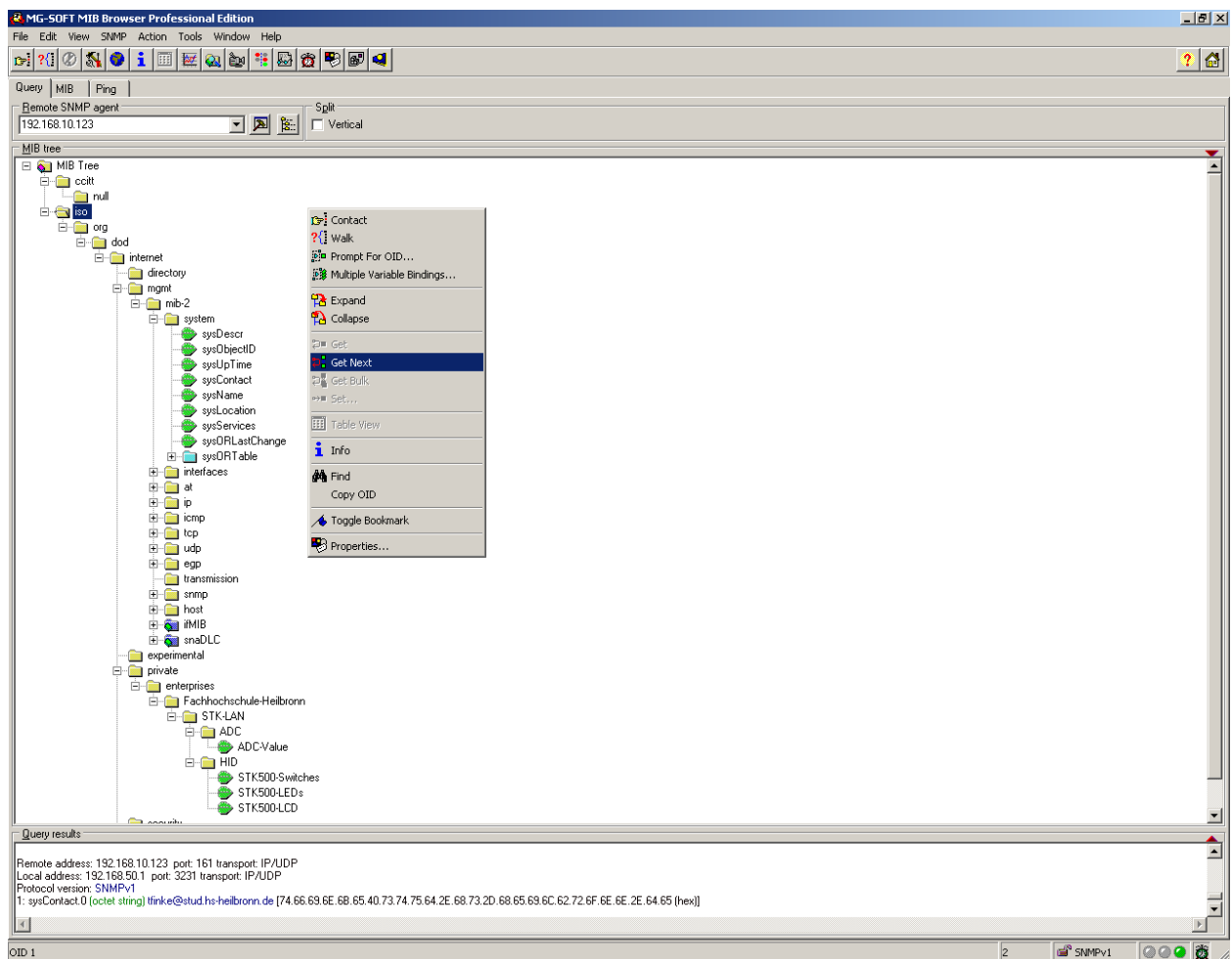


Abbildung 4.28.: Aufrufen der verschiedenen SNMP-Requests über einen Rechtsklick

KAPITEL 4. TEST UND INBETRIEBNAHME

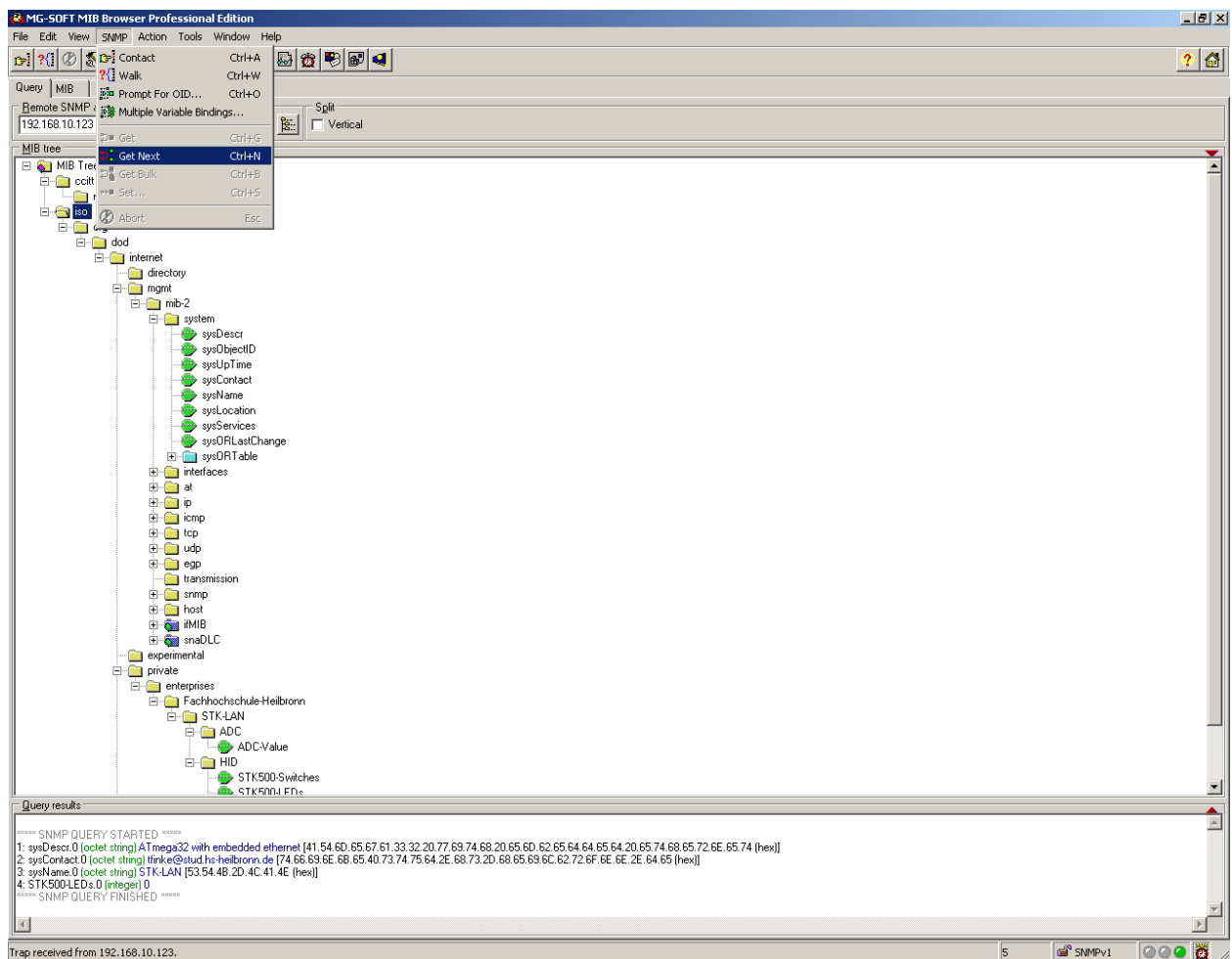


Abbildung 4.29.: Aufrufen der verschiedenen SNMP-Requests über das Menü

KAPITEL 4. TEST UND INBETRIEBNAHME

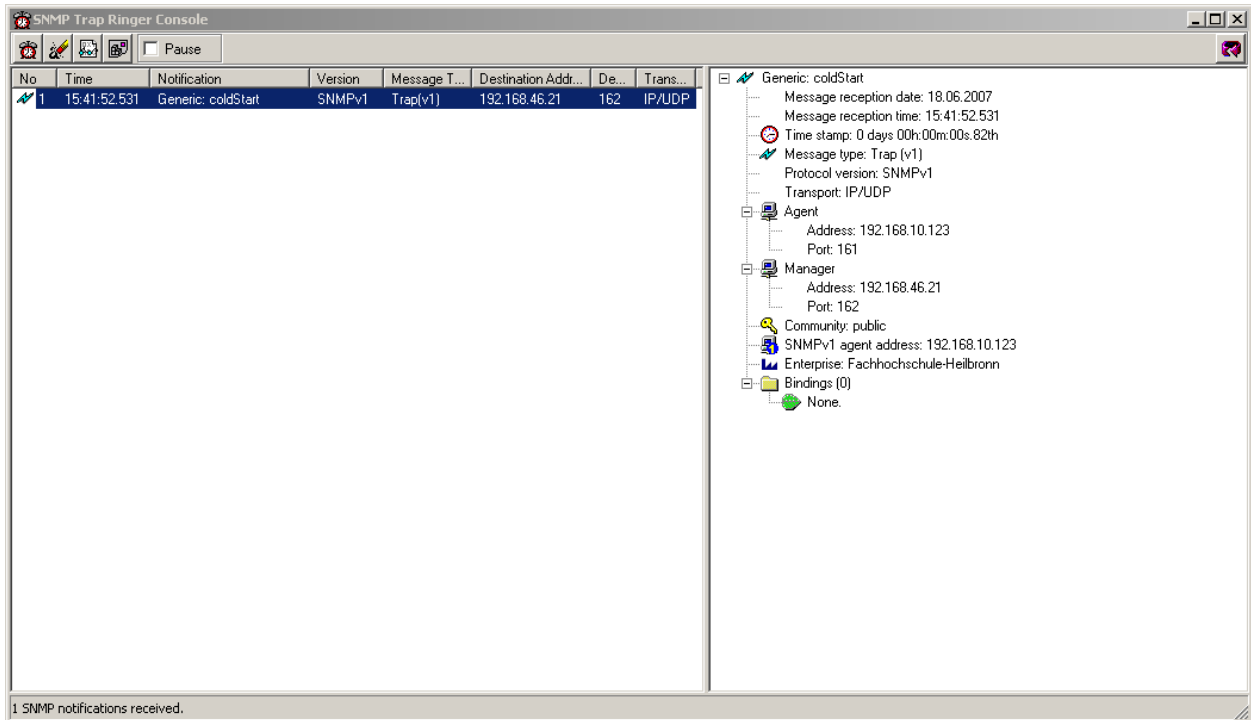


Abbildung 4.30.: Oberfläche der «SNMP Trap Ringer Console»

TRAP vom *STK500* zu sehen. Durch Betätigung des Tasters «SW3» auf dem *STK500* kann eine «Enterprise Specific»-TRAP-Nachricht ausgelöst werden.

4.2.2.2.2. MIB Compiler Der *MIB Compiler*, welcher über «Start - Programme - MG-SOFT MIB Browser - MIB Compiler» aufgerufen wird, dient zum Kompilieren einer *MIB-Datei*. Da die Anwendung dieses Tools in Anhang A.8.4 beschrieben ist, wird an dieser Stelle nicht weiter darauf eingegangen.

4.2.2.3. iReasoning MIB Browser

Ein weiteres Programmpaket für *SNMP* ist der *iReasoning MIB Browser*, welcher auf der Website des Herstellers [61] heruntergeladen werden kann. Dieses Programm gibt es als «Professional Edition» für kommerzielle Anwendungen sowie als kostenlose «Personal Edition» für den privaten Gebrauch.

Das Programmpaket unterteilt sich in die zwei Tools:

KAPITEL 4. TEST UND INBETRIEBNAHME

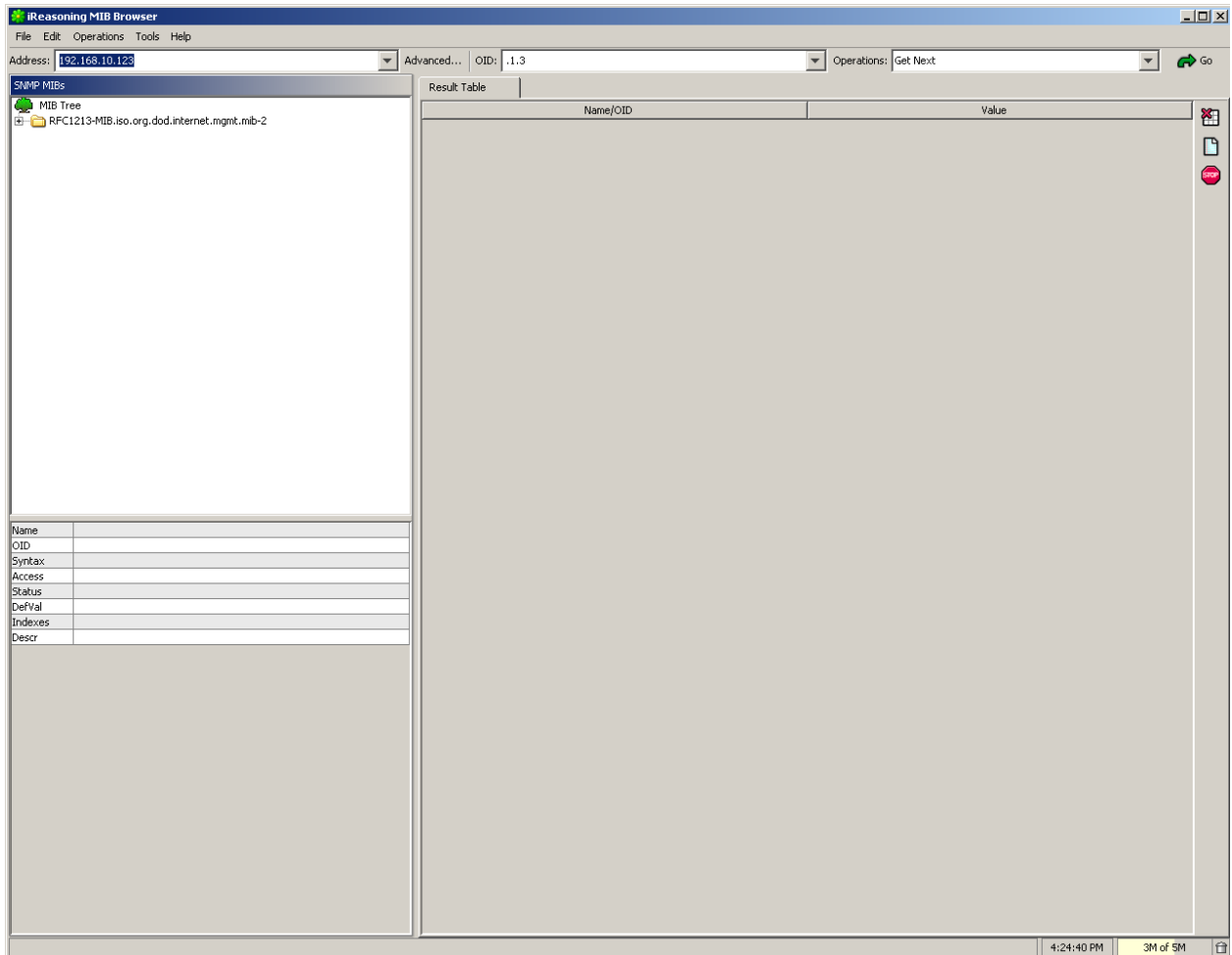


Abbildung 4.31.: Die Oberfläche des iReasoning MIB Browser

4.2.2.3.1. MIB Browser Nach dem Start des Programms über «Start - Programme - iReasoning - MIB Browser - MIB Browser» erscheint die Programmoberfläche (Abbildung 4.31) des *MIB Browsers*.

An dieser Stelle wird über das Feld «Address» die *IP-Adresse* des *SNMP-Agent* angegeben. Durch einen Klick auf die Schaltfläche «Advanced...» erscheinen wie beim *MG-SOFT MIB Browser* die erweiterten Einstellungen für den *SNMP-Agent*. Hier wird als «Port» der Wert 161 und als «Read-» und «Write Community» der Wert «public» angegeben. Die «SNMP Version» wird für eine erfolgreiche Kommunikation auf «1» gesetzt. Mit einem anschließenden Klick auf die Schaltfläche «Ok» werden die Einstellungen übernommen.

Bei Bedarf kann nun über den Menüpunkt «File» und den Eintrag «Load MIB» eine *MIB-Beschreibungsdatei* geladen werden. Nach dem Laden einer solchen Beschreibungsdatei sieht der angezeigte *MIB-Tree* beispielsweise wie in Abbildung 4.32 aus.

KAPITEL 4. TEST UND INBETRIEBNAHME

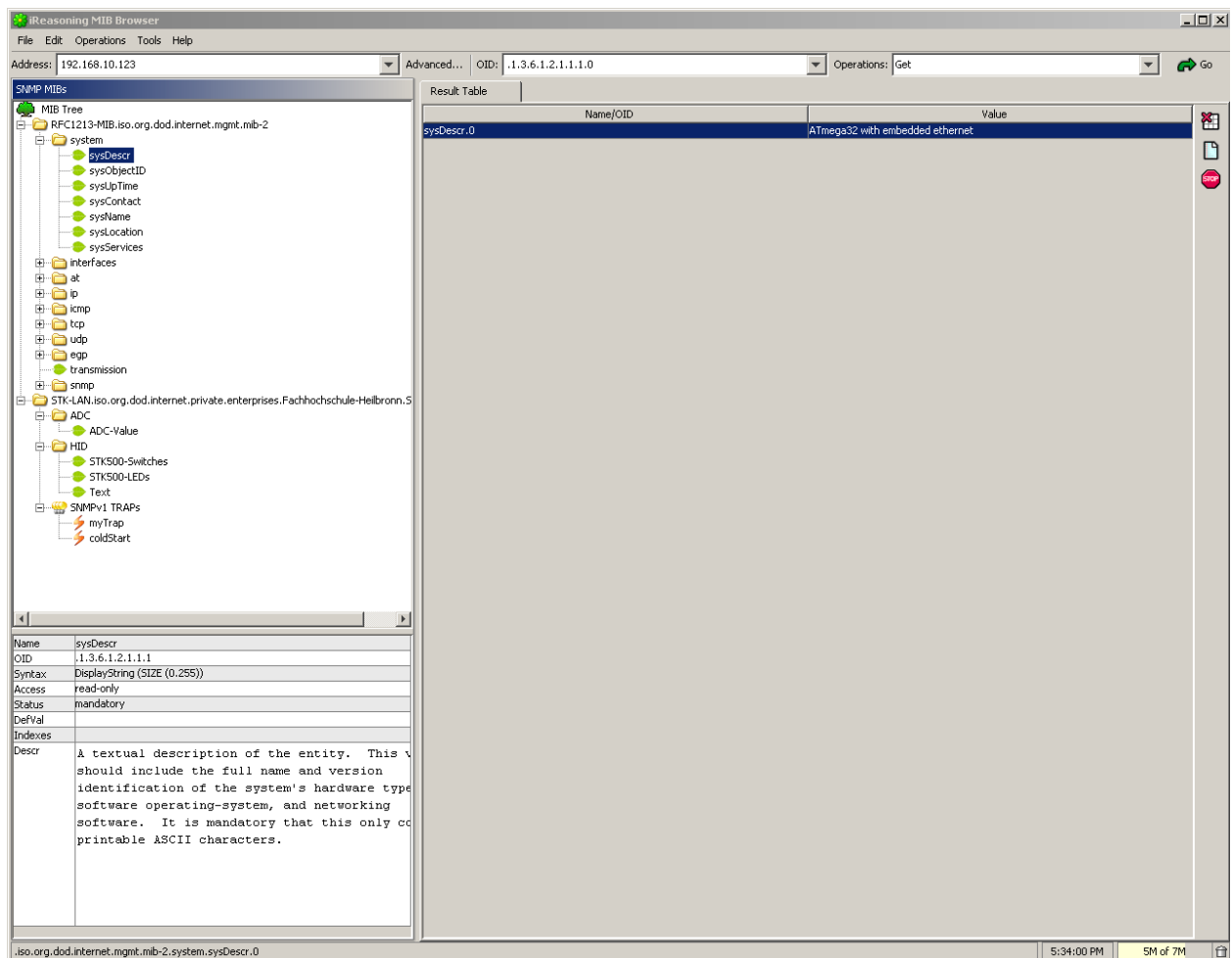


Abbildung 4.32.: Der erweiterte MIB-Tree nach dem Laden der MIB-Datei

KAPITEL 4. TEST UND INBETRIEBNAHME

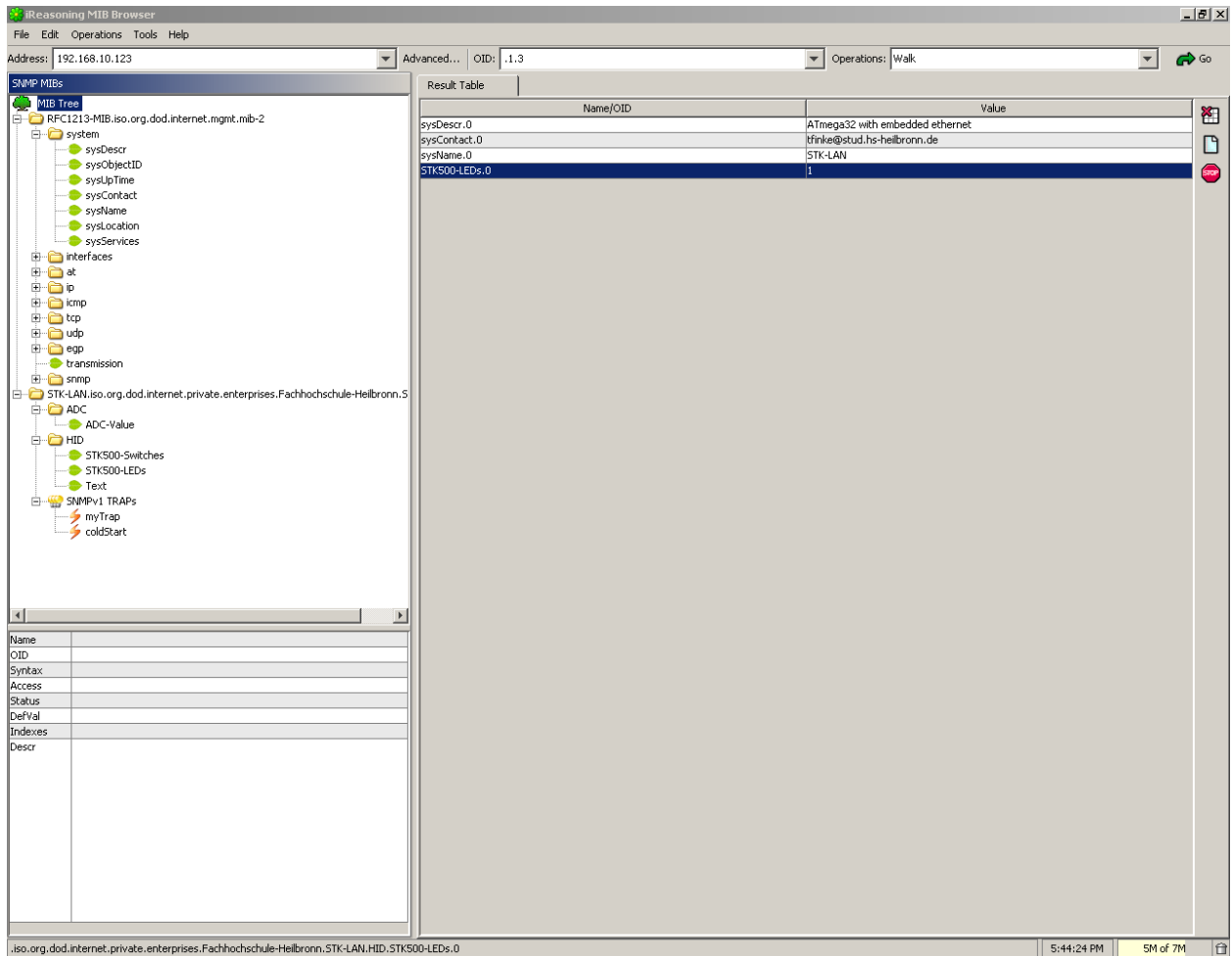


Abbildung 4.33.: Senden von Requests mit dem iReasoning MIB Browser

Über die Eingabefelder «OID» und «Operations» am oberen Rand der Programmoberfläche wird der gewünschte *SNMP-Request* eingestellt und mit einem Klick auf die Schaltfläche «Go» ausgeführt. Anschließend wird das Ergebnis der Anfrage in der «Result Table» (Abbildung 4.33) angezeigt.

4.2.2.3.2. Trap Receiver Der *Trap Receiver*, welcher sich über «Start - Programme - iReasoning - MIB Browser - Trap Receiver» aufrufen lässt, dient zum Empfangen und Anzeigen von *TRAP-Nachrichten*. Das Programm lauscht direkt nach dem Start auf Port 162 und zeigt alle empfangenen Nachrichten in einer Tabelle an. Durch das Markieren eines Eintrags in der Empfangstabelle werden weitere Informationen zum gewählten *TRAP* im unteren Informationsfenster angezeigt (Abbildung 4.34). Zum gezielten Generieren einer *TRAP-Nachricht* kann der Taster *SW3* auf dem *STK500* betätigt werden.

KAPITEL 4. TEST UND INBETRIEBNAHME

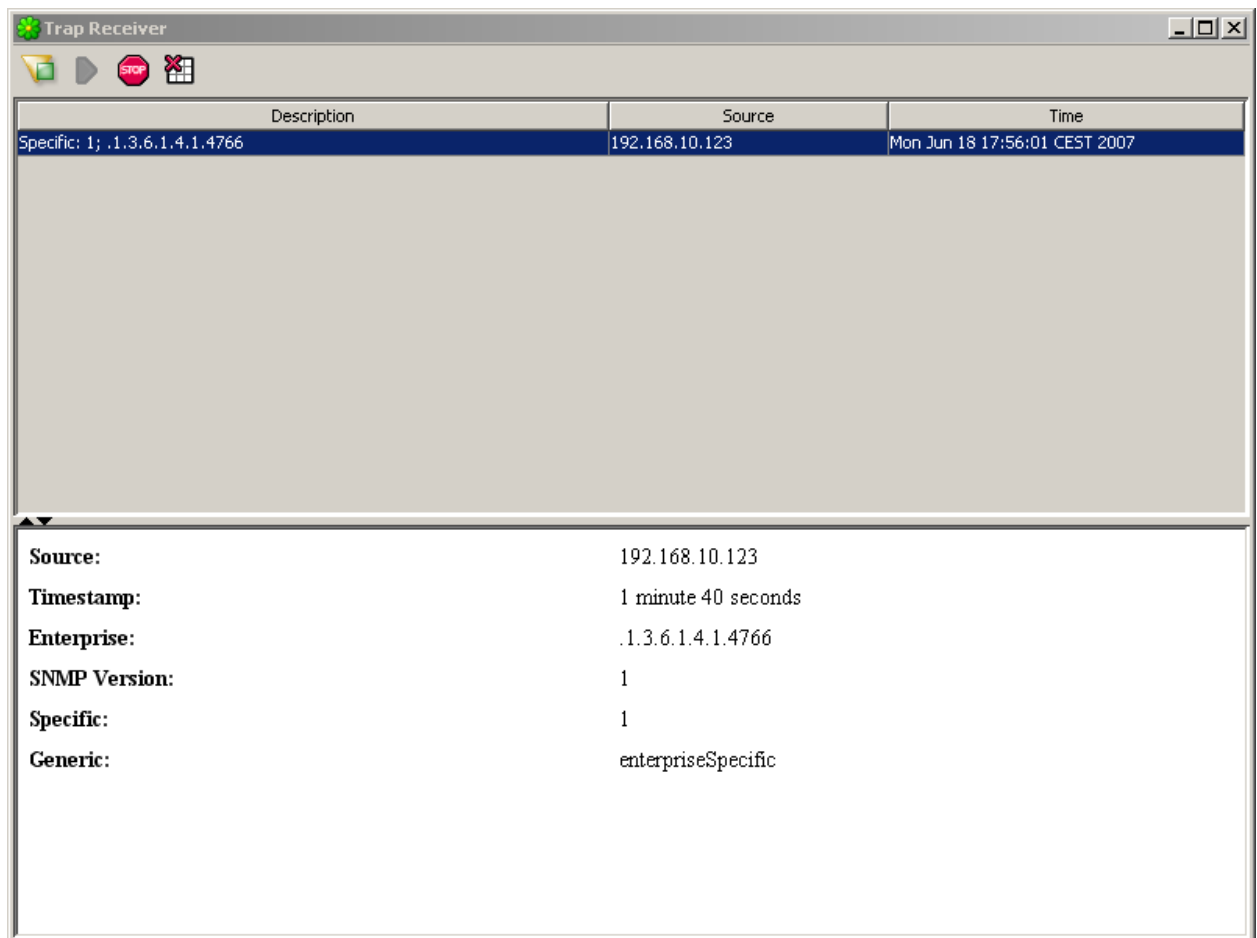


Abbildung 4.34.: Oberfläche des Trap Receivers

KAPITEL 4. TEST UND INBETRIEBNAHME

4.2.2.4. CurrPorts

Das Programm *CurrPorts* kann auf der Website von *NirSoft* [120] beziehungsweise dem Direktlink [119] kostenlos heruntergeladen werden. Mit Hilfe dieses Tools können alle aktuell verwendeten IP-Ports zusammen mit dem zugehörigen Prozess angezeigt werden. Weiterhin können gezielt Prozesse beendet und somit ein gewünschter Port wieder freigegeben werden. Im Rahmen dieser Diplomarbeit wurde dieses Tool des Öfteren dazu verwendet, den Prozess des *Trap Receivers* des *MG-SOFT MIB Browsers* zu beenden, da dieser den Port 162 blockiert hat und somit zum Beispiel der *iReasoning Trap Receiver* keine *TRAPS* empfangen konnte. Das Programm kann nach dem Download entpackt und direkt ausgeführt werden. Eine Installation ist nicht nötig. Die Programmoberfläche (Abbildung 4.35) ist sehr übersichtlich gehalten und somit selbsterklärend.

4.2.2.5. Wireshark

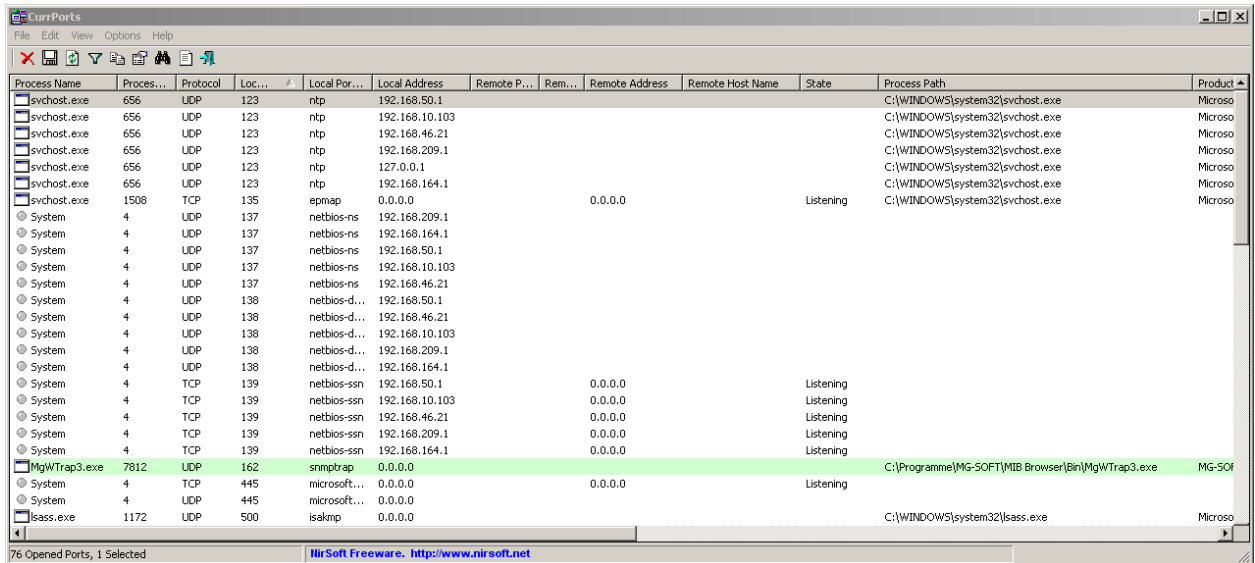
Das kostenlose Programm *Wireshark* [141] ist ein Netzwerk-Sniffer zur Analyse von Netzwerkverbindungen. Das Tool zeichnet den kompletten Datenverkehr einer bestimmten Netzwerkschnittstelle auf und stellt diesen in Form von Paketen übersichtlich dar. Zusätzlich können auf die empfangenen Daten einstellbare Filter angewandt werden, um nur diejenigen Daten anzuzeigen, welche von Interesse sind. Der dabei nötige hardwarenahe Zugriff auf das Netzwerkinterface wird bei *Wireshark* mit Hilfe von *WinPcap* realisiert.

Das Programm *Wireshark* wurde anfangs unter dem Namen *Ethereal* [58] von einem Team um *Gerald Combs* von der *Ethereal Software Inc.* entwickelt. Es unterliegt der *GNU General Public License* [39]. Nachdem *Gerald Combs* zu *CACE Technologies* [125] wechselte, startete er das neue Projekt namens *Wireshark*, welches auch unter der *GNU General Public License* verfügbar ist.

4.2.2.5.1. Konfiguration Nach der Installation wird das Programm über «Start - Programme - Wireshark - Wireshark» gestartet. Direkt nach dem Start erscheint die Programmoberfläche (Abbildung 4.36).

Bevor das eigentliche Aufzeichnen von Verbindungsdaten gestartet wird, wird zuerst das Standard-Netzwerkinterface festgelegt. Hierzu werden die Programmeinstellungen über das Menü «Edit» und den darin enthaltenen Menüpunkt «Preferences...» aufgerufen. In dem daraufhin erscheinenden Fenster (Abbildung 4.37) wird auf der linken Seite der Eintrag «Capture» und anschließend auf der rechten Seite das gewünschte «Default interface» gewählt. Mit einem anschließenden Klick auf die Schaltfläche «OK» werden die Einstellungen übernommen und das Einstellungsfenster geschlossen.

KAPITEL 4. TEST UND INBETRIEBNAHME



Process Name	Process ID	Protocol	Local Port	Local Address	Remote Port	Remote Address	Remote Host Name	State	Process Path	Product
svchost.exe	656	UDP	123	192.168.50.1					C:\WINDOWS\system32\svchost.exe	Microso
svchost.exe	656	UDP	123	192.168.10.103					C:\WINDOWS\system32\svchost.exe	Microso
svchost.exe	656	UDP	123	192.168.46.21					C:\WINDOWS\system32\svchost.exe	Microso
svchost.exe	656	UDP	123	192.168.209.1					C:\WINDOWS\system32\svchost.exe	Microso
svchost.exe	656	UDP	123	127.0.0.1					C:\WINDOWS\system32\svchost.exe	Microso
svchost.exe	656	UDP	123	192.168.164.1					C:\WINDOWS\system32\svchost.exe	Microso
svchost.exe	1508	TCP	135	0.0.0.0		0.0.0.0		Listening	C:\WINDOWS\system32\svchost.exe	Microso
System	4	UDP	137	192.168.209.1						
System	4	UDP	137	192.168.164.1						
System	4	UDP	137	192.168.50.1						
System	4	UDP	137	192.168.10.103						
System	4	UDP	137	192.168.46.21						
System	4	UDP	138	192.168.50.1						
System	4	UDP	138	192.168.46.21						
System	4	UDP	138	192.168.10.103						
System	4	UDP	138	192.168.209.1						
System	4	UDP	138	192.168.164.1						
System	4	TCP	139	192.168.50.1		0.0.0.0		Listening		
System	4	TCP	139	192.168.10.103		0.0.0.0		Listening		
System	4	TCP	139	192.168.46.21		0.0.0.0		Listening		
System	4	TCP	139	192.168.209.1		0.0.0.0		Listening		
System	4	TCP	139	192.168.164.1		0.0.0.0		Listening		
MgWTrap3.exe	7812	UDP	162	0.0.0.0					C:\Programme\MG-SOFT\WIB Browser\Bin\MgWTrap3.exe	MG-SOF
System	4	TCP	445	0.0.0.0		0.0.0.0		Listening		
System	4	UDP	445	0.0.0.0						
lsass.exe	1172	UDP	500	0.0.0.0					C:\WINDOWS\system32\lsass.exe	Microso

Abbildung 4.35.: Programmoberfläche von *CurrPorts*

4.2.2.5.2. Aufzeichnen von Daten Vor dem eigentlichen Aufzeichnen von Netzwerkdaten kann ein zusätzlicher Filter aktiviert werden. Bei den Aufzeichnungen im Rahmen dieser Diplomarbeit wurde meist ein Filter auf eine bestimmte *IP-Adresse* angewendet, da ansonsten sehr viele «uninteressante» Daten mit angezeigt wurden. Es ist mit dem Filter beispielsweise möglich, nur Pakete anzuzeigen, welche als Sender- oder Ziel-IP-Adresse einen bestimmten Wert enthalten. Während den Tests mit dem *STK-LAN* wurde dieser Filter auf die *IP-Adresse* «192.168.10.123» angewandt, somit konnten alle Daten ausgeblendet werden, welche nichts mit dem *STK-LAN* zu tun hatten.

Um die Aufzeichnung der Netzwerkpakete zu starten, genügt ein Klick auf den Eintrag «Start» im Menü «Capture». Anschließend werden alle ankommenden und abgehenden Daten auf dem Standard-Netzwerkinterface protokolliert und angezeigt.

Soll allerdings ein Filter angewandt oder beispielsweise vorübergehend ein anderes Netzwerkinterface verwendet werden, so wird im Menü «Capture» der Eintrag «Options...» gewählt. Im daraufhin erscheinenden Fenster, welches in Abbildung 4.38 zu sehen ist, kann im oberen Bereich das Netzwerkinterface für den aktuellen *Capture-Vorgang* ausgewählt werden. Im Feld «Capture Filter» kann beispielsweise durch die Angabe «host 192.168.10.123» ein Filter auf die *IP-Adresse* «192.168.10.123» gesetzt werden. Somit werden während der folgenden Aufzeichnung nur Pakete angezeigt, welche von dieser Adresse kommen oder an diese Adresse adressiert sind. Weitere mögliche Filter können durch einen Klick auf die Schaltfläche «Capture Filter:» angezeigt und ausgewählt werden. Ein anschließender Klick auf die Schaltfläche «Start» startet den *Capture-Vorgang*.

KAPITEL 4. TEST UND INBETRIEBNAHME

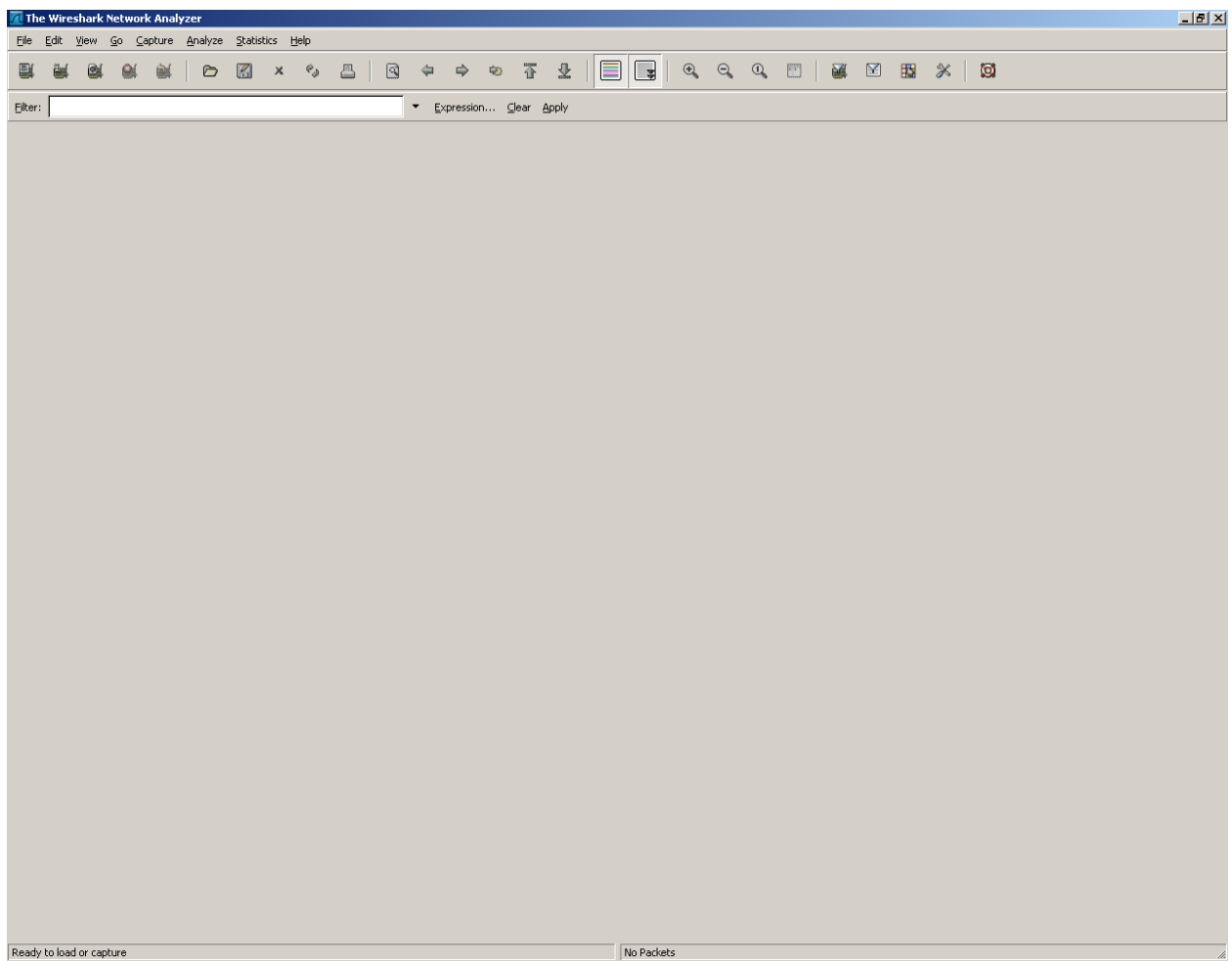


Abbildung 4.36.: Programmoberfläche von *Wireshark*

KAPITEL 4. TEST UND INBETRIEBNAHME

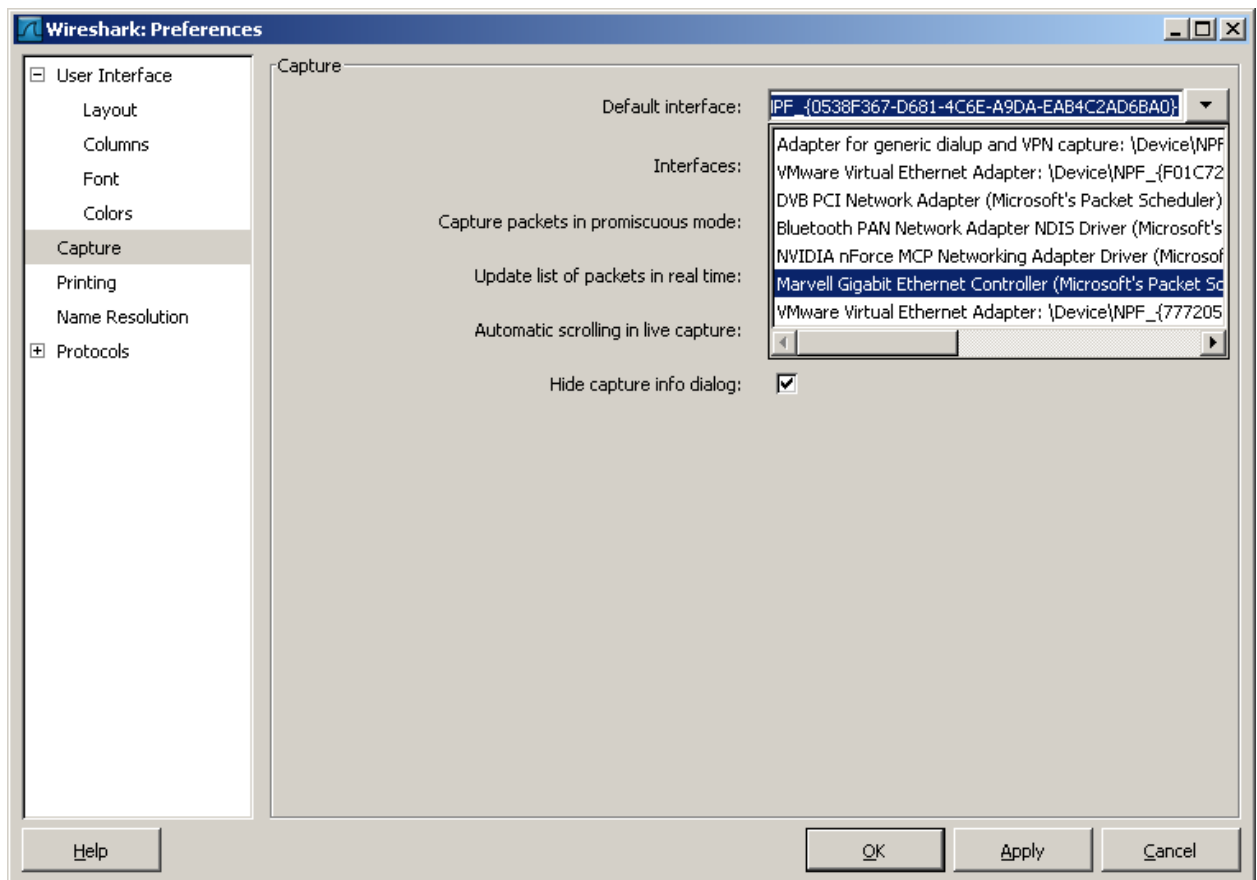


Abbildung 4.37.: Auswahl des Standard-Netzwerkinterfaces

KAPITEL 4. TEST UND INBETRIEBNAHME

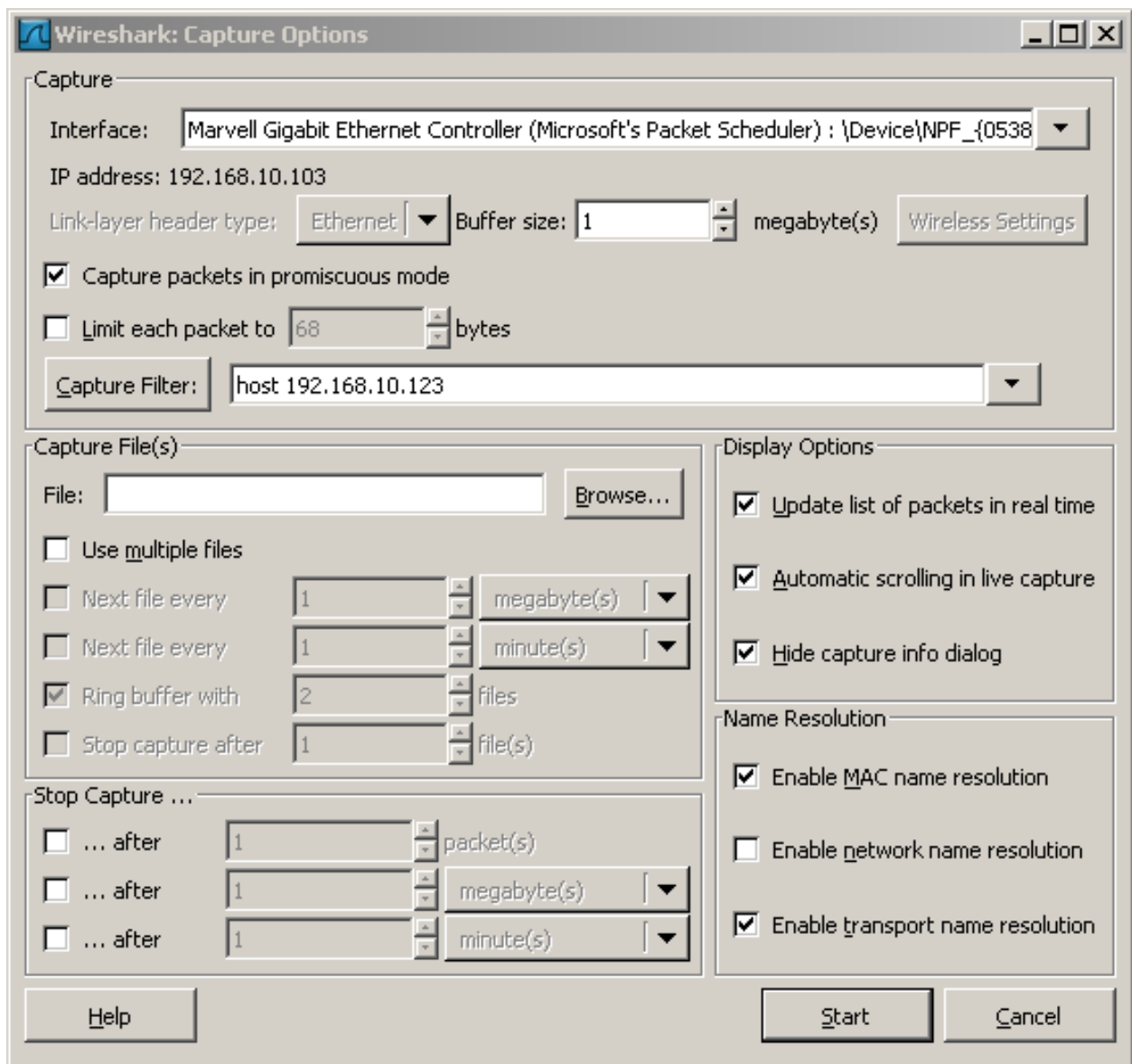


Abbildung 4.38.: Auswahl der Capture-Optionen

KAPITEL 4. TEST UND INBETRIEBNAHME

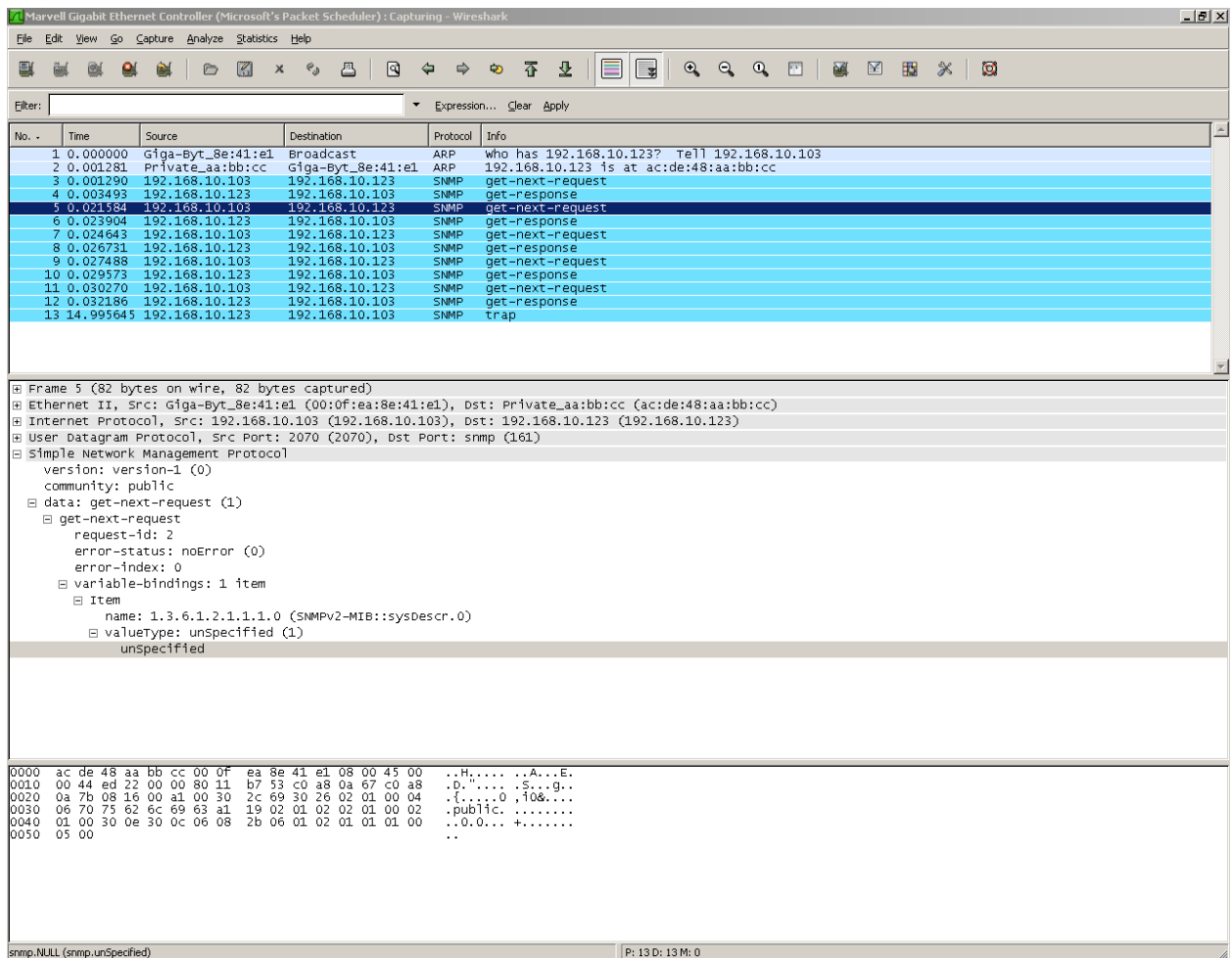


Abbildung 4.39.: Aktiver Capture-Vorgang

KAPITEL 4. TEST UND INBETRIEBNAHME

Während dem Capturevorgang werden die empfangenen Pakete mit einer fortlaufenden Nummer im oberen Bereich der Programmoberfläche aufgelistet, wie Abbildung 4.39 zeigt. Wird in dieser Paketauflistung ein bestimmtes Paket durch einen Mausklick markiert, so erscheinen in dem darunterliegenden Bereich die im Paket enthaltenen Daten in lesbarer Form. Im untersten Bereich der Programmoberfläche erscheinen zusätzlich alle Bytes des empfangenen und markierten Pakets in hexadezimaler Form.

Der Capture-Vorgang kann jederzeit über das Menü «Capture» und den darin enthaltenen Punkt «Stop» angehalten werden.

4.2.2.5.3. Analyse der Daten Nachdem nun die grundlegende Bedienung von *Wireshark* vorgestellt wurde, wird nun ein Netzwerkpaket analysiert. Für dieses Beispiel wird zuerst ein Capture-Vorgang mit einem Filter auf «host 192.168.10.123» gestartet. Anschließend wird mit dem Programm *SnmpTool* eine Anfrage in Form von «snmptool get 192.168.10.123 public .1.3.6.1.2.1.1.1.0» an das *STK-LAN* gestellt. Das *Request*- sowie das *Response-Paket* erscheinen daraufhin im Paketfenster von *Wireshark*. Der Capture-Vorgang kann an dieser Stelle angehalten werden, da die interessanten Pakete bereits aufgezeichnet wurden.

Um nun die *Response* des *STK-LAN* zu analysieren, wird in der Paketansicht das Paket mit der Info «get-response» markiert. Im mittleren Bereich der *Wireshark* Oberfläche kann nun, wie in Abbildung 4.40, die *Destination* des *Ethernetpakets* markiert werden. Im unteren Feld werden daraufhin die zu dieser Angabe gehörenden Bytes farblich markiert.

Um das komplette Datenpaket zu analysieren, werden die empfangenen Datenbytes betrachtet:

```
0000  00 0f ea 8e 41 e1 ac de 48 aa bb cc 08 00 45 00  ....A...H.....E.
0010  00 63 00 08 40 00 80 11 64 4f c0 a8 0a 7b c0 a8  .c...@....dO...{..
0020  0a 67 00 a1 08 95 00 4f 00 00 30 45 02 01 00 04  .g.....O..0E....
0030  06 70 75 62 6c 69 63 a2 38 02 01 01 02 01 00 02  .public.8.....
0040  01 00 30 2d 30 2b 06 08 2b 06 01 02 01 01 01 00  ..0-0+...+.....
0050  04 1f 41 54 6d 65 67 61 33 32 20 77 69 74 68 20  ..ATmega32 with
0060  65 6d 62 65 64 64 65 64 20 65 74 68 65 72 6e 65  embedded etherne
0070  74                                         t
```

An dieser Stelle können die Daten mit den Netzwerkgrundlagen aus Anhang A ausgewertet werden. Das vorliegende Datenpaket besteht aus einem *Ethernetframe*, welcher ein *IP-Paket* enthält. Im

KAPITEL 4. TEST UND INBETRIEBNAHME

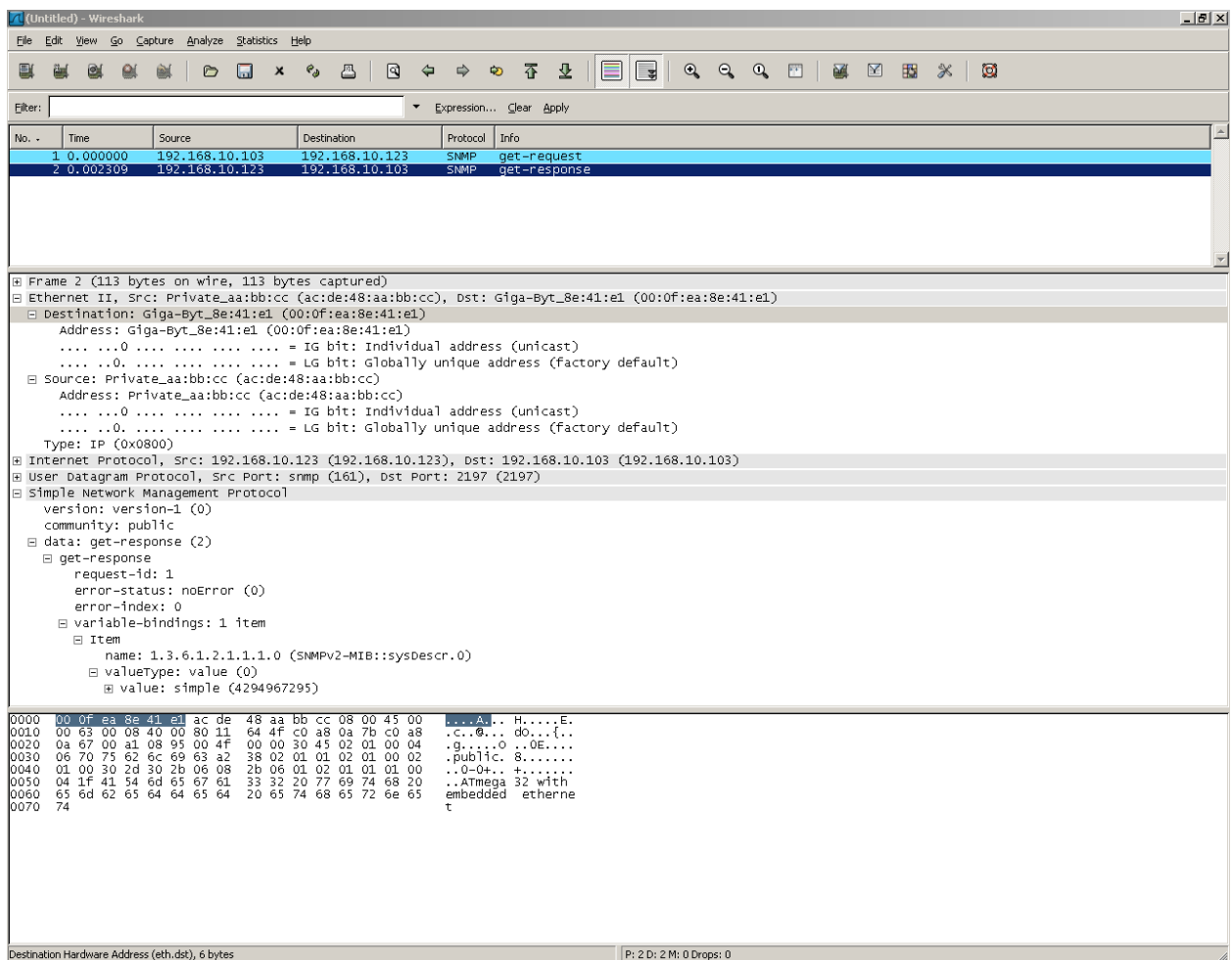


Abbildung 4.40.: Auswertung eines Datenpakets

KAPITEL 4. TEST UND INBETRIEBNAHME

Datenbytes	Bedeutung
0x45	IP-Version = IPv4, Header Length = 32 Bits · 5 = 160 Bits = 20 Bytes
0x00	Type Of Service = 0x00 (Es sind keine speziellen Informationen zum Steuern des Datenflusses vorhanden)
0x0063	Total Length = 99 Bytes (Komplette Länge des IP-Pakets)
0x0008	Fragment ID = 8
0x4000	DF = 1 (Do not fragment), Fragment Offset = 0
0x80	TTL = 128
0x11	Protocol = 17 (UDP)
0x644F	Header Checksum
0xC0A80A7B	Source IP Address = 192.168.10.123
0xC0A80A67	Destination IP Address = 192.168.10.103

Tabelle 4.1.: Auswertung des IP-Headers

Datenbereich des *IP-Pakets* liegt wiederum ein *UDP-Paket*, welches das *SNMP-Nachrichtenpaket* enthält.

Zuerst wird der *Header* des *Ethernetframes* betrachtet:

```
0000  00 0f ea 8e 41 e1 ac de 48 aa bb cc 08 00          ....A...H.....
```

Am Anfang steht die *Ziel-MAC-Adresse* «00 0F EA 8E 41 E1», also die *MAC-Adresse* des Computers, an welchen die Antwort gerichtet ist. Danach folgt die *Quell-MAC-Adresse* «AC DE 48 AA BB CC», also die Adresse des *STK-LAN*. Den beiden *MAC-Adressen* folgt schließlich noch das *Typ-Feld*, welches mit der 0x0800 ein nachfolgendes *IPv4-Paket* signalisiert.

Dem *Header* des *Ethernetframes* folgt der *IP-Header*. Dieser kann wie in Tabelle 4.1 gezeigt aufgeschlüsselt werden:

```
0000                                     45 00          E.
0010  00 63 00 08 40 00 80 11 64 4f c0 a8 0a 7b c0 a8  .c...@....dO...{..
0020  0a 67                                           .g
```


KAPITEL 4. TEST UND INBETRIEBNAHME

Darauf folgt der *UDP-Header* mit den Daten:

```
0020      00 a1 08 95 00 4F 00 00      .....O..
```

Der Port auf der Senderseite ist somit 161 ($0x00A1$), der Port auf Empfängerseite ist 2197 ($0x0895$). Die Länge des kompletten *UDP-Pakets* beträgt 79 Bytes ($0x004F$), die Checksumme ist nicht angegeben ($0x0000$). Im Datenbereich des *UDP-Pakets* folgt nun die *SNMP-Nachricht*, welche wie in Tabelle 4.2 gezeigt zu deuten ist.

```
0020      30 45 02 01 00 04      0E....
0030  06 70 75 62 6c 69 63 a2 38 02 01 01 02 01 00 02  .public.8.....
0040  01 00 30 2d 30 2b 06 08 2b 06 01 02 01 01 01 00  ..0-0+...+.....
0050  04 1f 41 54 6d 65 67 61 33 32 20 77 69 74 68 20  ..ATmega32 with
0060  65 6d 62 65 64 64 65 64 20 65 74 68 65 72 6e 65  embedded etherne
0070  74      t
```

An dem Beispiel ist schön zu sehen, dass die *Präambel* sowie die *CRC-Prüfsumme* des *Ethernet-frames* von Wireshark nicht angezeigt werden.

KAPITEL 4. TEST UND INBETRIEBNAHME

Datenbyte	Bedeutung
0x30	Anfang einer Sequenz
0x45	Anzahl der folgenden Bytes des SNMP-Pakets: 69 Bytes
0x02 0x01 0x00	SNMP-Version = SNMPv1
0x04 0x06 0x70 0x75 0x62 0x6C 0x69 0x63	Community-String = «public»
0xA2	Nachrichtentyp = RESPONSE
0x38	Folgebytes: 56 Bytes
0x02 0x01 0x01	Request-ID = 1
0x02 0x01 0x00	Error Status = 0
0x02 0x01 0x00	Error Index = 0
0x30	Anfang einer Sequenz
0x2D	Folgebytes: 45 Bytes
0x30	Anfang einer Sequenz
0x2B	Folgebytes: 43 Bytes
0x06 0x08 0x2B 0x06 0x01 0x02 0x01 0x01 0x01 0x00	OID: .1.3..1.2.1.1.1.0
0x04 0x1F 0x41 0x54 0x6D 0x65 0x67 0x61 0x33 0x32 0x20 0x77 0x69 0x74 0x68 0x20 0x65 0x6D 0x62 0x65 0x64 0x64 0x65 0x64 0x20 0x65 0x74 0x68 0x65 0x72 0x6E 0x65 0x74	OCTET STRING: «ATmega32 with embedded ethernet»

Tabelle 4.2.: Auswertung der SNMP-Nachricht

4.2.3. HTTP

Neben dem bereits vorgestellten Netzwerk-Sniffer *Wireshark* werden zum Testen der *HTTP-Kommunikation* die gängigen Webbrowser verwendet. Die vom *STK500* generierte und über das *STK-LAN* versendete Website wurde mit *Microsoft Internet Explorer 7.0.5730.11* (Abbildung 4.41), *Mozilla Firefox 2.0.0.6*, *Netscape Navigator 7.1*, *Opera 9.22* und *Konqueror 3.5.5* dargestellt und getestet.

KAPITEL 4. TEST UND INBETRIEBNAHME

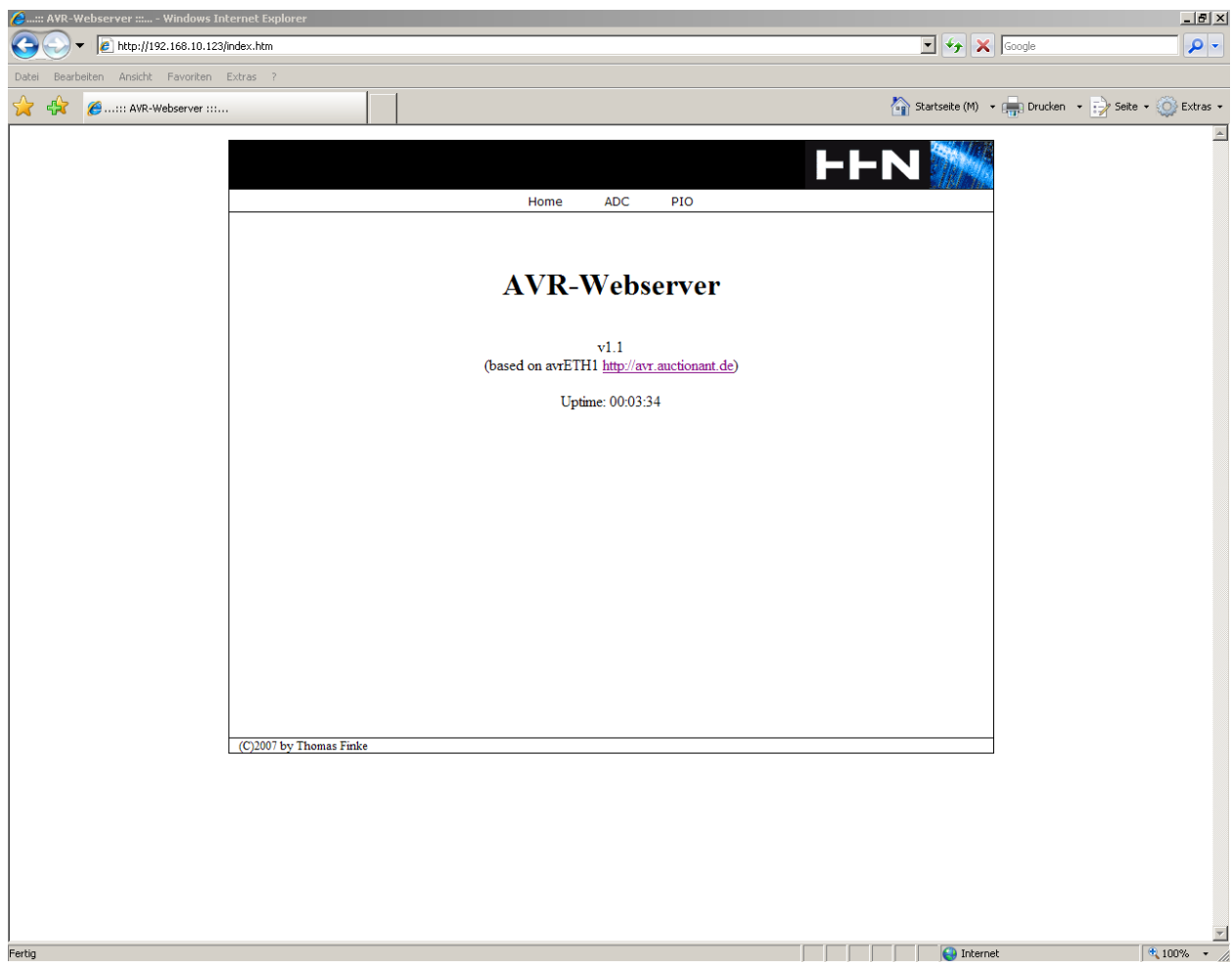


Abbildung 4.41.: Darstellung mit Microsoft Internet Explorer 7.0.5730.11

4.2.4. Terminalserver

Zur Kommunikation mit dem *Terminalserver* wird das *Hercules SETUP utility* [48] der Firma *HW group s.r.o.* [47] verwendet. Dieses Programm wurde ursprünglich dazu entwickelt, die Produkte der Firma zu konfigurieren. Inzwischen ist es als Freeware verfügbar. Dieses Tool bietet sich an dieser Stelle besonders als *Terminal-Client* an, da es sowohl die Kommunikation über *RS232* wie auch über *UDP* unterstützt. Für eine reine *RS232-Kommunikation* kann auch das *Microsoft HyperTerminal* oder *HTerm* verwendet werden.

Nachdem das Programm heruntergeladen und entpackt wurde, kann es über die Datei *Hercules.exe* gestartet werden. Nach dem Start erscheint die in Abbildung 4.42 gezeigte Oberfläche.

4.2.4.1. Kommunikation über RS232

Für eine Kommunikation über die serielle Schnittstelle wird hier die Registerkarte «Serial» (Abbildung 4.43) ausgewählt. Bevor die Verbindung zum *STK500* aufgebaut werden kann, muss *Hercules* konfiguriert werden. Hierzu wird auf der rechten Seite unter «Name» die verwendete Schnittstelle am PC angegeben. Im Feld darunter ist die *Baudrate* auf den Wert 9600 einzustellen. Nachdem der *Handshake* über die Option «OFF» deaktiviert wurde, kann die Verbindung durch einen Klick auf die Schaltfläche «Open» aufgebaut werden.

Im Feld «Received/Sent data» werden von nun an alle gesendeten und empfangenen Daten angezeigt. Zum Senden von Daten stehen die drei Felder im unteren Teil des Fensters zur Verfügung. Hier können mehrere Befehle eingetragen und durch einen Klick auf die entsprechende «Send»-Schaltfläche gesendet werden. Bei der Eingabe der Befehle ist darauf zu achten, dass jeder Befehl mit einem *Carriage Return* der Form «<CR>» abgeschlossen ist, da der Terminalserver den Befehl sonst nicht entgegennimmt.

4.2.4.2. Kommunikation über UDP

Soll über das Netzwerk per *UDP* mit dem Terminalserver kommuniziert werden, wird die Registerkarte «UDP» ausgewählt (Abbildung 4.44).

Auf der Registerkarte müssen zuerst die Verbindungsdaten eingetragen werden; als «Module IP» die *IP-Adresse* des *STK-LAN* und als «Port» 100. Der Port, welcher vom *Client* zum Empfangen von Nachrichten verwendet wird, kann über «Local port» angegeben werden. Anschließend wird der *Client* über einen Klick auf die Schaltfläche «Listen» gestartet.

KAPITEL 4. TEST UND INBETRIEBNAHME

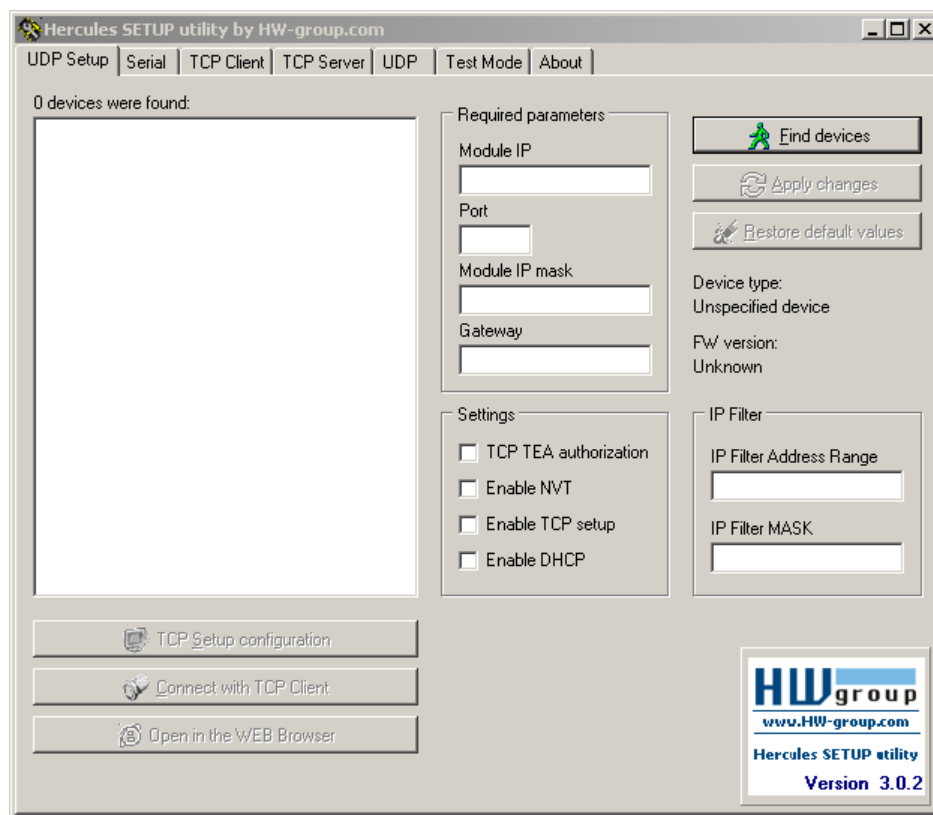


Abbildung 4.42.: Oberfläche des Hercules SETUP utility

KAPITEL 4. TEST UND INBETRIEBNAHME

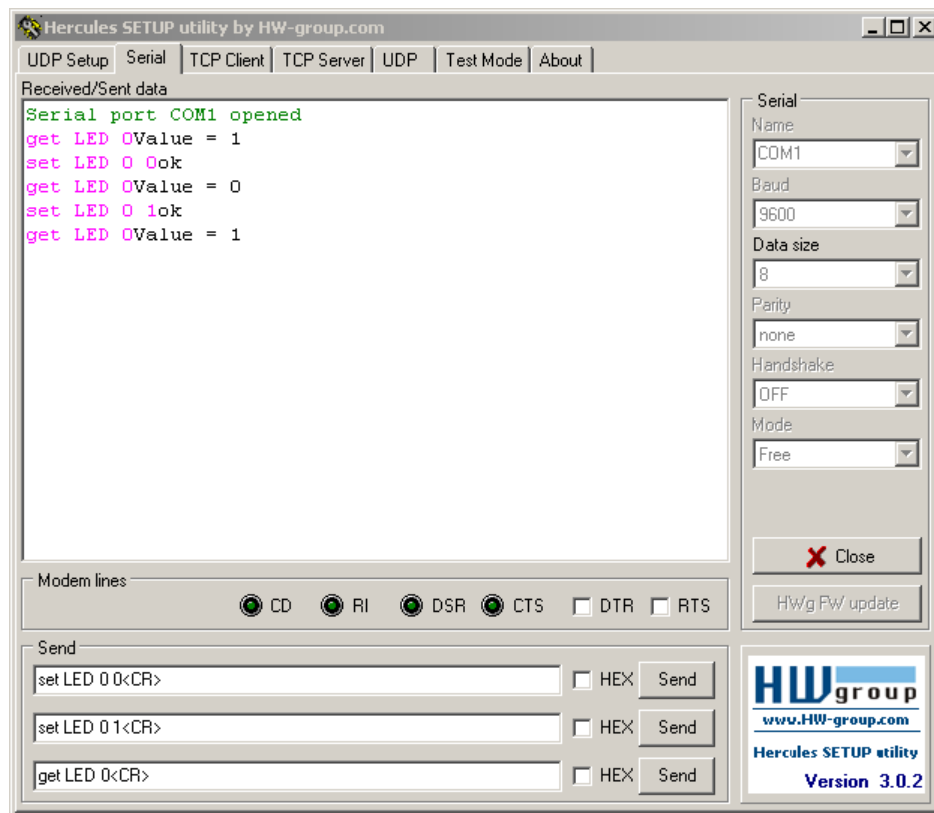


Abbildung 4.43.: Registerkarte für die Kommunikation über RS232

KAPITEL 4. TEST UND INBETRIEBNAHME

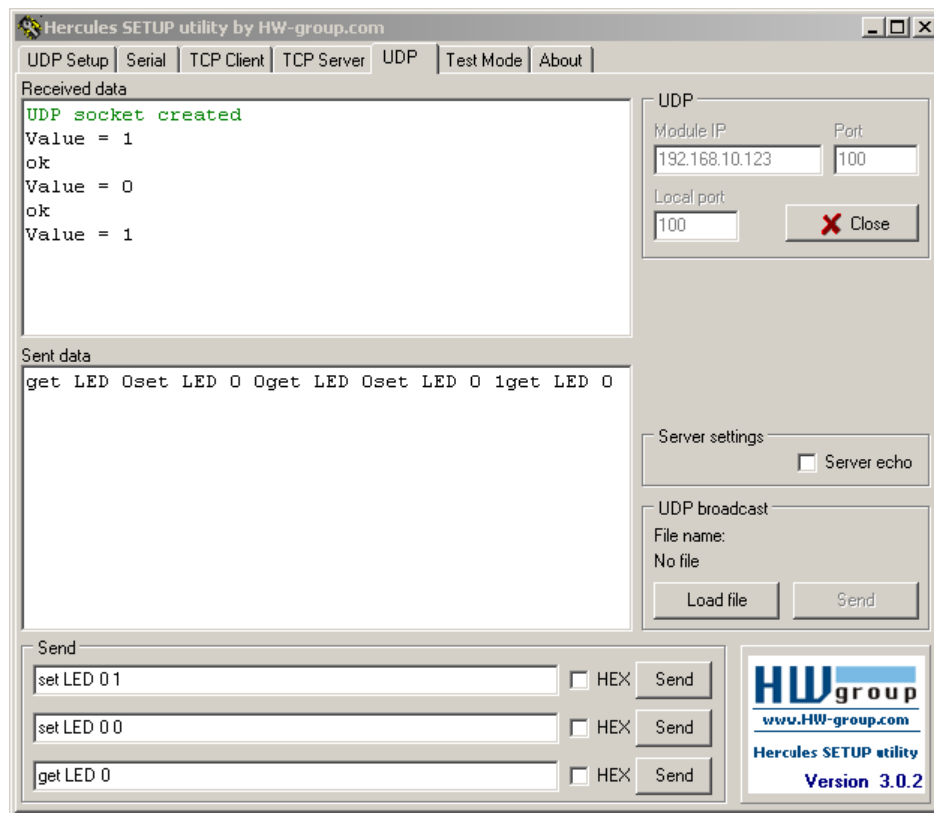


Abbildung 4.44.: Registerkarte für die Kommunikation über UDP

KAPITEL 4. TEST UND INBETRIEBNAHME

Im Fenster «Received data» werden von nun an alle über *UDP* empfangenen Daten angezeigt. Das Fenster «Sent data» zeigt dagegen die vom *Client* gesendeten Daten. Wie bei der Kommunikation über *RS232* können auch hier wieder drei Felder mit Befehlen belegt werden. Diese Befehle werden über einen Klick auf die entsprechende Schaltfläche «Send» gesendet. Bei der Verwendung der *UDP-Verbindung* ist das abschließende *Carriage Return* am Zeilenende allerdings nicht nötig, da die Nachrichtenpakete bereits durch das *UDP-Paket* begrenzt sind.

KAPITEL 4. TEST UND INBETRIEBNAHME

5. Resümee

Durch diese Diplomarbeit sollte eine Entwicklungsumgebung geschaffen werden, welche den Studierenden einen einfachen Einstieg in die Implementierung von Netzwerkprotokollen in einem Mikrocontroller gestattet.

Durch das entwickelte Erweiterungsboard *STK-LAN* kann das den Studierenden bekannte *STK500* um zwei Ethernetports erweitert werden. Der zur Ansteuerung verwendete Netzwerkstack *avrETH1* ist sehr übersichtlich gestaltet und kann leicht an die eigenen Anforderungen angepasst werden.

Um den Einstieg in die gängigen Netzwerkprotokolle zu erleichtern, wurde im Mikrocontrollerprogramm ein *Terminalserver* integriert. Hierdurch kann im Rahmen einer Vorlesung eine einfache *Terminalverbindung* über die *serielle Schnittstelle* mit dem *STK500* hergestellt werden. Nachdem sich die Studierenden mit dieser Verbindung vertraut gemacht haben, kann anstatt der *seriellen Schnittstelle* eine *UDP-Verbindung* verwendet werden. Für diese kann PC-seitig das selbe *Terminalprogramm* verwendet werden. Serverseitig werden die selben Befehle und Funktionen verwendet wie zuvor mit der *RS232-Verbindung*. Somit ist der Übergang einer einfachen seriellen Verbindung zu einer Netzwerkverbindung leicht nachvollziehbar. Sobald sich die Studierenden wiederum mit dieser Verbindungstechnologie vertraut gemacht haben, kann gezeigt werden, dass nun anstatt der selbst definierten Befehle auch standardkonforme Netzwerkprotokolle wie *SNMP* verwendet werden können. An dieser Stelle angekommen, haben die Studierenden ein grundlegendes Verständnis der verwendeten Schnittstellen und Protokolle erlangt und können sich beispielsweise mit dem integrierten *Webserver* beschäftigen oder auf tiefer liegenderer Ebene im *OSI-Schichtenmodell* einen kleinen *Router* mit den beiden *Ethernetports* des *STK-LAN* realisieren. Über die zahlreichen *Taster* und *Leuchtdioden* auf dem *STK500* steht ein gutes *Human Interface* zur Verfügung. Durch die Debugmöglichkeit des *Mikrocontrollers* wird die Softwareentwicklung stark vereinfacht.

Die Grundlagen der Netzwerkprotokolle *IP*, *ICMP*, *ARP*, *UDP*, *TCP*, *HTTP* und *SNMP* sind im Anhang sehr detailliert ausgeführt. Hierdurch steht ein gutes Nachschlagewerk für die verwendeten Netzwerkprotokolle zur Verfügung.

KAPITEL 5. RESÜMEE

Bei der verwendeten Software wurde darauf geachtet, dass es sich möglichst um *Freeware* oder *Open Source* handelt. Somit fallen bei der Anwendung im Labor, neben dem Preis für die Bauteile des *STK-LAN*, keine weiteren Kosten an.

An dieser Stelle soll auch darauf hingewiesen werden, dass der komplette Quellcode des Mikrocontrollerprogramms der *GNU General Public License (GPL)* unterliegt und das Programm somit nur mit dem kompletten Quellcode weitergegeben werden darf.

Das Programm wurde vom Speicherbedarf an einen *ATmega32* angepasst, dessen *Static Random Access Memory (SRAM)* in der momentanen Ausführung nur noch sehr wenig freien Platz bietet. Bei Bedarf kann auf einen *ATmega644* umgestiegen werden. Dieser Mikrocontroller ist pincompatibel und bietet die doppelte Speichermenge an *Flash* und *SRAM*. Allerdings lässt sich der größere Controllertyp nur noch mit einem originalen *JTAGICE mkII* debuggen. Für den kleineren *ATmega32* reicht auch ein günstiger Nachbau eines JTAG-Programmiersadapters.

Durch die sehr detaillierten Erläuterungen der Netzwerkprotokolle werden durch diese Diplomarbeit teilweise auch Schwachstellen der Protokolle aufgezeigt. Die Ergebnisse dieser Arbeit sind jedoch keineswegs als Hilfe für Angriffe auf fremde Netzwerke gedacht.

Abschließend kann gesagt werden, dass die Aufgabenstellung dieser Diplomarbeit vollständig gelöst wurde und für das Mikrocomputerlabor der *Hochschule Heilbronn* nun eine geeignete Plattform zur Einführung in das Thema «Implementierung von Internet-Protokollen für Remote-Control-Applikationen auf einem Mikrocontroller» zur Verfügung steht.

A. Netzwerkgrundlagen

Bei einem Netzwerk handelt es sich um einen Zusammenschluss verschiedenartiger Systeme zur Datenkommunikation. Die Kommunikation wird durch das *OSI-Referenzmodell* (*Open System Interconnection (OSI)*) grob strukturiert. Die eigentliche Kommunikation ist hierbei in die sieben folgenden *OSI-Schichten* unterteilt. Jede Schicht stellt der darüberliegenden Schicht Dienste zur Verfügung und greift selbst auf die Dienste der unmittelbar darunterliegenden Schicht zurück (Abbildung A.1).

OSI-7-Layer-Model (Open Systems Interconnection Reference Model)

Begriffe: Englisch - Deutsch

1 Application Layer	- Anwendungsschicht
2 Presentation Layer	- Darstellungsschicht
3 Session Layer	- Sitzungs- bzw. Kommunikationsschicht
4 Transport Layer	- Transportschicht
5 Network Layer	- Netzwerk- bzw. Vermittlungsschicht
6 Data Link Layer	- Sicherungsschicht
7 Physical Layer	- Bitübertragungsschicht

PC im Netzwerk
A

W <http://www.wikipedia.org>



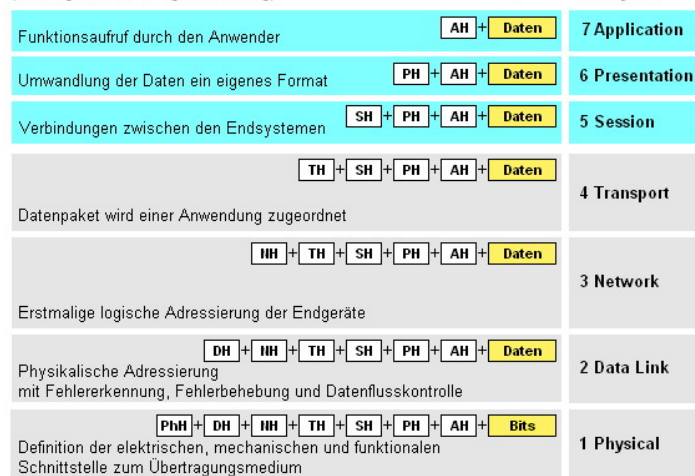
Der Benutzer empfängt lediglich die Antwort des Servers ("wikipedia.org"-Startseite). Im Allgemeinen bekommt er von der Schachtelung seines Seitenaufrufs durch die Ebenen seines PCs (abwärts) und vom Parsen der Antwort des Servers zurück durch die Ebenen seines PCs (aufwärts) nichts mit!

Server schickt die entsprechenden Daten über die selbe Methode zurück. (s.u.)

Server im Netzwerk
A



Zusammenbau des Pakets: (Package Assembling/Formatting)



Zusammensetzung der Abkürzungen oben:
Anfangsbuchstabe der Schicht und "H" für Header.
z.B. Application Header = AH

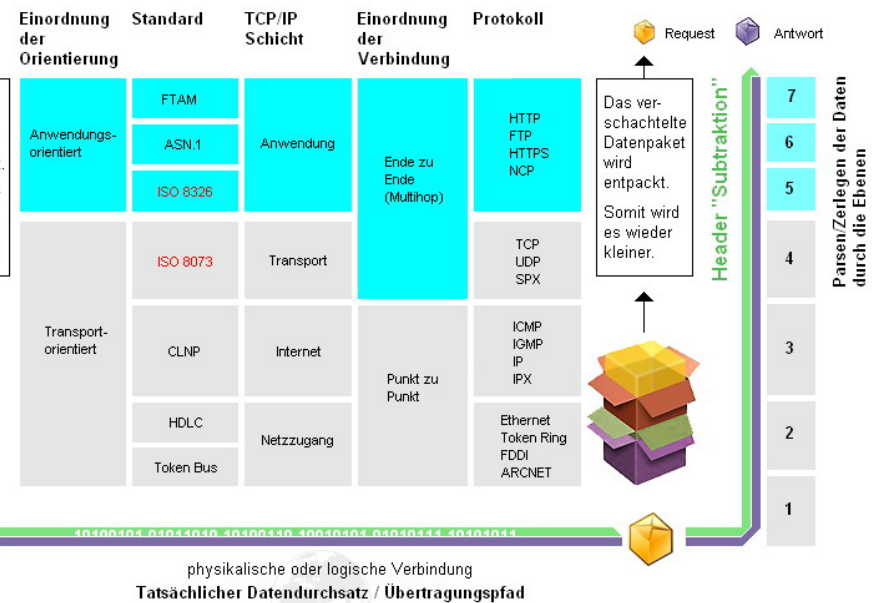


Abbildung A.1.: Kommunikation im OSI-Modell [123]

ANHANG A. NETZWERKGRUNDLAGEN

- Schicht 7 - Anwendungsschicht

Die Anwendungsschicht ist die oberste Ebene und stellt den eigentlichen Anwendungen die benötigten Funktionen, wie zum Beispiel das *File Transfer Protocol (FTP)*, *Telnet*, das *Simple Mail Transfer Protocol (SMTP)* oder *HTTP*, zur Verfügung.

- Schicht 6 - Darstellungsschicht

Da in einem Netzwerk unterschiedliche Rechnerarchitekturen zum Einsatz kommen können, ist es nötig, die zu übertragenden Daten an ein genormtes Datenformat anzupassen. Diese Aufgabe übernimmt die Darstellungsschicht des *OSI-Modells*. Sollen zum Beispiel *ASCII-Zeichen* über das Netzwerk übertragen werden, so kann diese Schicht das *ASCII-Format* zuerst in ein für alle Systeme verständliches Datenformat, wie beispielsweise *Abstract Syntax Notation One (ASN.1)*, umwandeln. Zusätzlich werden von dieser *OSI-Schicht* auch Datenkompressionen und Verschlüsselungen übernommen.

- Schicht 5 - Sitzungsschicht

Diese Schicht stellt Dienste für einen organisierten und synchronisierten Datenaustausch zur Verfügung. Hierzu werden Checkpoints verwendet, an welchen die Kommunikation wieder aufgenommen werden kann, nachdem ein Fehler in der Verbindung aufgetreten ist. Somit kann die Datenübertragung nach einem Fehler am letzten Checkpoint fortgesetzt werden, anstatt die komplette Datenübertragung neu zu beginnen.

- Schicht 4 - Transportschicht

Die Aufgabe der Transportschicht liegt in der Segmentierung von Datenpaketen. Diese Schicht stellt die unterste Ebene einer vollständigen Kommunikation zwischen Sender und Empfänger zur Verfügung. Die anwendungsorientierten Schichten 5 bis 7 erhalten somit einen genormten Zugriff auf die Daten, unabhängig von den Eigenschaften des Netzwerks.

- Schicht 3 - Vermittlungsschicht

Schicht drei sorgt für die Weiterleitung von Datenpaketen. Die Datenübertragung muss über das ganze Netzwerk hinweg möglich sein. Hierbei sorgt die Vermittlungsschicht für das nötige Routing zwischen den Netzknoten. Da eine direkte Kommunikation oft nicht möglich ist, müssen Pakete von Netzknoten, die auf dem Weg liegen, weitergeleitet werden. Diese weitergeleiteten Pakete gelangen dabei nicht in höhere *OSI-Schichten*, sondern werden nur mit einem neuen Zwischenziel versehen und weitergeleitet.

ANHANG A. NETZWERKGRUNDLAGEN

Die wichtigsten Aufgaben der Vermittlungsschicht sind die Aktualisierung von Routingtabellen, die Flusskontrolle sowie die Verwaltung der Netzwerkadressen.

- Schicht 2 - Sicherungsschicht

Diese Schicht hat die Aufgabe, eine sichere und weitestgehend fehlerfreie Übertragung zu gewährleisten. Weiterhin wird durch sie auch der Zugriff auf das Übertragungsmedium geregelt. Die Sicherungsschicht teilt den Datenstrom in Blöcke auf und fügt Folgenummern und Prüfsummen hinzu. Somit können verfälschte oder verloren gegangene Blöcke rekonstruiert oder neu angefordert werden.

- Schicht 1 - Bitübertragungsschicht

Hier wird die Datenübertragung auf der Ebene physikalischer Signale, wie zum Beispiel elektrischer oder optischer Impulse, geregelt. Wenn die Daten gemultiplext über das Übertragungsmedium übertragen werden, so wird dieses Multiplexen in der Bitübertragungsschicht realisiert.

In dieser Schicht wird auch definiert wie die Zustände eines Bits auf dem Übertragungsmedium realisiert sind.

Das bisher vorgestellte *OSI-Referenzmodell* ist eher theoretisch orientiert und die Organisation der einzelnen Schichten ist sehr gut durchdacht.

Neben diesem Modell existiert noch das *TCP/IP-Referenzmodell*, welches nach den beiden primären Protokollen *TCP* und *IP* bezeichnet ist und dessen Ursprünge mehr in der praktischen Anwendung liegen. Daher sind die Aufgaben der einzelnen Schichten nicht so exakt abgegrenzt wie beim *OSI-Modell*.

Im direkten Vergleich der beiden Modelle ist zu erkennen, dass das *TCP/IP-Referenzmodell* nur aus vier Schichten besteht und nicht wie das *OSI-Referenzmodell* aus sieben. Der Zusammenhang zwischen diesen beiden Modellen kann Abbildung A.2 entnommen werden.

Im Folgenden wird das *TCP/IP-Referenzmodell* dazu verwendet, um die Positionen bestimmter Protokolle im Netzwerk-Protokollstapel zu verdeutlichen. Das *TCP/IP-Modell* bietet dabei den Vorteil, dass aus ihm der Zusammenhang der einzelnen Protokolle zueinander deutlicher hervorgeht.

ANHANG A. NETZWERKGRUNDLAGEN

OSI		TCP/IP
Anwendungsschicht		Anwendungsschicht
Darstellungsschicht		
Sitzungsschicht		
Transportschicht		Transportschicht
Vermittlungsschicht		Internetschicht
Sicherungsschicht		Netzzugangsschicht
Bitübertragungsschicht		

Abbildung A.2.: Vergleich des OSI-Modells mit dem TCP/IP-Modell

Anwendungsschicht		HTTP	SNMP	
Transportschicht		TCP	UDP	
Internetschicht	ICMP			
Internetschicht	IP			ARP
Netzzugangsschicht	Ethernet			

Tabelle A.1.: Position von Ethernet im TCP/IP-Referenzmodell

A.1. Ethernet

Bei Ethernet handelt es sich um eine kabelgebundene Netzwerktechnologie nach dem *IEEE 802.3*-Standard [77] (*Institute of Electrical and Electronics Engineers (IEEE)*). Die Daten werden dabei in genormten Datenpaketen übertragen. Verwendung findet das Ethernet in LANs (Local Area Networks) sowie im Internet.

Ethernet beschreibt dabei die Netzkabel/-stecker und legt fest wie einzelne Bits übertragen werden. Somit werden durch Ethernet die *OSI-Schichten* 1 «Bitübertragungsschicht» und 2 «Sicherungsschicht» abgebildet. Der Zugriff auf das Übertragungsmedium wird durch *Carrier Sense Multiple Access with Collision Detection (CSMA/CD)* gesteuert. Dabei wartet ein Netzwerkteilnehmer, welcher senden möchte, solange, bis keine Kommunikation mehr auf dem Übertragungsmedium stattfindet. Danach fängt das Gerät an zu senden. Sollte zu diesem Zeitpunkt ein zweites Gerät ver-

ANHANG A. NETZWERKGRUNDLAGEN

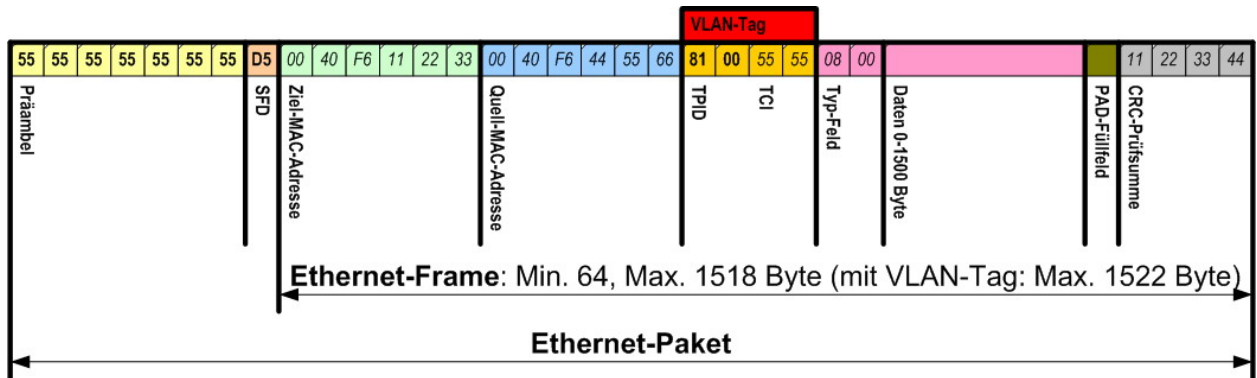


Abbildung A.3.: Ethernet-II-Frameformat aus IEEE 802.3, inklusive VLAN Tag aus IEEE 802.1Q [136]

suchen zu senden, so gibt es eine Kollision, beide Geräte stoppen den Sendevorgang sofort und warten eine zufällige Zeit, bevor sie erneut versuchen zu senden.

Die eigentlichen Daten werden dabei in einem sogenannten Ethernet-Frame übertragen, welcher den Aufbau nach Abbildung A.3 hat.

Die Übertragung der Daten erfolgt mit dem *Least Significant Bit (LSB)* zuerst, beginnend mit dem höchstwertigsten Byte, dies entspricht dem Format «Little Endian». Die CRC-Checksumme am Ende des Frames wird allerdings mit dem *Most Significant Bit (MSB)* zuerst übertragen, somit im Format «Big Endian».

A.1.1. Präambel

Dem eigentlichen *Ethernetframe* geht eine Präambel sowie ein *Start Frame Delimiter (SFD)* voran. Diese Bytes wurden in älteren Netzwerken dazu verwendet, die Netzwerkteilnehmer auf die Datenübertragung einzusynchronisieren. In modernen Netzwerken wird diese Einsynchronisierung nicht mehr benötigt. Aus Kompatibilitätsgründen sieht der IEEE 802.3 Standard diese Bytes jedoch weiterhin vor.

Die sieben Bytes der Präambel haben alternierende Bits (101010... bzw. 0x55), gefolgt von dem *Start Frame Delimiter* in der Form 10101011.

A.1.2. MAC-Adressen

Die *MAC-Adressen* (*Medium Access Control (MAC)*) haben die Aufgabe, den Sender und Empfänger eines Datenpakets eindeutig zu identifizieren. Dazu besteht jede *MAC-Adresse* aus sechs Bytes. Das Bit null der *MAC-Adresse* signalisiert, ob es sich um eine Unicast- (0) oder Broadcast/Multicast- (1) Nachricht handelt. Unicast-Nachrichten sind an genau einen Empfänger gerichtet, während Broadcast/Multicast-Nachrichten an eine ganze Gruppe von Empfängern gerichtet sind. Der zweitgenannte Typ von Nachrichten wird zum Beispiel dafür verwendet, eine *ARP-Anfrage* «Address Resolution Protocol» zu senden.

Bit eins der *MAC-Adresse* signalisiert, ob es sich um eine globale (0) oder lokale (1) Adresse handelt. Die globalen Adressen werden hierbei vom Netzwerkkartenhersteller vergeben und sind einmalig. Die lokalen Adressen hingegen können vom Benutzer selbstständig verändert und angepasst werden. Hierbei kann es dann allerdings vorkommen, dass zwei Geräte die gleiche *MAC-Adresse* verwenden. In diesem Fall kann die Adressierung eines Ethernetpakets nicht fehlerfrei stattfinden. Die meisten Implementierungen erlauben ein Anpassen der *MAC-Adressen*. Unter *Microsoft Windows* kann dies beispielsweise über die *Registry* erfolgen.

Im *Ethernetframe* gibt die «Ziel-MAC-Adresse» den Empfänger des Ethernetpakets an, während die «Quell-MAC-Adresse» den Absender des Ethernetpakets kennzeichnet.

A.1.3. Typ-Feld

Das *Typ-Feld* (EtherType) gibt den Protokolltyp des nächsthöheren Protokolls innerhalb der Nutzdaten an. Einige mögliche Protokolltypen sind:

- 0x0800 IP Internet Protocol (IPv4)
- 0x0806 Address Resolution Protocol (ARP)
- 0x8035 Reverse Address Resolution Protocol (RARP)
- 0x8100 IEEE 802.1Q-tagged frame (VLAN)
- 0x86DD IP Internet Protocol, Version 6 (IPv6)

A.1.4. Nutzdaten

Die Nutzdaten dürfen eine Länge zwischen 0 Byte und 1500 Bytes haben. In ihnen werden die eigentlichen Nutzdaten des Ethernetpakets, also das unter *EtherType* angegebene Protokoll, übertragen.

A.1.5. PAD-Feld

Die minimale Länge eines Ethernet-Frames muss 64 Bytes betragen. Sobald die Nutzdatenmenge zu gering ist, um eine minimale Ethernetframe-Länge von 64 Bytes zu erreichen, wird das *PAD-Feld* dazu verwendet, die Framelänge auf 64 Bytes aufzufüllen. Hierzu werden an das Ende der Nutzdaten Füllbytes angehängt. Das mit *EtherType* angegebene Protokoll hat anschließend dafür zu sorgen, dass diese Füllbytes beim Empfang nicht interpretiert werden.

A.1.6. CRC-Prüfsumme

Am Ende eines *Ethernetframes* steht eine vier Bytes lange *CRC-Checksumme* (*Cyclic Redundancy Check (CRC)*). Diese Checksumme wird über den ganzen *Ethernetframe*, also von *MAC-Adressen* bis *PAD-Feld*, berechnet. Der Empfänger des Ethernetpakets berechnet die Checksumme erneut und vergleicht diese mit der empfangenen. Sollte eine Differenz auftreten, wird das empfangene Datenpaket verworfen.

A.1.7. VLAN-Tag

Nach der *IEEE 802* besteht die Möglichkeit, virtuelle Netzwerke (*Virtual Local Area Network (VLAN)*) innerhalb eines Übertragungsmediums zu verwenden. Hierzu kann in einen *Ethernetframe* ein vier Bytes langer *VLAN-Tag* eingefügt werden. Die Bytes dieses *VLAN-Tags* werden direkt vor die zwei Bytes des Typ-Felds gesetzt, so dass die ersten beiden Bytes des *VLAN-Tag* die Positionen des Typ-Felds einnehmen.

Ein *VLAN-Tag* hat in seinen ersten beiden Bytes den Wert 0x8100, was dem *EtherType* «IEEE 802.1Q-tagged frame» entspricht. Das Ethernetpaket signalisiert somit, dass es sich um einen *VLAN-Tag* handelt. In den darauf folgenden drei Bits wird die *VLAN-Priority* übertragen, gefolgt von einem Bit für den «Canonical Format Indicator». Danach kommen noch 12 Bits für die *VLAN-ID*, welche $2^{12} = 4096$ virtuelle Netzwerke ermöglicht.

Durch das Einfügen eines *VLAN-Tags* ist es außerdem möglich, eine maximale Ethernetframe-Länge von 1522 Bytes zu erreichen.

A.2. Address Resolution Protocol

Anwendungsschicht		HTTP	SNMP	
Transportschicht		TCP	UDP	
Internetschicht	ICMP			
Internetschicht	IP			ARP
Netzzugangsschicht	Ethernet			

Tabelle A.2.: Position von ARP im TCP/IP-Referenzmodell

Das «*Address Resolution Protocol*» (ARP) arbeitet laut RFC826 [99] (*Request for Comments (RFC)*) auf Schicht drei des *OSI-Referenzmodells* und dient der Zuordnung von Netzwerkadressen (*IP-Adressen*) zu Hardwareadressen (*MAC-Adressen*), da es zwischen diesen keine festen Zuordnungen gibt. Wenn ein Netzwerkteilnehmer seine Daten über *Ethernet* versenden möchte, so muss die *Ziel-MAC-Adresse* im *Ethernetheader* eingetragen werden. Wenn diese jedoch nicht bekannt ist, wird *ARP* verwendet, um die *MAC-Adresse* des Zielsystems zu ermitteln.

Wie Abbildung A.4 zeigt, versucht der Sender zuerst die geeignete Schnittstelle zur Kommunikation zu finden. Für lokale Anfragen, also Datenpakete an sich, muss der Sender die Ethernetschnittstelle nicht verwenden und muss somit auch keine Adressen auflösen. Stattdessen kann er mit seiner bekannten *MAC-Adresse* sofort das Paket versenden. Für den Fall, dass das Paket aber an einen anderen Rechner gesendet werden soll, muss der Sender nun die *MAC-Adresse* des Zielsystems ermitteln. Hierzu überprüft er zuerst seine interne *ARP-Tabelle* (*ARP-Cache*). Findet er in dieser Tabelle eine passende *MAC-IP-Zuweisung*, so muss überprüft werden, ob diese Daten noch gültig sind. Dies ist notwendig, da die Angaben im *ARP-Cache* nach einer gewissen Zeit verfallen. Bei auf Linux basierenden Betriebssystemen liegt diese Gültigkeitsdauer bei circa fünf Minuten. Sollte die Überprüfung noch gültige Daten liefern, kann das Ethernetpaket sofort versendet werden. Für den Fall, dass diese MAC-IP-Zuordnung jedoch bereits abgelaufen ist oder keine passende Zuordnung gefunden werden kann, muss der Sender die *MAC-Adresse* des Empfängers zuerst ermitteln. Dies geschieht über die eigentliche *ARP-Anfrage*. Da der Sender nun zuerst ein Datenpaket versenden muss, um die *MAC-Adresse* des Empfängers zu ermitteln, kann das eigentliche Nutzdatenpaket zu diesem Zeitpunkt nicht zugestellt werden. Daher wird dieses verworfen. *ARP* befindet sich in

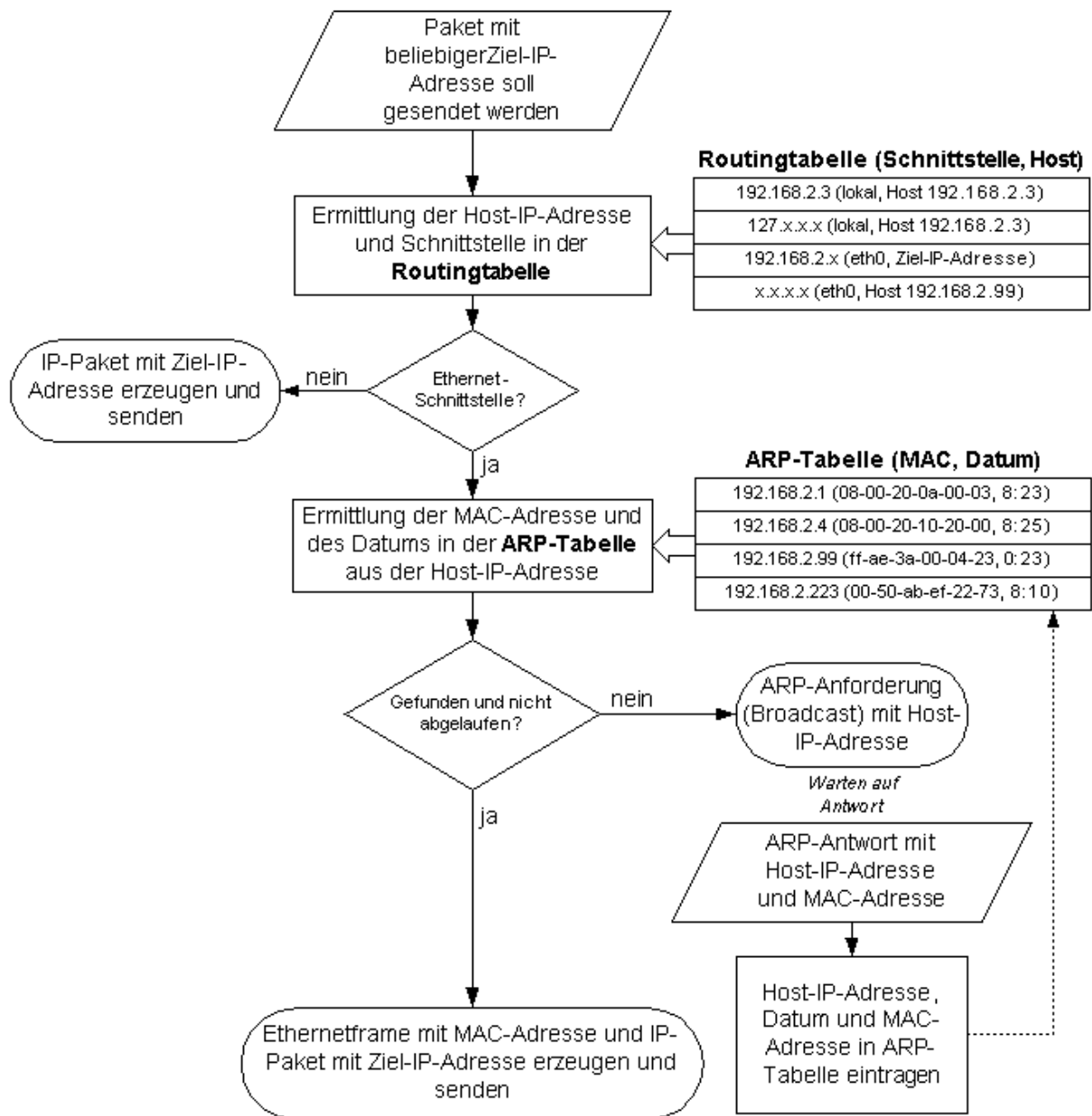


Abbildung A.4.: Ablauf zum Senden eines IP-Pakets [126] (modifiziert)

ANHANG A. NETZWERKGRUNDLAGEN

den unteren Schichten des Netzwerkprotokollstacks. In diesen Schichten ist es erlaubt, dass Pakete verloren gehen dürfen. Die darüberliegenden Schichten haben in diesem Fall die Aufgabe, eine erneute Übertragung dieser Daten anzufordern.

Sobald aus dem Netzwerk eine *ARP-Antwort* (*ARP-Reply*) empfangen wird, wird die darin enthaltene *MAC-IP-Zuordnung* in den *ARP-Cache* eingetragen. Wenn die höheren Netzwerkprotokollschichten nun eine neue Übertragung des verloren gegangenen Pakets anfordern, kann dieses mit der nun aktualisierten *ARP-Tabelle* zugestellt werden.

A.2.1. Aufbau eines ARP-Datagramms

Ein *ARP-Request* und eine *ARP-Response* werden jeweils in einem *ARP-Datagramm* übertragen. Hierbei ist das *ARP-Datagramm* im Nutzdatenbereich eines *Ethernetframes* untergebracht. Da ein *Ethernetframe* mindestens 64 Bytes lang sein muss, müssen dem *ARP-Paket* eventuell Paddingbytes angehängt werden. Zum Sendezeitpunkt eines *ARP-Requests* ist die Ziel-MAC-Adresse noch nicht bekannt, daher muss mit einem *Ethernet-Broadcast* jeder Rechner im aktuellen Subnetz nach der gesuchten IP befragt werden. Hierzu wird das Ethernetpaket an die *MAC-Adresse* $FF - FF - FF - FF - FF - FF$ gesendet. Diese *MAC-Adresse* ist reserviert für einen *Broadcast* und somit wird dieser Ethernetframe von jedem Netzwerkteilnehmer im aktuellen Subnetz entgegengenommen und verarbeitet. Da *ARP* auf *Ethernet-Broadcasts* aufbaut, sind ARP-Anfragen über Subnetze hinweg nicht möglich. Im Fall einer benötigten subnetzübergreifenden Anfrage antwortet der jeweilige Router mit seiner *MAC-Adresse* auf das Paket. Auf diese Weise werden die nachfolgenden Datenpakete direkt an den Router gesendet, welcher diese «*forwarded*». Ansonsten antwortet der Rechner mit der gesuchten *IP-Adresse* auf die Anfrage durch einen *ARP-Reply*. Die restlichen Rechner aktualisieren ihren eigenen *ARP-Cache* mit den empfangenen Informationen. Ein ARP-Datenpaket ist in Abbildung A.5 zu sehen. Die einzelnen Datenfelder haben die folgenden Funktionen:

A.2.1.1. Hardwareadrestyp

Dieses Feld enthält eine Angabe über den Typ der *MAC-Adresse*. Bei Ethernet muss als Hardwareadrestyp der Wert eins gewählt werden.

ANHANG A. NETZWERKGRUNDLAGEN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31																								
Hardwareadrestyp																Protokolladrestyp																																							
Hardwareadressgröße								Protokolladressgröße								Operation																																							
Quell-MAC-Adresse																																																							
Quell-MAC-Adresse																Quell-IP-Adresse																																							
Quell-IP-Adresse																Ziel-MAC-Adresse																																							
Ziel-MAC-Adresse																																																							
Ziel-IP-Adresse																																																							

Abbildung A.5.: Aufbau eines ARP-Datagramms

A.2.1.2. Protokolladrestyp

Diese Angabe wählt den Protokolltyp für die *MAC-Adresse*. Für *IPv4* ist der Wert 0x0800 zu wählen.

A.2.1.3. Hardwareadressgröße

Die Hardwareadressgröße gibt die Länge der *MAC-Adresse* an. Im Falle von *Ethernet* beträgt diese Länge sechs Bytes.

A.2.1.4. Protokolladressgröße

Bei Ethernet beinhaltet dieses Feld die Länge einer *IP-Adresse*. Dies bedeutet für *IPv4* den Wert vier und für *IPv6* den Wert 16.

A.2.1.5. Operation

Mit Hilfe dieser Angabe wird die eigentliche Aufgabe des ARP-Datagramms angegeben. Der Wert eins steht hierbei für einen *ARP-Request*, während eine zwei für eine *ARP-Response* steht.

A.2.1.6. Quell-MAC-Adresse

Dieses Feld enthält die *MAC-Adresse* des Absenders vom aktuellen Paket.

ANHANG A. NETZWERKGRUNDLAGEN

A.2.1.7. Quell-IP-Adresse

Mit der Quell-IP-Adresse wird die *IP-Adresse* des Absenders angegeben.

A.2.1.8. Ziel-MAC-Adresse

Diese Angabe ist bei einem *ARP-Request* nicht definiert. Da die *MAC-Adresse* des Empfängers zu diesem Zeitpunkt noch nicht bekannt ist, wird die für einen *Broadcast* zuständige Adresse $FF - FF - FF - FF - FF$ verwendet. Bei einer *ARP-Response* beinhaltet dieses Feld dagegen die *MAC-Adresse* des Empfängers.

A.2.1.9. Ziel-IP-Adresse

Das Feld *Ziel-IP-Adresse* enthält die *IP-Adresse* des Empfängers dieses Datagramms.

A.2.2. Proxy ARP

Wie bereits erwähnt, können *Ethernet-Broadcasts* nur innerhalb eines Subnetzes versendet werden. Soll jedoch ein Host in einem anderen Subnetz gefunden werden, so muss diese Anfrage dennoch verarbeitet werden können. Hierzu antwortet der zuständige Router mit seiner eigenen *MAC-Adresse* auf den *ARP-Request*. Da der Router in diesem Fall stellvertretend für einen anderen Netzwerkteilnehmer den *ARP-Request* beantwortet, wird dieses Verfahren «*Proxy ARP*» genannt.

Wenn der Sender anschließend die Datenpakete an diese *MAC-Adresse* sendet, kommen diese direkt beim Router an. Dieser wiederum nimmt die Pakete entgegen und leitet sie an das Ziel-Subnetz weiter, wo die Pakete ihr eigentliches Zielsystem erreichen. Für diese Datenübertragung verhält sich der Router transparent; das heißt, der Sender und Empfänger bemerken nichts von der Arbeit des Routers.

Die Tätigkeit eines Routers kann vom Quell- und Zielsystem nur anhand des *ARP-Cache* erkannt werden. Wenn in dieser Tabelle mehrere *IP-Adressen* die gleiche *MAC-Adresse* zugewiesen bekommen, kann davon ausgegangen werden, dass sich ein Router im Übertragungszeitpunkt befindet. Ein anderer Grund für die Zuordnung der gleichen *MAC-Adresse* zu unterschiedlichen *IP-Adressen* könnte aber auch ein Hackerangriff mittels *ARP-Spoofing* sein. Hierbei werden die IP-MAC-Zuordnungen gezielt verfälscht, um den Datenfluss im Netzwerk umleiten zu können. Mit diesem Verfahren ist es zum Beispiel möglich, die Daten zwischen zwei Kommunikationspartnern über einen weiteren

ANHANG A. NETZWERKGRUNDLAGEN

Rechner zu leiten, welcher die Daten mithört kann. Dabei handelt es sich dann um einen *Man-In-The-Middle* Angriff.

A.2.3. Gratuitous ARP

Beim *Gratuitous ARP* handelt es sich um ein unaufgefordertes *ARP*. Dies bedeutet, dass ein Netzwerkteilnehmer einen *ARP-Request* versendet, ohne eine Antwort darauf zu erwarten. Stattdessen hat diese Anfrage die Aufgabe, den *ARP-Cache* aller mithörenden Systeme zu aktualisieren. Der Sender verschickt hierbei einen *ARP-Request*, bei welchem als Quell- und Ziel-IP-Adresse seine eigene Adresse eingetragen ist. Da eine *IP-Adresse* in einem Subnetz nur einmal vorkommen darf, darf auf dieses Paket auch kein anderes Netzwerkgerät antworten. Sollte dies doch der Fall sein, liegt ein Fehler in der IP-Vergabe vor.

Gratuitous ARP wird bei vielen Rechnern während des Bootvorgangs ausgeführt, um den anderen Systemen die Adresse des neu hinzugekommenen Geräts mitzuteilen. Ein anderes Anwendungsgebiet liegt bei redundanten Systemen. Sobald bei diesen Systemen Gerät A ausfällt, kann Gerät B den kompletten Traffic von Gerät A übernehmen. Hierzu verknüpft Gerät B seine *MAC-Adresse* mit der *IP-Adresse* von Gerät A und macht diese Kombination per *Gratuitous ARP* bekannt.

A.2.4. Reverse ARP (RARP)

Nach RFC903 [37] funktioniert *RARP* umgekehrt wie *ARP*, das heißt, es findet zu einer *MAC-Adresse* die zugehörige *IP-Adresse*. Hierzu werden normale *ARP-Pakete* verwendet. Bei diesen wird für einen *RARP-Request* im *Operation-Feld* der Wert drei angegeben, für einen *RARP-Reply* der Wert vier.

Die Quell- und Ziel-IP-Adressen sind bei einem *RARP-Request* undefiniert, da diese zu dem Zeitpunkt noch nicht bekannt sind. Das Feld für die Quell-MAC-Adresse enthält die Adresse des Senders. Die Ziel-MAC-Adresse wird auf die *MAC-Adresse* des Empfängers gesetzt oder falls der Sender seine eigene *IP-Adresse* ermitteln möchte, enthält das Feld die Adresse des Senders.

Bei einem anschließenden *RARP-Reply* enthält das Feld der Quell-IP-Adresse die Adresse des Senders. Die Ziel-IP-Adresse adressiert den Empfänger. Die Angaben für die *MAC-Adressen* sind analog zu den angegebenen *IP-Adressen*.

A.2.5. Schwachstellen von ARP

Da es sich bei *ARP* um ein grundlegendes Protokoll zum Auflösen von *IP-Adressen* handelt, hängt die Zuverlässigkeit von IP-Verbindungen stark von diesem Protokoll ab. Dies hat zur Folge, dass eine Kommunikation nicht mehr fehlerfrei funktionieren kann, wenn zum Beispiel der *ARP-Cache* eines Systems fehlerhafte Daten enthält. In diesem Fall kann die gesuchte *IP-Adresse* nicht mehr fehlerfrei einer *MAC-Adresse* zugeordnet werden und somit auch keine Datenübertragung stattfinden. Ein solch fehlerhafter *ARP-Cache* lässt sich unter anderem auf einen langsamen Netzwerkteilnehmer zurückführen, welcher durch hohe Systemlast erst recht spät auf einen *ARP-Request* antwortet. Wenn der *ARP-Reply* erst eintrifft, nachdem sich die *IP-MAC-Zuordnungen* im Netzwerk geändert haben, so wird der *ARP-Cache* mit «alten Daten» aktualisiert.

Wie bereits erwähnt, ist *ARP* auch anfällig gegenüber gezieltem Verfälschen des *ARP-Cache*, dem sogenannten «*ARP-Spoofing*».

Um diese Probleme bestmöglichst umgehen zu können, nehmen moderne *ARP*-Implementierungen nur noch *ARP-Replies* entgegen, zu welchen sie zuvor einen *ARP-Reply* gesendet haben.

A.3. Internet Protocol

Anwendungsschicht		HTTP	SNMP	
Transportschicht		TCP	UDP	
Internetschicht	ICMP			
Internetschicht	IP			ARP
Netzzugangsschicht	Ethernet			

Tabelle A.3.: Position von IP im TCP/IP-Referenzmodell

Das *Internet Protocol* (kurz: IP) entspricht der Implementierung der Schicht drei «Vermittlungsschicht» des *OSI-Referenzmodells*. Es handelt sich bei dieser Schicht um die erste vom Übertragungsmedium unabhängige bzw. routingfähige Schicht und somit um das unterste Protokoll, mit welchem Netzwerke aufgebaut werden können. Dies beruht darauf, dass mit IP die Vergabe von *IP-Adressen* und *Subnetz-Masken* möglich ist. Mit Hilfe dieser Adressen können Netzwerke mit Routern aufgebaut und die Datenpakete zwischen verschiedenen Netzwerken weitergeleitet werden.

ANHANG A. NETZWERKGRUNDLAGEN

Das *Internet Protocol* wird hauptsächlich noch in der Version 4 (*IPv4*) verwendet, welches *IP-Adressen* mit einer Länge von 32 Bits verarbeitet. Damit sind $2^{32} = 4.294.967.296$ Adressen möglich. Diese Anzahl von Netzwerkgeräten reicht im Jahr 2007 gerade noch aus. Wenn das Wachstum der Netzwerke und vor allem das Wachstum des Internets aber anhalten, so werden diese vier Milliarden Adressen sehr bald aufgebraucht sein. Aus diesem Grund wurde das *Internet Protocol* in der Version 6 (*IPv6*) entwickelt, welches Adressen mit einer Länge von 128 Bits verwendet und somit $2^{128} = 340.282.366.920.938.463.463.374.607.431.768.211.456 \approx 340$ Sextillionen Netzwerkgeräte adressieren kann.

Im Rahmen dieser Diplomarbeit wird allerdings nur *IPv4* verwendet, da *IPv6* bisher noch nicht so weit verbreitet ist. Die Spezifikationen zu *IPv4* finden sich in der RFC791 [96], diejenigen zu *IPv6* in der RFC2460 [27].

A.3.1. Adressierung mit IPv4

Der 32-Bit-Wert der *IPv4-Adressen* wird zur einfacheren Handhabung in Blöcken zu je acht Bit geschrieben. Somit wird beispielsweise aus der Adresse 11000000 10101000 00001010 00000001 die einfacher darzustellende Adresse 192.168.10.1.

Diese Adresse setzt sich aus einem *Netzwerkteil* und einem *Hostteil* zusammen, welche durch die *Subnetmask* maskiert werden. Eine solche *Subnetmask* hat beispielsweise die Form 255.255.255.0, dies entspricht der Bitfolge 11111111 11111111 11111111 00000000. Hierbei maskieren die Einsen den *Netzwerkteil* der *IP-Adresse* und die Nullen den *Hostteil*. Solange zwei Netzwerkgeräte den gleichen *Netzwerkteil* der *IP-Adresse* besitzen, befinden sich diese im gleichen Subnetz und können daher direkt miteinander kommunizieren. Versuchen allerdings zwei Geräte aus unterschiedlichen Subnetzen miteinander zu kommunizieren, so wird ein Router benötigt, welcher die Datenpakete von dem einem Netzwerk in das andere weiterleitet.

Anstatt der relativ langen Schreibweise von 255.255.255.0 für die *Subnetmask* kann auch auf die einfachere Darstellung 192.168.0.0/24 zurückgegriffen werden. Dies drückt aus, dass es sich um das Netz mit der Adresse 192.168.0.0 handelt und die linken 24 Bits der *Subnetmask* auf eins gesetzt sind. Somit ist es möglich, in einem 192.168.0.0/24-Netz 254 Rechner bzw. Netzwerkteilnehmer unterzubringen. Dabei werden in diesem Beispiel die ersten 24 Bits für den *Netzwerkteil* der Adresse verwendet, die verbleibenden 8 Bits bilden den *Hostteil*. Durch die acht Bits des *Hostteils* können $2^8 = 256$ Adressen gebildet werden. Allerdings ist zum Einen die Adresse 0 für die eigentliche Netzwerkadresse reserviert, zum Anderen ist die Adresse 255 für einen Broadcast im Netzwerk

ANHANG A. NETZWERKGRUNDLAGEN

Netzwerk	Verwendung
0.0.0.0/8	Aktuelles Netzwerk
127.0.0.0/8	Loopback (Eigener Rechner)
192.168.0.0/16	Privates Netzwerk

Tabelle A.4.: Reservierungen einiger spezieller IPv4-Adressen

Subnetzmaske	11111111	11111111	11111111	00000000	255.255.255.0
Netzwerkteil	11000000	10101000	00001010		
Netzname	11000000	10101000	00001010	00000000	192.168.10.0
Erste Adresse	11000000	10101000	00001010	00000001	192.168.10.1
Letzte Adresse	11000000	10101000	00001010	11111110	192.168.10.254
Broadcast	11000000	10101000	00001010	11111111	192.168.10.255

Tabelle A.5.: Beispielrechnung eines Subnetzes

reserviert. Daraus resultierend können im genannten 192.168.0.0/24-Netz maximal $2^8 - 2 = 254$ Netzwerkteilnehmer betrieben werden.

Laut RFC3330 [55] sind einige Netzwerkadressblöcke für bestimmte Anwendungen reserviert. Die Tabelle A.4 zeigt einen kleinen Auszug aus diesen Reservierungen. Tabelle A.5 zeigt die Adressanteile des Netzes 192.168.10.0/24.

A.3.2. Aufbau des IPv4-Headers

Wie der Abbildung A.1 zu entnehmen ist, werden im Netzwerkprotokollstapel *Header + Daten* einer Schicht im *Datenbereich* der darunterliegenden Schicht übertragen. Somit werden beispielsweise der *Header* und die Nutzdaten des *Internet Protokolls* im Datenbereich des *Ethernetframes* übertragen. In diesem Kapitel wird der Aufbau von *IP* auf Bitebene beschrieben.

Der grundlegende Aufbau des *IP-Headers* [35] ist aus Abbildung A.6 ersichtlich. Die einzelnen Datenbereiche werden im Folgenden genauer beschrieben.

Ein IP-Paket hat im einfachsten Fall einen 20 Bytes langen *Header*; dieser kann bei Bedarf jedoch mit zusätzlichen Optionen auf bis zu 60 Bytes anwachsen. Beim *Internet Protokoll* ist zu beachten,

ANHANG A. NETZWERKGRUNDLAGEN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31				
IP Version				Header Length				Type Of Service								Total Length																			
Identification (Fragment ID)																R	DF	MF	Fragment Offset																
Time-To-Live (TTL)								Protocol								Header Checksum																			
Source IP Address																																			
Destination IP Address																																			
Options																																Padding			
Data																																			

Abbildung A.6.: Aufbau des IPv4-Headers

dass ein Paket beim Empfänger nicht in genau der gleichen Form ankommen muss wie es der Sender abgeschickt hat. Vielmehr kann ein Paket auf dem Weg durch die Netzwerke verloren gehen, beim Empfänger mehrmals ankommen oder fragmentieren. Mit «fragmentieren» wird das Ereignis bezeichnet, dass ein Datenpaket auf dem Weg von Sender zu Empfänger in mehrere kleinere Datenpakete zerstückelt wird, welche dann einzeln beim Empfänger ankommen und jeweils einen eigenen *IP-Header* vorangestellt haben.

Der Grund einer Fragmentierung ist folgender: Ein IP-Paket, welches auf *Ethernet* basiert, darf eine maximale Länge von 1500 Bytes aufweisen (siehe Abbildung A.3). Wenn dieses Datenpaket nun zwischen Sender und Empfänger ein Netzwerk mit einer anderen Technologie passieren muss, so kann es vorkommen, dass dieses andere Netzwerk eine *Maximum Transfer Unit (MTU)* < 1500 Bytes hat. Somit muss das Format des IP-Pakets angepasst werden, um das Netzwerk passieren zu können. Hierzu werden nun die Nutzdaten des IP-Pakets in kleinere Blöcke aufgeteilt. Anschließend bekommt jeder dieser Blöcke einen eigenen *IP-Header* vorangestellt und setzt die Reise zum Empfänger fort. Auf dem Weg können die Pakete immer nur weiter fragmentieren. Ein «Defragmentieren» ist erst beim Empfänger möglich.

Weiterhin wird für jedes Datenpaket, welches über ein Netzwerk übertragen wird, die Route individuell festgelegt. Somit kann die Zeit, welche ein Paket zum Passieren der Strecke benötigt, variieren. Sollte ein Datenpaket auf dem Weg einen Router erreichen, welcher überlastet ist, so kann dieser Router das Paket verwerfen.

Beim *Internet Protokoll* in Version 4 haben die einzelnen Blöcke des *Headers* die folgenden Aufgaben:

ANHANG A. NETZWERKGRUNDLAGEN

Binärwert	Dezimalwert	Version
0000	0	reserviert
0001, 0010, 0011	1-3	nicht vergeben
0100	4	IPv4
0101	5	Stream IP Datagram Mode (experimentell)
0110	6	IPv6 (bzw. IPng)
0111	7	TP/IX: The Next Internet
1000	8	«P» Internet Protocol
1001	9	TUBA
1010-1110	10-14	nicht vergeben
1111	15	reserviert

Tabelle A.6.: Versionen des IP-Protokolls

A.3.2.1. IP Version

Die *IP-Version* gibt an, um welche Version des IP-Pakets es sich handelt. Die möglichen Werte sind in Tabelle A.6 zusammengefasst.

A.3.2.2. Header Length

Dieser Wert gibt die Länge des *IP-Headers* in 32-Bit-Blöcken an. Da für die Minimallänge eines *IP-Headers* 20 Bytes vorgeschrieben sind, liegt der Minimalwert für die *Header Length* bei $\frac{20 \cdot 8}{32} = 5$. Der Maximalwert liegt bei $\frac{60 \cdot 8}{32} = 15$. Durch das *Options-Feld* im *IP-Header* kann die Startposition der Nutzdaten variieren. Da die Länge des *Headers* jedoch bekannt ist, kann die Startposition ermittelt werden. Weiterhin ist durch die Headerlänge erkennbar, ob Zusatzoptionen im *Header* vorhanden sind.

A.3.2.3. Type Of Service

Das *Type Of Service* Feld enthält Angaben zur Verarbeitung des Datenpakets. Es wird somit zum Beispiel ermöglicht, dass Daten, welche zum Steuern des Datenflusses benötigt werden, vorrangig verarbeitet werden. Der Aufbau des «Type Of Service»-Felds kann Tabelle A.7 entnommen werden.

ANHANG A. NETZWERKGRUNDLAGEN

0	1	2	3	4	5	6	7
Precedence			Delay	Throughput	Reliability	Cost	MBZ

Tabelle A.7.: Angaben im «Type Of Service»-Feld

Type Of Service	Verwendung
111	Steuerung des Netzwerks
110	Steuerung des Verbundnetzes
001	Einfache Priorität
000	Routine

Tabelle A.8.: Mögliche Prioritätsangaben bei IPv4

Eine ausführliche Beschreibung aller Optionen des «Type Of Service»-Feldes ist in der RFC1349 [5] zu finden.

A.3.2.3.1. Precedence Das *Precedence* Feld gibt die Priorität des *IP-Pakets* an. Es wurde vom *Department of Defense* definiert, wird jedoch heutzutage kaum noch verwendet. Vier mögliche Prioritätsangaben sind Tabelle A.8 zu entnehmen.

A.3.2.3.2. Delay Dieses Bit signalisiert, dass das Datenpaket mit minimaler Verzögerung transportiert werden soll.

A.3.2.3.3. Throughput Durch dieses Bit wird ein maximaler Datendurchsatz gefordert.

A.3.2.3.4. Reliability Datenpakete, welche dieses Bit gesetzt haben, sollen möglichst zuverlässig übertragen werden.

A.3.2.3.5. Cost Die Übertragung eines Pakets, bei welchem das *Cost* Bit gesetzt ist, soll möglichst geringe Kosten verursachen.

A.3.2.3.6. MBZ MBZ steht hier für *Must Be Zero*. Es handelt sich dabei um ein Bit, welches immer auf null gesetzt ist.

A.3.2.4. Total Length

Dieses Feld gibt die komplette Länge des *IP-Pakets* an, also die Länge von *Header* und *Nutzdaten* in Bytes. Für die Längenangabe stehen maximal 16 Bits und für die Paketlänge somit 65535 Bytes¹ zur Verfügung. Wird von dieser Länge die des *Headers* abgezogen, resultiert dies in der maximalen Länge der Nutzdaten, welche mit einem *IP-Paket* verschickt werden können.

Es muss hierbei jedoch beachtet werden, dass große Datenpakete eher dazu neigen, fragmentiert zu werden, als kleinere Pakete.

A.3.2.5. Identification (Fragment ID)

Bei der *Fragment ID* handelt es sich um eine 16-Bit-Zahl, welche ein *IP-Datagramm* eindeutig identifiziert. Wird ein solches *IP-Datagramm* nun fragmentiert, so erhalten alle Fragmente diese *Identifikationsnummer (ID)* zur Kennung.

A.3.2.6. Flags

A.3.2.6.1. R Das *R-Bit* steht für «reserved» und kennzeichnet ein Bit, welches in IPv4 nicht verwendet wird, und immer auf *null* gesetzt ist.

A.3.2.6.2. DF *DF* steht bei diesem Bit für «Do not fragment». Wenn dieses Bit auf eins gesetzt ist, dürfen Netzwerkgeräte das IP-Datagramm nicht fragmentieren.

A.3.2.6.3. MF Dieses Bit hat die Bedeutung *More fragments to come*. Eine Eins signalisiert, dass weitere Fragmente des Datagramms folgen. Eine Null signalisiert dem Empfänger, dass er das letzte Fragment erhalten hat.

¹ $2^{16} - 1 = 65535$

ANHANG A. NETZWERKGRUNDLAGEN

Header	1500 Datenbytes
DF=0	Byte 0 ... 1499
ID=123	

Tabelle A.9.: Ursprüngliches IP-Datagramm

Header	Daten	Header	Daten	Header	Daten
MF=1	Byte 0 ... 503	MF=1	Byte 504 ... 1007	MF=0	Byte 1008 ... 1499
ID=123		ID=123		ID=123	
Offset=0		Offset=63		Offset=126	

Tabelle A.10.: Mögliche Fragmentierung des IP-Datagramms

A.3.2.7. Fragment Offset

Bei dem *Fragment Offset* handelt es sich um einen 13-Bit-Wert, welcher die Position des Fragments innerhalb des Original-IP-Datagramms angibt. Dabei erhält das erste Fragment den Wert null und alle weiteren Pakete erhalten einen Wert, welcher die Position in 8-Byte-Schritten angibt.

Somit ergibt der *Fragment Offset* multipliziert mit *acht Bytes* die absolute Position des Fragments im Original-IP-Datagramm. Tabelle A.10 zeigt ein Beispiel für eine mögliche Fragmentierung des IP-Pakets aus Tabelle A.9.

A.3.2.8. Time-To-Live (TTL)

Die *Time-To-Live* gibt an, wie lange ein IP-Datenpaket maximal im Netzwerk unterwegs sein darf. Der Sender setzt die *TTL* auf einen gewissen Wert in Sekunden und jeder Router, über den das Paket geleitet wird, dekrementiert die *TTL*. Dabei ist vorgesehen, dass die *TTL* jeweils um die Zeit dekrementiert wird, welche das Paket durch den jeweiligen Router benötigt hat.

Es ist jedoch nicht immer möglich, die *TTL* exakt um die Verweilzeit zu dekrementieren, da die Router das Paket meist in einer viel kürzeren Zeit als einer Sekunde verarbeiten. Weiterhin ist es für einen Router sehr aufwendig, für jedes Datenpaket die exakte Verweildauer zu bestimmen. Daher verringern die meisten Router die *TTL* einfach um den Wert eins. Somit ist die *Time-To-Live* Angabe eher als *Hop-Count* statt als exakte Verweilzeit zu sehen.

Sobald die *TTL* den Wert null erreicht hat, wird das IP-Datenpaket vom Router verworfen.

A.3.2.9. Protocol

Dieser 8-Bit-Wert gibt an, welcher Protokolltyp im Datenbereich des *IP-Pakets* übertragen wird. Die möglichen Protokolle sind in RFC1700 [106] unter dem Abschnitt *Assigned Internet Protocol Numbers* definiert.

A.3.2.10. Header Checksum

Jedes IP-Paket enthält eine *Header Checksum*. Hierbei handelt es sich um eine einfache Prüfsumme zur Kontrolle des *Headers*. Sobald ein Netzwerkgerät ein *IP-Paket* empfängt, wird die Prüfsumme über den kompletten *Header* - ohne die *Nutzdaten* - gebildet. Diese Überprüfung muss im Ergebnis 0xFFFF resultieren, andernfalls ist ein Fehler aufgetreten und das Datenpaket wird verworfen.

Nachdem der Router anschließend die *TTL* angepasst hat, wird die neue *Header Checksum* berechnet und das Paket weitergeleitet.

Die eigentliche Checksummen-Berechnung basiert auf einfacher Einerkomplement-Arithmetik. Bei einem empfangenen Paket werden zuerst die Werte aller 16-Bit-Blöcke des *Headers* addiert, anschließend wird ein eventueller Überlauf zum Ergebnis dazu addiert und die daraus resultierende Summe wird auf den Wert 0xFFFF geprüft.

Soll stattdessen ein Paket gesendet werden, so wird das Checksummen-Feld zuerst auf den Wert 0x0000 gesetzt, anschließend werden wieder alle Werte der 16-Bit-Blöcke des *Headers* addiert. Ein möglicher Überlauf wird anschließend noch zum Ergebnis addiert und alle Bits der Summe werden invertiert. Bei dem daraus resultierenden 16-Bit-Wert handelt es sich um die Checksumme des *IP-Headers*, welche in das zu sendende Paket eingesetzt wird.

Der Vorgang der IP-Checksummenberechnung soll anhand des folgenden Beispiels verdeutlicht werden:

A.3.2.10.1. Prüfsummen-Kontrolle beim Empfang eines IP-Pakets Der Empfangene Header hat die Daten: 0x45 0x00 0x00 0x34 0x0C 0x01 0x40 0x00 0x80 0x06 0xFE 0x41 0xC0 0xA8 0x0A 0x67 0x48 0x0E 0xDD 0x63

Nun muss zuerst die Summe aller 16-Bit-Blöcke gebildet werden, also $0x4500 + 0x0034 + 0x0C01 + 0x4000 + 0x8006 + 0xFE41 + 0xC0A8 + 0x0A67 + 0x480E + 0xDD63 = 0x3FFFC$. Da das Ergebnis einen Überlauf enthält, muss dieser zum eigentlichen 16-Bit-Wert addiert werden, also $0x0003 + 0x3FFFC =$

ANHANG A. NETZWERKGRUNDLAGEN

0xFFFF. Das Ergebnis lautet 0xFFFF und somit wurde der *IP-Header* erfolgreich überprüft und als gültig empfunden.

A.3.2.10.2. Prüfsummen-Erzeugung beim Senden eines IP-Pakets Wenn das oben genannte IP-Paket nun erneut versendet werden soll, muss zuerst die Header-Checksumme neu berechnet werden. Dazu wird der 16-Bit-Wert der alten Checksumme zuerst auf 0x0000 gesetzt, also 0x45 0x00 0x00 0x34 0x0C 0x01 0x40 0x00 0x80 0x06 0x00 0x00 0xC0 0xA8 0x0A 0x67 0x48 0x0E 0xDD 0x63.

Anschließend wird wieder die Summe über den kompletten Header gebildet: $0x4500 + 0x0034 + 0x0C01 + 0x4000 + 0x8006 + 0x0000 + 0xC0A8 + 0x0A67 + 0x480E + 0xDD63 = 0x301BB$

Der Überlauf wird wieder zum 16-Bit-Ergebniss addiert. Somit ergibt sich $0x0003 + 0x01BB = 0x01BE$. Dieses Ergebnis muss nun noch invertiert werden: $\overline{0x01BE} = 0xFE41$

Das Ergebnis ist also wieder der Wert 0xFE41, welcher auch zuvor die Checksumme repräsentierte. Da an den Daten des *Headers* nichts verändert wurde, musste der *Header* auch wieder diesen Wert erhalten.

A.3.2.11. Source IP Address

Dieses Feld enthält die 32-Bit breite Quelladresse des IP-Pakets. Die Angabe erfolgt im Format *Big Endian*, also mit dem höchstwertigsten Byte zuerst.

Destination IP Address

In diesem Feld steht die Zieladresse des Pakets. Das Datenformat ist das Gleiche wie bei der *Source IP Address*.

A.3.2.12. Options

Im *Options-Feld* können Zusatzangaben zum Routing gemacht werden. Hierfür können eine oder mehrere Optionen angegeben werden. Da die Größe des *Headers* beschränkt ist, dürfen die Zusatzoptionen maximal 40 Bytes lang sein.

Die Optionen setzen sich aus *Oktetts* zusammen. Es gibt hierbei zwei unterschiedliche Fälle. Im einen Fall besteht eine Option aus einem einzelnen *Option-Type Octet*, im anderen Fall besteht die

ANHANG A. NETZWERKGRUNDLAGEN

0	1	2	3	4	5	6	7
C	Class		Option				

Tabelle A.11.: Aufbau des «Option-Type Octet»

Wert	Beschreibung
0	Control
1	Reserved
2	Debugging and measurement
3	Reserved

Tabelle A.12.: Werte für die «Option Class»

komplette Option aus einem Block von *Option-Type Octet*, *Octet-Length Octet* und einer bestimmten Anzahl von *Option-Data Octets*. Diese drei verschiedenen Typen von *Octetts* haben folgende Bedeutungen:

A.3.2.12.1. Option-Type Octet Ein Datenfeld vom Typ *Option-Type Octet* hat den Aufbau wie ihn Tabelle A.11 zeigt. Dieser Typ setzt grundlegende Parameter für eine Option und gibt an, welcher Optionstyp verwendet wird.

Die einzelnen Angaben haben hierbei die nachfolgenden Bedeutungen:

A.3.2.12.1.1. C Hierbei handelt es sich um ein Flag, welches angibt ob die Option auch an fragmentierte Pakete angehängt werden soll oder nicht. Dabei repräsentiert eine Null den Zustand *Do not copy* und eine Eins steht für *Copy*.

A.3.2.12.1.2. Class Hiermit wird die *Option Class* bestimmt, welche die Bedeutung der Option angibt. Die möglichen Werte sind aus Tabelle A.12 ersichtlich.

A.3.2.12.1.3. Option Diese fünf Bits stehen für die *Option Number* und wählen die eigentliche Option aus. Einige mögliche Optionen sind in Tabelle A.13 aufgezeigt.

Option Class	Option Number	Length	Beschreibung
0	2	11	Sicherheitsoptionen
0	3	var.	Eine Liste von zu passierenden Netzknoten
2	4	var.	Timestamps aufzeichnen
0	7	var.	Komplette Route aufzeichnen
0	9	var.	Komplette Wegangabe für das Paket

Tabelle A.13.: Mögliche Angaben für die «Option Number»

A.3.2.12.2. Option-Length Octet Das *Option-Length Octet* gibt die Gesamtlänge aller *Octetts* einer Option, bestehend aus *Option-Type Octet* + *Option-Length Octet* + *Option-Data Octets*, an.

A.3.2.12.3. Option-Data Octets Die *Option-Data Octets* enthalten die eigentlichen Daten der Option. Die Länge der Daten beträgt:

Angabe im Feld *Option-Length Octets* – 2 (die ersten beiden Oktetts einer Option gehören nicht zu den Optionsdaten).

A.3.2.13. Padding

Die Länge der Optionen muss ein Vielfaches von 32-Bit sein. Sollte diese Forderung nicht automatisch zutreffen, wird der aktuelle 32-Bit-Block mit Füllbits aufgefüllt.

A.3.2.14. Data

An dieser Stelle stehen die eigentlichen Nutzdaten, welche im *IP-Paket* transportiert werden. Normalerweise handelt es sich hierbei um Daten aus Protokollen, welche in einer höheren Schicht im Netzwerkprotokollstapel angeordnet sind.

A.4. Internet Control Message Protocol

Das *Internet Control Message Protocol* (ICMP) aus der RFC792 [101] befindet sich, zusammen mit dem *Internet Protocol*, auf Ebene drei des *OSI-Modells*. Das Protokoll dient zum Austausch

ANHANG A. NETZWERKGRUNDLAGEN

Anwendungsschicht		HTTP	SNMP	
Transportschicht		TCP	UDP	
Internetschicht	ICMP			
Internetschicht	IP			ARP
Netzzugangsschicht	Ethernet			

Tabelle A.14.: Position von ICMP im TCP/IP-Referenzmodell

Bit 0-7	Bit 8-15	Bit 16-23	Bit 24-31
Typ	Code	Checksumme	
Daten (optional)			

Tabelle A.15.: Aufbau eines ICMP-Pakets

von Informations- und Fehlermeldungen über Netzwerkverbindungen und befindet sich hierzu im Datenbereich eines *IP-Pakets*, dessen *Type Of Service* auf null und *Protocol* auf eins gesetzt sind. Es hat den Aufbau nach Tabelle A.15. Laut Definition darf ein *ICMP-Paket* niemals ein anderes *ICMP-Paket* auslösen. Geht also eine *ICMP-Nachricht* verloren, so wird dies nicht durch ein anderes Paket gemeldet.

Tabelle A.14 enthält zwei übereinanderliegende Internetschichten. Dies wurde in der Darstellung so gewählt, da sich *ICMP* zwar auf der selben Schicht wie *IP* befindet, aber dennoch auf die Dienste des «darunterliegenden» *IP* zugreift.

A.4.1. Mögliche ICMP-Typen

Jede *ICMP-Nachricht* enthält eine so genannte *Typnummer*, welche die Nachricht in eine Nachrichtenklasse einteilt. Die hierbei möglichen *Typnummern* sind laut *Internet Assigned Numbers Authority (IANA)* [52] in der Tabelle A.16 zusammengefasst.

ANHANG A. NETZWERKGRUNDLAGEN

Tabelle A.16.: Auflistung der ICMP-Typnummern nach IANA [52]

Typnummer	Nachrichtenklasse
0	Echo Reply
1	Unassigned
2	Unassigned
3	Destination Unreachable
4	Source Quench
5	Redirect
6	Alternate Host Address
7	Unassigned
8	Echo
9	Router Advertisement
10	Router Solicitation
11	Time Exceeded
12	Parameter Problem
13	Timestamp
14	Timestamp Reply
15	Information Request
16	Information Reply
17	Address Mask Request
18	Address Mask Reply
19	Reserved (for Security)
20-29	Reserved (for Robustness Experiment)
30	Traceroute
31	Datagram Conversion Error
32	Mobile Host Redirect
33	IPv6 Where-Are-You
34	IPv6 I-Am-Here

ANHANG A. NETZWERKGRUNDLAGEN

Typnummer	Nachrichtenklasse
35	Mobile Registration Request
36	Mobile Registration Reply
37	Domain Name Request
38	Domain Name Reply
39	SKIP
40	Photuris
41	ICMP messages utilized by experimental mobility protocols such as Seamoby
42-255	Reserved

A.4.2. Angabe des ICMP-Codes

Bei vielen der *ICMP-Nachrichten* kann zur genaueren Spezifikation einer Nachricht zusätzlich ein *ICMP-Code* mit angegeben werden. Die hierbei zu verwendenden Codenummern sind ebenfalls durch die *IANA* [52] aufgeführt und in der Tabelle A.17 dargestellt.

Tabelle A.17.: Auflistung der ICMP-Codes nach IANA [52]

Typ	Code	Nachrichtenklasse	Meldung
0	0	Echo Reply	
1		Unassigned	
2		Unassigned	
3	0	Destination Unreachable	Net Unreachable
3	1	Destination Unreachable	Host Unreachable
3	2	Destination Unreachable	Protocol Unreachable
3	3	Destination Unreachable	Port Unreachable
3	4	Destination Unreachable	Fragmentation Needed and Don't Fragment was Set
3	5	Destination Unreachable	Source Route Failed
3	6	Destination Unreachable	Destination Network Unknown

ANHANG A. NETZWERKGRUNDLAGEN

Typ	Code	Nachrichtenklasse	Meldung
3	7	Destination Unreachable	Destination Host Unknown
3	8	Destination Unreachable	Source Host Isolated
3	9	Destination Unreachable	Communication with Destination Network is Administratively Prohibited
3	10	Destination Unreachable	Communication with Destination Host is Administratively Prohibited
3	11	Destination Network Unreachable for Type of Service	
3	12	Destination Host Unreachable for Type of Service	
3	13	Destination Unreachable	Communication Administratively Prohibited
3	14	Destination Unreachable	Host Precedence Violation
3	15	Destination Unreachable	Precedence cutoff in effect
4	0	Source Quench	
5	0	Redirect	Datagram for the Network (or subnet)
5	1	Redirect	Datagram for the Host
5	2	Redirect	Datagram for the Type of Service and Network
5	3	Redirect	Datagram for the Type of Service and Host
6	0	Alternate Host Address	
7		Unassigned	
8	0	Echo	
9	0	Router Advertisement	Normal router advertisement
9	16	Router Advertisement	Does not route common traffic
10	0	Router Selection	
11	0	Time Exceeded	Time to Live exceeded in Transit

ANHANG A. NETZWERKGRUNDLAGEN

Typ	Code	Nachrichtenklasse	Meldung
11	1	Time Exceeded	Fragment Reassembly Time Exceeded
12	0	Parameter Problem	Pointer indicates the error
12	1	Parameter Problem	Missing a Required Option
12	2	Parameter Problem	Bad Length
13	0	Timestamp	
14	0	Timestamp Reply	
15	0	Information Request	
16	0	Information Reply	
17	0	Address Mask Request	
18	0	Address Mask Reply	
19		Reserved (for Security)	
20-29		Reserved (for Robustness Experiment)	
30		Traceroute	
31		Datagram Conversion Error	
32		Mobile Host Redirect	
33		IPv6 Where-Are-You	
34		IPv6 I-Am-Here	
35		Mobile Registration Request	
36		Mobile Registration Reply	
39		SKIP	
40	0	Photuris	Bad SPI
40	1	Photuris	Authentication Failed
40	2	Photuris	Decompression Failed
40	3	Photuris	Decryption Failed
40	4	Photuris	Need Authentication
40	5	Photuris	Need Authorization
41-252		Unassigned	

ANHANG A. NETZWERKGRUNDLAGEN

Typ	Code	Nachrichtenklasse	Meldung
253		RFC3692-style Experiment 1	
254		RFC3692-style Experiment 2	

Somit antwortet ein *Host* beispielsweise mit einer *ICMP-Nachricht* vom *Typ* drei und dem *Code* drei, wenn er ein Netzwerkpaket erhält und dieses an einen nicht geöffneten *Port* gerichtet ist.

Eine andere Möglichkeit der Verwendung von *ICMP* ist *Traceroute*. Hierbei wird zuerst ein Paket mit einer *TTL* von eins verschickt, dies wird im Netzwerk genau bis zum ersten Netzknoten geleitet. Dort angekommen, erkennt das jeweilige Netzwerkgerät, dass die *TTL* nach dem Dekrementieren abgelaufen ist und sendet daher ein *ICMP-Paket* an den Absender zurück. Dieses Paket enthält den *Typ* 11 und den *Code* null «Time to Live exceeded in Transit». Die *Traceroute-Routine* kennt somit nun den ersten Netzknoten, welchen die Datenpakete passieren. Die *TTL* wird anschließend auf den Wert zwei inkrementiert, um den nachfolgenden Netzknoten ausfindig zu machen. Sobald das ausgesendete Paket mit einer *ICMP-Nachricht* vom *Typ* drei und dem *Code* drei «Port Unreachable» beantwortet wird, weiß die *Traceroute-Routine*, dass das Paket beim Empfänger angekommen ist. Mit diesem Verfahren kann auf einfache Weise der Weg eines Pakets durch ein Netzwerk festgestellt werden. Allerdings muss an diesem Punkt erwähnt werden, dass es nicht ausgeschlossen ist, dass die Pakete einer *Traceroute* im Netzwerk unterschiedliche Wege nehmen und somit das Ergebnis eventuell verfälscht wird.

Eine weitere, sehr bekannte Anwendung von *ICMP* ist der *Ping* in Netzwerken. Hierzu sendet Teilnehmer A eine *ICMP-Nachricht* vom *Typ* acht «Echo Request», welche im Erfolgsfall von Teilnehmer B mit einer Nachricht vom *Typ* null «Echo Reply» beantwortet wird.

A.4.3. Berechnung der Prüfsumme

Die Berechnung der *Prüfsumme* eines *ICMP-Pakets* erfolgt wie beim *Internet Protocol* über Einerkomplement-Arithmetik und kann in Anhang A.3.2.10 nachgelesen werden.

A.5. Transmission Control Protocol

Wie bisher gezeigt, besitzt das *Internet Protocol* keinerlei Vorkehrungen, um eine zuverlässige Datenübertragung zu garantieren. Auf IP-Ebene können Pakete verloren gehen, in falscher Reihenfolge

ANHANG A. NETZWERKGRUNDLAGEN

Anwendungsschicht		HTTP	SNMP	
Transportschicht		TCP	UDP	
Internetschicht	ICMP			
Internetschicht	IP			ARP
Netzzugangsschicht	Ethernet			

Tabelle A.18.: Position von TCP im TCP/IP-Referenzmodell

zugestellt werden oder unter Umständen sogar doppelt ankommen. Um diese «Schwächen» von *IP* zu umgehen, wird das *Transmission Control Protocol* (*TCP*), welches nach RFC793 [97], RFC896 [3], RFC1122 [15], RFC1323 [63] und RFC2581 [4] definiert ist, eingesetzt. Es repräsentiert die *OSI-Schicht* vier und setzt somit direkt auf *IP* auf, was auch Abbildung A.18 zu entnehmen ist.

Es handelt sich bei *TCP* um ein verbindungsorientiertes, bidirektionales und vollduplexes Protokoll.

Bei *TCP* wird jede Verbindung eindeutig durch zwei Endpunkte identifiziert. Solch ein Endpunkt besteht aus einer eindeutigen *IP-Adresse* und einem *Port*. Diese beiden Angaben haben die folgenden Aufgaben:

- IP-Adresse

Sie identifiziert die beiden Rechner im Netzwerk eindeutig, vergleichbar mit einer Postanschrift.

- Port

Bei dem *Port* handelt es sich um eine 16-Bit Zahl. Diese Angabe erlaubt eine weitere Adressierung innerhalb des Zielgeräts, vergleichbar mit einer Abteilungsbezeichnung auf einem Brief. Durch *Ports* können die ankommenden Daten den verschiedenen Anwendungen beziehungsweise Diensten zugestellt werden. Ein Webserver läuft standardmäßig zum Beispiel auf *Port* 80.

Die *Ports* 0 – 1023 sind durch die IANA [51] reservierte «well known ports». Die Verwendung dieser *Ports* ist jedoch nicht bindend.

ANHANG A. NETZWERKGRUNDLAGEN

Wenn *TCP* Daten empfängt, können diese Daten bereits durch darunterliegende Ebenen mit Fehlern versehen sein. Diese Übertragungsfehler resultieren beispielsweise aus den Übertragungseigenschaften von *Ethernet* oder dem *Internet Protokoll*. *TCP* besitzt nun Vorkehrungen, um mit den nachfolgend genannten Fehlern umgehen zu können.

1. Datenpakete können verloren gehen

Dieses Problem kann *TCP* dadurch beheben, dass jedes *Datensegment* eine eindeutige *Sequenznummer* erhält und nach dem Versand bestätigt werden muss. Bleibt diese Bestätigung aus, wird das Paket erneut versendet.

2. Der eigentliche Dateninhalt kann durch Störungen verändert werden

Sollten auf *Ethernet-Ebene* einige Bits im Datenbereich verändert werden, würde *IP* dies nicht registrieren. *IP* verfügt nur über eine einfache Checksumme über den *IP-Header*. *TCP* dagegen verfügt über eine aufwendigere Checksummenprüfung über das komplette *TCP-Paket*. Somit können fehlerhafte Daten erkannt und verworfen werden.

3. Die Datenpakete können in einer anderen Reihenfolge beim Empfänger ankommen

Durch die Routingeigenschaften auf *IP-Ebene* können Datenpakete unterschiedliche Wege durch Netzwerke nehmen und somit unterschiedlich lange unterwegs sein. Dadurch können die Pakete beim Empfänger in einer anderen Reihenfolge eintreffen als sie beim Sender abgeschickt wurden.

Da *TCP* jedem *Datensegment* eine eindeutige *Sequenznummer* zuordnet, können die empfangenen Pakete beim Empfänger problemlos wieder in die richtige Reihenfolge gebracht werden.

4. Das gleiche Datenpaket kann mehrmals beim Empfänger ankommen

Unter Umständen kann ein Datenpaket auch mehrmals beim Empfänger ankommen. *TCP* erkennt dies ebenfalls durch die *Sequenznummer* im *Header* und kann somit bei mehrfachem Empfang eines Pakets die «Kopien» verwerfen.

5. Unterhalb von TCP gibt es keinerlei Netzwerküberlast- und Flusskontrolle

In den *OSI-Schichten* unterhalb von *TCP* sind keine Einrichtungen vorgesehen, welche den Datenfluss im Netzwerk anpassen. Somit kann es passieren, dass ein Empfänger mit Daten «überschwemmt» wird oder eine Netzwerkverbindung durch zu viele Datenpakete «verstopft». *TCP* besitzt hierfür Vorkehrungen, auf die später näher eingegangen wird.

ANHANG A. NETZWERKGRUNDLAGEN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Source-Port																Destination-Port															
Sequence-Number																															
Acknowledge-Number																															
Data Offset		Reserved						URG	ACK	PSH	RST	SYN	FIN	Window-Size																	
Checksum																Urgent-Pointer															
Options (if any, variable length)																								Padding							
Data (if any)																															

Abbildung A.7.: Aufbau des TCP-Headers

Es ist noch zu erwähnen, dass *TCP* oft fälschlicherweise als *TCP/IP-Protokoll* bezeichnet wird. Korrekterweise kann höchstens die komplette Protokollfamilie als *TCP/IP-Protokollfamilie* beziehungsweise das Schichtenmodell als *TCP/IP-Referenzmodell* bezeichnet werden, nicht aber das *Transmission Control Protocol*.

A.5.1. Aufbau des TCP-Headers

Im Datenbereich des *IP-Pakets* liegt das *TCP-Paket*. Dieses besitzt, genau wie das *IP-Paket*, einen eigenen *Header*. Dieser *Header* hat normalerweise eine Länge von 20 Bytes und ist in Abbildung A.7 dargestellt. Nach den 20 Bytes eines Standard-TCP-Headers können bei Bedarf noch zusätzliche Optionen eingefügt werden. Diese führen dann allerdings zu einer Vergrößerung des *Headers* auf maximal 60 Bytes. Nach dem *Header* folgen die eigentlichen Nutzdaten im *TCP-Paket*.

Die maximale Länge eines *TCP-Pakets* wird durch die darunterliegende Schicht (*IP*) begrenzt. Das *Internet Protocol* ermöglicht eine maximale Nutzdatenlänge von 65535 Bytes. Da das *Internetprotokoll* aber wiederum in das *Ethernetprotokoll* eingebettet ist, welches nur eine maximale Nutzdatenlänge von 1500 Bytes erlaubt, wird die mögliche Datenlänge verringert. Die somit für ein TCP-Paket maximal mögliche Nutzdatenlänge errechnet sich aus 1500 Bytes (Ethernet) – 20 Bytes (IP-Header) – 20 Bytes (TCP-Header) = 1460 Bytes. Somit liegt die *Maximum Segment Size (MSS)* nach RFC879 [102] bei 1460 Bytes.

Die Angaben im *TCP-Header* haben die folgenden Bedeutungen:

A.5.1.1. Source-Port

Beim *Source-Port* handelt es sich um den *Port* auf der Senderseite.

ANHANG A. NETZWERKGRUNDLAGEN

A.5.1.2. Destination-Port

Der *Destination-Port* ist der *Port* auf der Empfangsseite.

A.5.1.3. Sequence-Number

Die *Sequence-Number* kann zwei Aufgaben haben. Zum Einen wird sie dazu verwendet, die Sequenznummer des ersten Daten-Oktetts des aktuellen TCP-Pakets anzugeben, um die Daten beim Empfänger wieder richtig anordnen zu können. Zum Anderen wird sie beim Verbindungsaufbau verwendet. Sobald das *SYN-Flag* (*Synchronisation (SYN)*) gesetzt ist, gibt die *Sequence-Number* die *Initial Sequence Number (ISN)* an.

A.5.1.4. Acknowledgement-Number

Dieses Feld gibt die *Acknowledgement-Number* an. Dabei handelt es sich um die Sequenznummer, welche die Gegenstelle als nächstes erwartet. Dieses Feld ist nur gültig, wenn das *ACK-Flag* (Acknowledgement (ACK)) gesetzt ist.

A.5.1.5. Data Offset

Der *Data Offset* gibt die Länge des *TCP-Headers* in 32-Bit-Blöcken an und adressiert somit den Beginn der Nutzdaten im *TCP-Paket*.

A.5.1.6. Reserved

Dieses Feld wird normalerweise nicht verwendet und standardmäßig sind die Bits alle auf null gesetzt.

A.5.1.7. Flags

A.5.1.7.1. URG Das *Urgent-Flag* ist ein Dringlichkeitsbit. Sobald dieses Bit gesetzt ist, bricht der Empfänger sofort die Verarbeitung seines aktuellen Datenpakets ab und liest die Daten aus, auf welche der *Urgent-Pointer* zeigt. Diese Unterbrechungsanforderung kann mit einem Softwareinterrupt verglichen werden.

ANHANG A. NETZWERKGRUNDLAGEN

A.5.1.7.2. ACK Das *Acknowledgement-Flag* hat bei einer TCP-Datenübertragung zum Einen die Aufgabe, übertragene Segmente zu bestätigen. Zum Anderen wird das Flag bei der Initialisierung einer TCP-Verbindung in Kombination mit dem *SYN-Flag* für den «Drei-Wege-Handshake» verwendet.

A.5.1.7.3. PSH Das *Push-Flag* weist den Empfänger an, die empfangenen Daten direkt an die Anwendung weiterzuleiten, ohne diese so lange im Empfangspuffer zu behalten, bis das komplette *Datensegment* empfangen wurde. Dies resultiert bei zeitkritischen Anwendungen in einer kürzeren Reaktionszeit.

A.5.1.7.4. RST Das *Reset-Flag* wird zum Abbrechen/Abweisen einer Verbindung verwendet.

A.5.1.7.5. SYN Dieses Bit dient zur Synchronisation der *Sequenznummer* während des «Drei-Wege-Handshakes» beim Verbindungsaufbau.

A.5.1.7.6. FIN Mit diesem *FIN-Flag* (*Finish (FIN)*) signalisiert der Sender, dass er keine weiteren Daten zur Übertragung bereit hält und die Verbindung beenden möchte.

A.5.1.8. Window-Size

Mit der Angabe im Feld *Window-Size* teilt der Sender des Pakets mit, wie viele Daten-Oktetts er bereit ist zu empfangen. Die Größenangabe beginnt dabei mit dem durch das *Acknowledgement-Number* indizierten Daten-Oktett. Dieses Feld wird für die später behandelte Flusskontrolle verwendet.

A.5.1.9. Checksum

Zum Überprüfen der empfangenen Daten beinhaltet der *TCP-Header* eine 16-Bit-Prüfsumme über das ganze *TCP-Segment*, welche über das Einerkomplement berechnet wird. Hierzu wird dem eigentlichen *TCP-Header* zuerst ein *Pseudoheader* vorangestellt. Anschließend werden in die Prüfsummenberechnung die einzelnen 16-Bit-Werte des Pseudoheaders mit einbezogen. Sollte die Anzahl der Bytes im TCP-Paket ungerade sein, wird dem Paket ein Füllbyte der Form *0x00* angehängt. Der *Pseudoheader* und das eventuelle Füllbyte werden später jedoch nicht mit übertragen. Der Datenblock, über welchen die *TCP-Prüfsumme* berechnet wird, ist aus Abbildung A.8 ersichtlich.

ANHANG A. NETZWERKGRUNDLAGEN

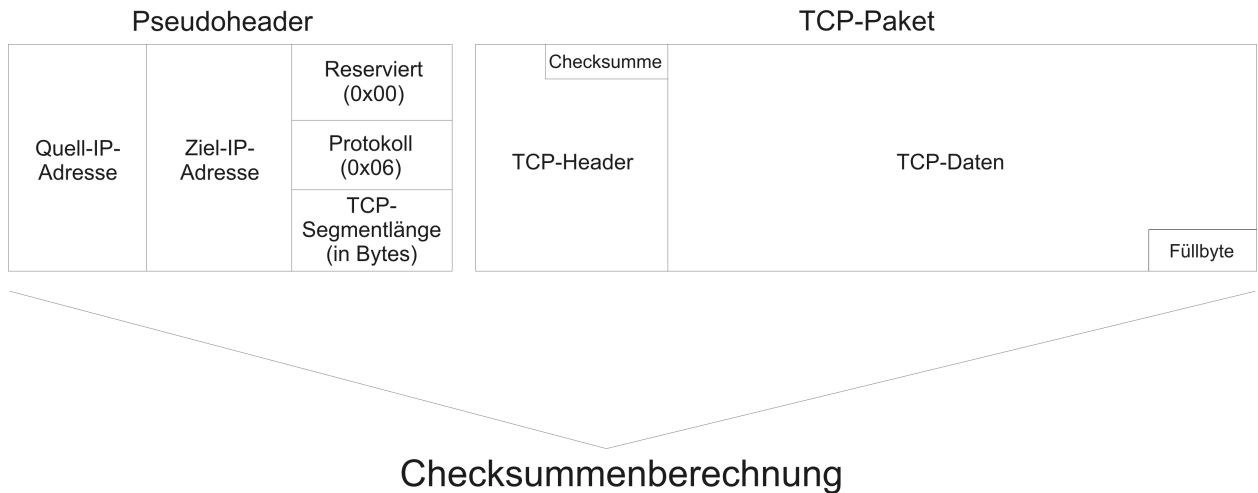


Abbildung A.8.: Berechnung der TCP-Checksumme

Zur Berechnung der Checksumme wird das Checksummenfeld zuerst auf Null gesetzt, anschließend werden alle 16-Bit-Worte addiert. Sollte dabei ein Überlauf entstehen, wird dieser am Ende zum 16-Bit-Wert addiert. Das Ergebnis wird invertiert und die daraus erhaltene Prüfsumme in den *TCP-Header* eingefügt.

Zur Kontrolle eines empfangenen Datenpakets wird die gleiche Berechnung durchgeführt, allerdings wird die empfangene Checksumme in diesem Fall mit verrechnet. Das Ergebnis ist bei einer fehlerfreien Übertragung *0xFFFF*.

Die *TCP-Checksumme* wird im Prinzip genau wie die *IP-Checksumme* gebildet, mit dem Unterschied, dass bei *TCP* der *Pseudoheader*, die Nutzdaten und eventuell ein Füllbyte mit eingerechnet werden.

A.5.1.10. Urgent-Pointer

Dieses Feld enthält einen Pointer auf diejenigen Daten, welche sofort verarbeitet werden sollen, wenn das *URG-Flag* gesetzt ist und das Paket empfangen wird. Die Nummer des letzten Bytes der dringenden Daten errechnet sich aus der Summe von *Urgent-Pointer* und *Sequence-Number*.

A.5.1.11. Options

In diesem Feld können an den *TCP-Header* Zusatzinformationen angehängt werden. Diese Optionen werden meist zur Steuerung des Datenflusses und der Netzwerkauslastung verwendet. Jede Option

Optionstyp	Länge des Optionsfelds	Verwendung
0	nicht vorgesehen	Ende der Optionsliste
1	nicht vorgesehen	No Operation
2	4 Byte	Maximum Segment Size (MSS)
3	3 Byte	Window Scale (WSopt)
4	2 Byte	SACK erlaubt
5	variabel	SACK
8	10 Byte	Time-Stamp-Abgleich
11	6 Byte	Connection Count CC (T/TCP)
12	6 Byte	CC.NEW (T/TCP)
12	6 Byte	CC.ECHO (T/TCP)

Tabelle A.19.: Optionen des TCP-Headers

muss dabei 32 Bits belegen, sonst wird diese mit Füllbits (Null-Bits) aufgefüllt.

Die möglichen Optionen sind durch RFC1323 [63] und RFC2018 [81] spezifiziert und unter «Transport-Protokolle TCP und UDP» [70] nochmals ausführlich erläutert. Das erste Byte einer jeden Option definiert den Optionstyp. Die möglichen Optionstypen mit den dazugehörigen Längenangaben des Optionsfelds und ihrer Bedeutung sind Tabelle A.19 zu entnehmen.

Diese Optionen besitzen folgende Bedeutung:

- Maximum Segment Size (MSS)

Mit dieser Option kann dem Verbindungspartner die maximale Segmentgröße mitgeteilt werden.

- Window Scale (WSopt)

Mittels *Window Scale* kann während der Initialisierung einer Verbindung ausgehandelt werden, dass die *Window Size* mit einem festen Skalenwert von maximal 14 multipliziert wird. Somit kann die Fenstergröße auf eine Größe von 32 Bits anwachsen. Dieser Faktor kann für beide Kommunikationsteilnehmer getrennt ausgehandelt werden.

ANHANG A. NETZWERKGRUNDLAGEN

- Timestamps Option (TSopt)

Diese Option besteht aus dem *Timestamp Wert* (TSval) und dem bei *ACK-Paketen* vorhandenen *Timestamp Echo Reply* (TSecr). Mit Hilfe dieser Angaben informieren sich die beiden Endgeräte über die so genannte *Round Trip Time*.

- Selective Acknowledgement

Dieses Feld steht laut RFC2018 [81] für selektive Paketbestätigungen zur Verfügung.

- T/TCP

Dieses Optionsfeld steht nach RFC1379 [16] für *Transactional TCP* (T/TCP) zur Verfügung. Dabei handelt es sich um eine TCP-Erweiterung für effiziente kurze TCP-Verbindungen mit geringem Datenvolumen.

A.5.1.12. Padding

Dieses Feld enthält Füllbits. Diese werden verwendet, wenn das *Options-Feld* kein Vielfaches von 32 Bits lang ist. In diesem Fall wird mit dem *Padding-Feld* auf 32 Bits aufgefüllt.

A.5.1.13. Data

An dieser Stelle stehen die eigentlichen zu übertragenden Daten des *TCP-Pakets*.

A.5.2. Verbindungsauf-/abbau

Da es sich bei *TCP* um ein verbindungsorientiertes Protokoll zwischen zwei Endgeräten handelt, muss vor einer Datenübertragung zuerst eine virtuelle Verbindung zwischen diesen Endgeräten initiiert werden. Dieser Vorgang ist in «TCP/IP Illustrated» [68] und «Konzepte der Internet-Technik» [127] erklärt.

Der Server, also der Rechner, der einen Dienst zur Verfügung stellt, besitzt eine eindeutige *IP-Adresse* und stellt seinen Dienst auf einem bestimmten *Port* zur Verfügung. Solange keine Verbindung besteht, befindet sich der Server im Modus «passive open» beziehungsweise «listen».

Der Client, welcher eine Verbindung aufbauen möchte («active open»), generiert nun seinen eigenen Endpunkt mit Hilfe seiner *IP-Adresse* und einer noch freien *Portnummer*. Anschließend wird der

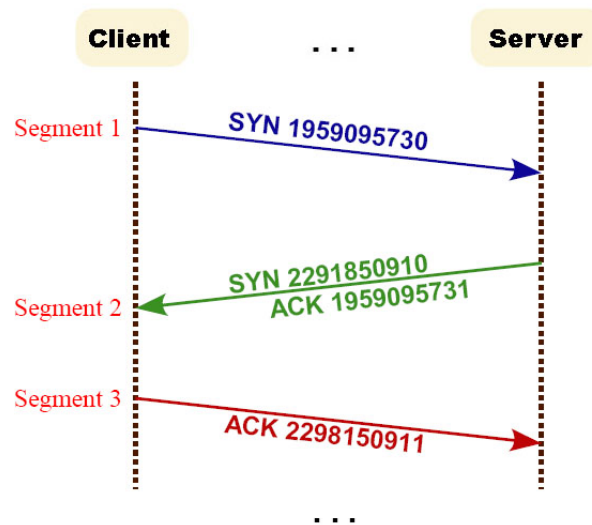


Abbildung A.9.: Ablauf eines Drei-Wege-Handshakes [68]

Verbindungsaufbau mit einem «Drei-Wege-Handshake» laut Abbildung A.9 gestartet. Dieser *Drei-Wege-Handshake* hat den Vorteil, dass sich beide Teilnehmer gegenseitig den Empfang der Pakete bestätigen und somit sicher sein können, dass die Gegenstelle die Informationen erhalten hat.

Hierbei sendet der Client zuerst ein Paket mit gesetztem *SYN-Flag* und einer beliebigen *Sequenznummer* x . Nutzdaten enthalten diese *SYN-Pakete* standardmäßig nicht. Auf dieses «active open» folgt eine von drei Möglichkeiten:

- Der Server akzeptiert die Verbindung

Akzeptiert der Server die Verbindung, so antwortet er mit einem Paket, in welchem das *SYN-Flag* gesetzt ist. Weiterhin bekommt dieses Paket vom Server eine beliebige *Sequenznummer* y zugeteilt, das *ACK-Flag* im Paket wird gesetzt und die *Acknowledgement-Number* erhält den Wert $x+1$.

- Der Server weist die Anfrage ab

Weist der Server die Verbindung ab, so sendet er ein TCP-Paket mit gesetztem *RST-Flag* zurück.

- Das SYN-Paket geht verloren

Das genaue Verhalten bei einem verlorenen SYN-Paket variiert von Betriebssystem zu Betriebssystem. Bei der *Berkeley Software Distribution (BSD)* [95] muss beim Client spätestens

ANHANG A. NETZWERKGRUNDLAGEN

sechs Sekunden nach dem Senden des SYN-Pakets eine Antwort vom Server eingehen, andernfalls wird das Paket erneut versendet. Sollte daraufhin wieder keine Antwort beim Client eingehen, so wird nach weiteren 24 Sekunden ein drittes Mal das SYN-Paket verschickt. Wenn anschließend 75 Sekunden nach dem ersten SYN-Paket keine Antwort eingetroffen ist, wird der Verbindungsversuch abgebrochen. Im Gegensatz dazu werden unter *Debian Linux* [22] insgesamt sechs SYN-Pakete versendet. Der Zeitabstand zwischen zwei Paketen wird nach jedem weiteren Paket verdoppelt, beginnend bei drei Sekunden nach dem ersten SYN-Paket. Wenn 96 Sekunden nach dem sechsten Paket noch immer keine Antwort beim Client eingetroffen ist, beendet dieser den Verbindungsaufbau.

Sollte der Server die Verbindung angenommen und dem Client geantwortet haben, so sendet der Client erneut ein Paket zum Server, um zu zeigen, dass das *Acknowledge* ankam. Hierzu wird im Paket das *ACK-Flag* gesetzt und die *Acknowledgement-Number* erhält den Wert $y+1$.

Bei der Wahl der *Startsequenznummern* (*ISN*) sollten aus Sicherheitsgründen Zufallswerte verwendet werden. Damit sinkt die Wahrscheinlichkeit, dass eine neu aufgesetzte Verbindung Pakete empfängt, welche noch von einer früheren Verbindung im Netz unterwegs sind. Diese Zufallswerte können zum Beispiel dadurch generiert werden, dass ein 32-Bit-Zähler ab Systemstart alle a Millisekunden um b inkrementiert wird. Wenn der Zähler überläuft, beginnt dieser wieder bei null. Als *ISN* kann nun zum Verbindungszeitpunkt einfach der aktuelle Zählerstand verwendet werden.

Während der TCP-Verbindung («active open») sind beide Teilnehmer gleichberechtigt, somit kann zum Ende der Verbindung auch jeder der beiden Teilnehmer einen Verbindungsabbau (Abbildung A.10) einleiten.

Dabei gibt es zwei Arten von Verbindungsabbau:

- Vier-Wege-Verbindungsabbau

Hierbei initiiert Teilnehmer eins einen Verbindungsabbau «close», indem er ein *TCP-Segment* mit gesetztem *FIN* Flag sendet. Teilnehmer zwei empfängt dieses Paket und antwortet darauf mit einem *Segment*, in welchem das *ACK* Flag gesetzt ist. Anschließend sendet Teilnehmer zwei ein Paket zu Teilnehmer eins, in welchem das *FIN* Flag gesetzt ist, und bekommt ein Paket mit gesetztem *ACK* zurück. Die Verbindung ist anschließend von beiden Seiten beendet. Zur Optimierung der übertragenen Datenmenge kann Teilnehmer zwei in seinem ACK-Paket für Teilnehmer eins mit dem gesetzten FIN-Flag auch den eigenen Verbindungsabbau einleiten. Somit wird die Übertragung von einem TCP-Paket eingespart.

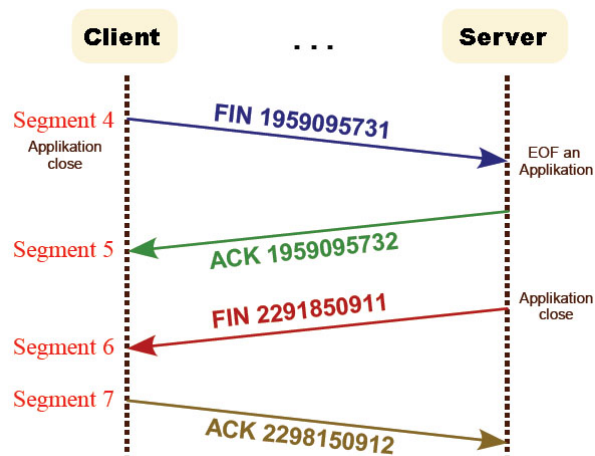


Abbildung A.10.: Vier-Wege-Verfahren zum Verbindungsabbau [68]

- Half-Close

Als *Half-Close* wird eine Verbindung bezeichnet, in welcher Teilnehmer eins die Verbindung beendet, Teilnehmer zwei aber weiterhin Daten senden möchte. In diesem Fall antwortet Teilnehmer zwei mit einem *ACK* auf das *FIN* von Teilnehmer eins, sendet jedoch kein eigenes *FIN*. Stattdessen werden weiterhin die Nutzdaten übertragen und ganz am Ende kann Teilnehmer zwei mit einem *FIN* die «halb offene» Kommunikation beenden. Anschließend antwortet Teilnehmer eins mit einem letzten *ACK*, um das Schließen der Verbindung zu bestätigen.

Nach der Übertragung des letzten *ACK* vom Server zum Client wartet der Client eine Zeit von zwei *Maximum Segment Lifetime (MSL)*. Anschließend werden alle weiteren ankommenden Pakete verworfen. Somit können verspätete Pakete nicht als Teil einer neuen Verbindung gewertet werden.

A.5.3. Datenübertragung

Wie bereits erwähnt, liegt die maximal mögliche *MSS* eines *TCP-Pakets* bei 1460 Bytes. Um dennoch ein größeres Datenpaket übertragen zu können, muss dieses segmentiert werden. Soll beispielsweise ein zehn Kilobytes großer Datenblock über eine Verbindung mit einer *MSS* von 1460 Bytes übertragen werden, so müssen die Nutzdaten in 1460 Bytes große Blöcke aufgeteilt werden. Jeder dieser Blöcke bekommt von *TCP* einen eigenen *Header* und eine eigene, eindeutige *Sequenznummer* zugewiesen.

ANHANG A. NETZWERKGRUNDLAGEN

Der Empfänger bestätigt dem Sender die angekommenen Pakete, bei welchen die *CRC-Prüfung* gültig war. Bei dieser Bestätigung muss der Empfänger jedoch nicht jedes Paket einer zusammenhängenden Paketfolge bestätigen, sondern es reicht, wenn nur jeweils das letzte bestätigt wird. Kommen also die Segmente 1 bis 5 beim Empfänger an, so reicht es, wenn nur Paket 5 bestätigt wird. Kommen dagegen die Pakete 1,2,4 und 5 an, so kann der Empfänger nur die Pakete 1 und 2 bestätigen, da die zusammenhängende Paketfolge durch das fehlende Paket 3 unterbrochen wird.

Der Sender startet beim Versenden jedes Pakets einen internen Timer mit der *Round Trip Time (RTT)*. Läuft dieser Timer für ein Paket ab, da kein *Acknowledge* empfangen wurde, so wird dieses Paket erneut versendet. Auf diese Weise werden diejenigen Pakete erneut übertragen, welche der Empfänger nicht bestätigt hat.

Zu Beginn einer Verbindung können sich beide Kommunikationsteilnehmer gegenseitig ihre *MSS* mitteilen. Dies geschieht über das *Options-Feld* im *TCP-Header*.

A.5.4. Flusssteuerung

Um den *Traffic* in einem Netzwerk anzupassen, gibt es unterschiedliche Verfahren zur Flusssteuerung, die jedoch zusammenarbeiten können.

Ein verwendetes Verfahren ist die Verwendung eines *Sliding Window*. Wird eine Verbindung über einen Satelliten oder ein interkontinentales Seekabel hergestellt, sind die Datenpakete vom Sender zum Empfänger relativ lange unterwegs. Wenn der Sender nun jedes Mal auf ein *Acknowledge* vom Empfänger warten müsste, bevor er neue Daten sendet, so würde dies die Datenrate stark verringern.

Stattdessen wird ein *Sliding Window* verwendet. Hierbei teilt der Empfänger dem Sender mit, wie viel freier Platz im Empfangspuffer vorhanden ist. Anschließend generiert der Sender ein virtuelles *Sliding Window* über die zu sendenden Daten und schickt eine angepasste Anzahl von Datenpaketen auf die Reise. Sobald der Empfänger die Daten erhält, bestätigt er diese. Dabei reicht es, wenn in einer fortlaufenden Kette von empfangenen Paketen das letzte bestätigt wird. Der Sender wiederum verschiebt sein *Sliding Window* um so viele Pakete wie bestätigt wurden. Auf diese Weise wird eine Verbindung mit einer großen *RTT* dennoch optimal ausgelastet. Weiterhin wird verhindert, dass der Puffer im Empfänger überläuft und somit Daten verworfen werden. Eine Ausnahme bilden allerdings Segmente mit gesetztem *URG-Flag*. Da diese nicht den Empfangspuffer passieren und stattdessen vom Empfänger sofort verarbeitet werden, können diese auch bei vollem Empfangspuffer entgegengenommen werden.

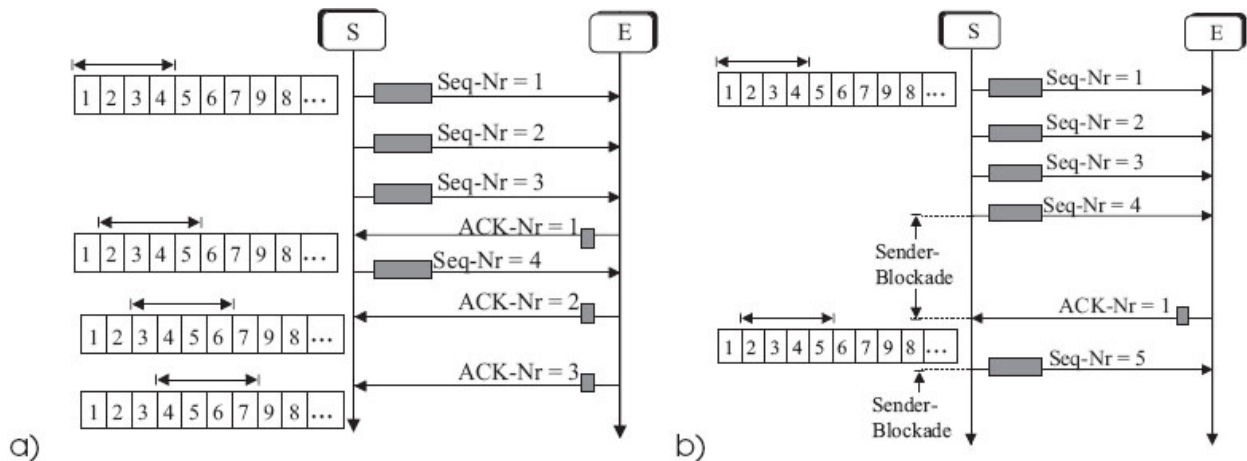


Abbildung A.11.: Verwendung eines Sliding Window [70]

In Abbildung A.11 hat das *Sliding Window* eine Größe von vier Bytes. Der Sender darf somit maximal vier Segmente abschicken, ohne auf eine Bestätigung warten zu müssen. In Teil a) der Abbildung werden die Bytes eins bis drei gesendet. Anschließend erhält der Sender ein *Acknowledge* von Byte eins. Dadurch wird der Beginn des *Sliding Window* auf Byte zwei gelegt. Das vier Bytes große Fenster reicht somit bis zu Byte fünf. Der Sender sendet daher nun Byte vier und Byte fünf. Nach dem Senden von Byte vier erreicht den Sender allerdings erneut ein *Acknowledge*, diesmal von Byte zwei. Das *Sliding Window* wird wieder verschoben.

In Teil b) von Abbildung A.11 ist der gleiche Übertragungsfall wie in Teil a) dargestellt. Allerdings erreichen die *ACK-Pakete* den Sender erst nach einer gewissen Verzögerung. Aus der Abbildung geht hervor, dass der Sender beim Senden die komplette Größe des erlaubten *Sliding Window* ausnutzt und daher vier Pakete auf die Reise schickt. Da der Sender aber kein *Acknowledge* erhält, dürfen die nachfolgenden Bytes nicht gesendet werden, der Sender wartet. Wenn das *Acknowledge* von Byte eins beim Sender eintrifft, verschiebt dieser sein *Sliding Window* und schickt Byte fünf auf die Reise.

A.5.4.1. Silly Window Syndrome

Bei der Flusssteuerung mittels *Sliding Window* existiert jedoch ein als *Silly Window Syndrome* bekanntes Problem. Der Empfänger meldet mit *Zero Window*, dass sein Empfangspuffer voll ist. Daraufhin stellt der Sender die Übertragung ein. Nun werden im Puffer des Empfängers beispielsweise zwei Bytes frei. Der Empfänger teilt dem Sender mit einer neuen Nachricht mit, dass zwei Bytes gesendet werden können. Der Sender verpackt diese zwei Bytes zusammen mit den 40 Bytes

ANHANG A. NETZWERKGRUNDLAGEN

des *Headers* in ein *TCP-Paket* und versendet es. Der Empfangspuffer ist somit wieder komplett gefüllt und der Empfänger meldet erneut *Zero Window*. Dieser Vorgang wiederholt sich nun bei jedem kleinsten Anteil freien Speichers im Empfangspuffer. Dies ist sehr ressourcenverschwendend, da zwei Nutzdatenbytes mit 40 Headerbytes verpackt werden und dazu noch die «*Zero Window*»-Messages des Empfängers übertragen werden.

Um dieses Problem zu optimieren, gibt es zwei Algorithmen. Zum Einen die Lösung von *Dave Clark*, welche aussagt, dass der Empfänger nach einem *Zero Window* erst wieder die *Window Size* aktualisieren darf, wenn eine gewisse Menge freier Speicher zur Verfügung steht. Hierfür gibt es zwei Schwellen. Die eine Schwelle besagt, dass mindestens die *MSS* im Puffer frei werden muss. Die andere Schwelle besagt, dass der Puffer mindestens halb leer werden muss. Die Schwelle, welche zuerst erreicht wird, triggert die Aktualisierung des Übertragungsfensters. Zur Behebung des *Silly Window Syndroms* auf der Senderseite existiert der so genannte *Nagle-Algorithmus*, bei welchem ein Paket erst versendet wird, wenn es vollständig mit Daten gefüllt ist. Wenn das Füllen des Pakets nicht möglich ist, wird das Paket erst dann versendet, wenn keine weiteren unbestätigten Pakete mehr unterwegs sind.

A.5.4.2. Slow Start

Zu Beginn einer Datenübertragung besitzt der Sender keinerlei Informationen über die momentane Auslastung des Netzwerks. Da diese Informationen jedoch für eine optimale Flusssteuerung benötigt werden, muss der Sender versuchen, die momentane Netzwerkauslastung selbstständig zu bestimmen. Hierzu verwendet er ein sogenanntes *Congestion Window*. Die Größe dieses Fensters wird durch das «*Slow Start*»-Verfahren bestimmt. Hierbei setzt der Sender das *Congestion Window* zu Beginn beispielsweise auf die Größe von einem Paket. Mit jeder Empfangsbestätigung, die nach der *RTT* beim Sender eingeht, wird das *Congestion Window* um eins inkrementiert. Dies bedeutet für den Sender, dass er nach einem bestätigten Paket in der nächsten Runde mit zwei Paketen, also einem doppelt so großen Fenster, arbeiten kann. Werden daraufhin wieder alle Pakete, also zwei Stück, bestätigt, so wächst das Fenster auf eine Größe von vier Paketen an. Die Größe des *Congestion Window* verläuft somit exponentiell.

Sobald die Schwelle *Slow Start Threshold* erreicht wurde, wird das *Congestion Window* nur noch um eins inkrementiert, sobald alle Pakete des Fensters übertragen wurden. Das Fenster wächst in dieser als *Congestion Avoidance* bezeichneten Phase nur noch linear an. Sobald der Sender für ein oder mehrere Pakete keine Bestätigung mehr erhält (Timeout), geht er davon aus, dass das Netzwerk überlastet ist. In diesem Fall wird das *Congestion Window* sofort auf den Wert eins zurückgesetzt

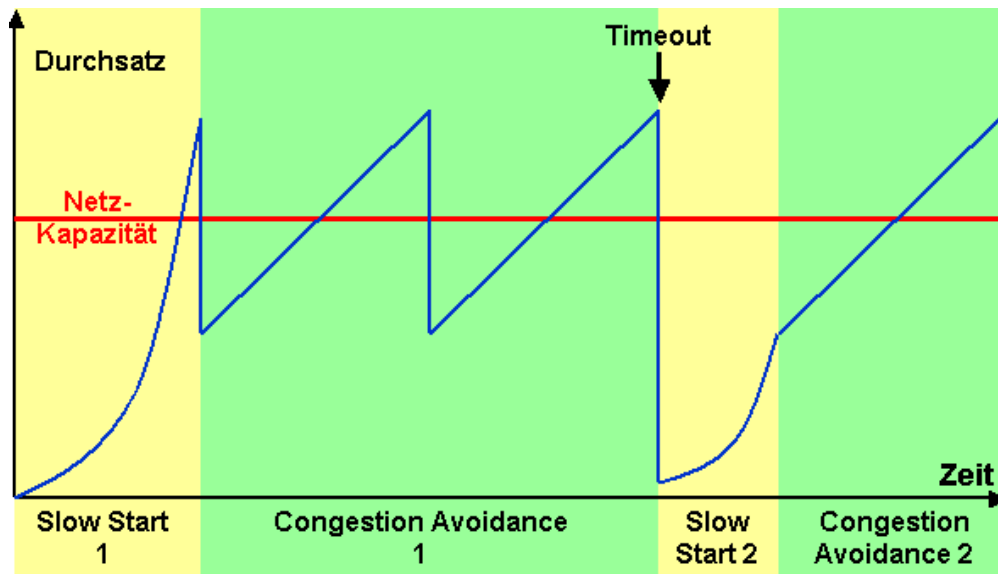


Abbildung A.12.: Verwendung des Slow-Start-Verfahrens [127]

und die *Slow Start Threshold* wird auf die Hälfte des vorigen Wertes gesetzt. Anschließend beginnt das Vergrößern des *Congestion Window* von Neuem. Wenn anstatt eines Timeouts jedoch nur Bestätigungswiederholungen eines oder mehrerer Pakete eintreffen, sogenannte *Duplicate Acknowledgements*, so geht der Sender davon aus, dass das Netzwerk gerade an der Überlastgrenze betrieben wird. In diesem Fall wird das *Congestion Window* nur halbiert und anschließend wieder per *Congestion Avoidance* vergrößert. Diese Vorgänge zur Anpassung des *Congestion Window* sind in Abbildung A.12 nochmals schematisch dargestellt.

A.5.4.3. Fast-Retransmit und Fast-Recovery

Gehen Pakete verloren, können *Duplicate Acknowledgements* auftreten. Diese mehrfachen Bestätigungen eines Pakets treten dann auf, wenn beispielsweise der Sender fünf Pakete abschickt und der Empfänger alle Pakete bis auf Paket Nummer drei erhält. In diesem Fall kann der Empfänger die Pakete Nummer vier und fünf nicht bestätigen, da Paket Nummer drei in der Paketfolge fehlt. Stattdessen bestätigt der Empfänger das Paket zwei mehrmals und zwar für jedes folgende Paket. Der Sender erkennt diese *Duplicate Acknowledgements* und wartet daher bei Paket drei nicht auf den Timeout, sondern sendet nach der dritten Mehrfachbestätigung das Paket drei erneut. Daher auch der Name *Fast-Retransmit*. Weiterhin wird durch jede Mehrfachbestätigung auch das *Congestion Window* wieder erhöht. Der Sender erhöht also die Übertragungsgeschwindigkeit, da nur ein Paket verloren ging, die nachfolgenden Pakete aber fehlerfrei zugestellt wurden. Diese Eigenschaft wird *Fast-Recovery* genannt.

A.5.4.4. Kumuliertes Acknowledge

Das eben begonnene Beispiel wird an dieser Stelle fortgesetzt. Der Empfänger hat die Pakete 1, 2, 4 und 5 empfangen. Da Paket drei verloren ging, wurde es vom Sender erneut versendet. Sobald es beim Empfänger eintrifft, bestätigt dieser nur Paket fünf. Dadurch erkennt der Sender, dass bis zu Paket fünf alle Pakete korrekt ankamen. Diese Bestätigung über mehrere Pakete wird als *kumuliertes Acknowledge* bezeichnet.

A.5.4.5. Selektives Acknowledge

Durch *selektive Acknowledges* steht eine noch genauere Datenflusskontrolle zur Verfügung. Der Empfänger fügt hierbei im Optionsfeld des Headers genaue Informationen ein, welche Pakete bisher eingetroffen sind. Der Sender kann dadurch noch gezielter einzelne Pakete erneut versenden. Ein Paket gilt aber weiterhin erst dann als bestätigt, wenn ein normales *Acknowledge* empfangen wurde.

A.5.4.6. Weitere Möglichkeiten der Überlaststeuerung

Wie in diesem Kapitel gezeigt, existieren sehr viele Ansätze zur Überlaststeuerung in Netzwerken. Dennoch wird auf diesem Gebiet weiter geforscht, um die Überlaststeuerung noch besser an die Gegebenheiten anzupassen. Vor allem äußere Einflüsse, wie zum Beispiel bei Drahtlosnetzwerken, führen zu stark schwankenden Laufzeitverzögerungen und Paketverlusten. Neue Methoden zur Flusskontrolle basieren einerseits häufig auf komplexen mathematischen und regelungstechnischen Grundlagen; andererseits müssen die neuen Verfahren zu den alten Kontrollmechanismen kompatibel sein beziehungsweise auf diese «Rücksicht» nehmen. Ansonsten hätten die neueren Verfahren gegenüber den älteren einen sehr großen Vorteil beim «Kampf» um die Bandbreite.

Einige weit verbreitete und als Quasi-Standards angesehene Mechanismen zur Überlaststeuerung sind *TCP-Reno*, *TCP-Tahoe* und *TCP-Vegas*.

Weiterhin gibt es Ansätze wie *Router Congestion Feedback (RCF)*, bei welchem über die Router auf dem Übertragungsweg umfangreiche Informationen gesammelt werden. Mit Hilfe dieser Informationen kann anschließend die Netzwerkauslastung besser gesteuert werden.

Anwendungsschicht		HTTP	SNMP	
Transportschicht		TCP	UDP	
Internetschicht	ICMP			
Internetschicht	IP			ARP
Netzzugangsschicht	Ethernet			

Tabelle A.20.: Position von UDP im TCP/IP-Referenzmodell

A.6. User Datagram Protocol

Beim *User Datagram Protocol* (UDP) nach RFC768 [100] handelt es sich um ein Protokoll der Transportschicht, also der *OSI-Schicht* vier. Im Gegensatz zu *TCP* ist *UDP* jedoch ein minimales, verbindungsloses Protokoll, welches keine unnötigen Verzögerungen aufweist. Zur Adressierung der Anwendungen sind, wie bei *TCP*, *Sockets*² vorhanden, welche durch eine *IP-Adresse* und einen eindeutigen *Port* gebildet werden.

Da *UDP* verbindungslos ist, gibt es allerdings keine Garantie, dass ein Paket auch wirklich beim Empfänger ankommt. Genauso wenig kann wegen fehlender Paketnummerierung garantiert werden, dass die Paketreihenfolge eingehalten wird. Die auf *UDP* basierte Anwendung muss also mit diesen «Schwächen» zurechtkommen. Um die Fehlerfreiheit der empfangenen Daten kontrollieren zu können, besteht bei *UDP* die Möglichkeit, eine Checksumme über das komplette *UDP-Datagramm* zu bilden.

Der große Vorteil von *UDP* liegt darin, dass nicht, wie zum Beispiel bei *TCP* zuerst eine Verbindung aufgebaut werden muss, sondern dass die Daten sofort übertragen werden können. Diese schnelle Datenübertragung wird vor allem bei Multimediaanwendungen, wie Audio- und Videostreaming oder *Voice over IP* (*VoIP*) benötigt. Würde hierzu *TCP* verwendet werden, so könnte die Übertragung bei einem fehlenden Paket ins Stocken kommen. Bei *UDP* hingegen wird das eine Paket lediglich verworfen, was sich nur in einer Qualitätseinbuße bemerkbar macht. Eine Möglichkeit, ein verloren gegangenes Paket erneut zu erhalten, besteht nicht. Dank des geringen Overheads der eigentlichen Nutzdaten eignet sich *UDP* auch hervorragend für einfache Steueraufgaben, bei welchen nur sehr geringe Datenmengen übertragen werden.

UDP ist *multi-* und *broadcastfähig*. Bei *Multicast* kann ein Paket gleichzeitig an mehrere Rechner

²Bei einem *Socket* handelt es sich um eine virtuelle Schnittstelle auf einem System, welche aus der *IP-Adresse* und einem *Port* besteht.

ANHANG A. NETZWERKGRUNDLAGEN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Source-Port																Destination-Port															
Length																Checksum															
Data (if any)																															

Abbildung A.13.: Aufbau des UDP-Headers

gesendet werden, obwohl der Sender es nur einmal absendet. Dies wird bei Audio- und Video-streaming, wie zum Beispiel von *Internet-Radio-Stationen*, verwendet. *Broadcast* hingegen wird dazu verwendet, um alle Rechner in einem Netzwerk zu erreichen. Somit kann ein *UDP-Datagramm* einmal abgesendet und von allen Rechnern im Netzwerk empfangen werden.

A.6.1. Aufbau des UDP-Headers

Genauso wie das *TCP-Paket* befindet sich auch das *UDP-Paket* im Datenbereich des *IP-Pakets*. Da ein *IP-Paket* eine maximale Länge von 65535 Bytes haben kann, stehen für das *UDP-Paket* maximal $65535 \text{ Bytes (IP-Paket)} - 20 \text{ Bytes (IP-Header)} = 65515 \text{ Bytes}$ zur Verfügung. Da das *IP-Paket* aber wiederum in einen *Ethernet-Frame* verpackt wird, reduziert sich die maximale Größe auf $1500 \text{ Bytes (Ethernet)} - 20 \text{ Bytes (IP-Header)} = 1480 \text{ Bytes}$. Der *UDP-Header* hat laut Abbildung A.13 eine minimale Länge von acht Bytes. Für die Nutzdaten in einem *UDP-Paket* bleiben somit maximal $1500 \text{ Bytes (Ethernet)} - 20 \text{ Bytes (IP-Header)} - 8 \text{ Bytes (UDP-Header)} = 1472 \text{ Bytes}$ übrig.

Die einzelnen Bereiche des *UDP-Headers* haben die nachfolgenden Bedeutungen:

A.6.1.1. Source-Port

Dieses zwei Bytes große Feld enthält die Nummer der *Ports* auf der Senderseite. Da es sich bei *UDP* um ein verbindungsloses Protokoll handelt, kann diese Angabe auch eingespart werden. In diesem Fall wird der *Source-Port* einfach auf null gesetzt.

A.6.1.2. Destination-Port

Der Zielport gibt den *Port* auf der Empfängerseite an. Mit seiner Hilfe wird die gewünschte Anwendung auf dem Zielsystem ausgewählt, welche die Daten entgegennehmen soll.

ANHANG A. NETZWERKGRUNDLAGEN

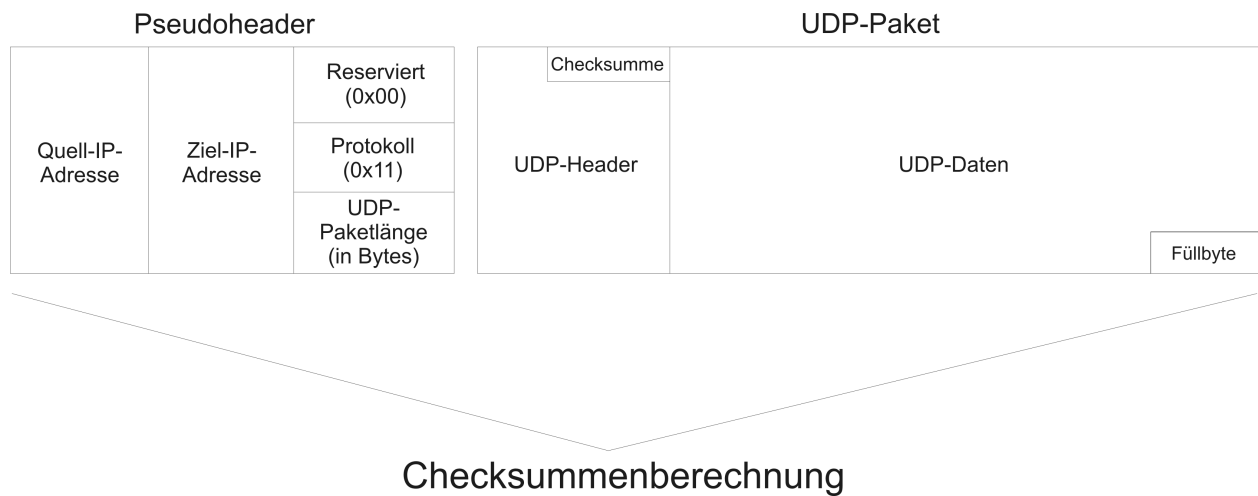


Abbildung A.14.: Berechnung der UDP-Checksumme

A.6.1.3. Length

Dieses Feld beinhaltet die Längenangabe in Bytes über das komplette *UDP-Paket*, also über *Header* und Daten. Da die Daten in einem *UDP-Paket* optional sind, verbleibt eine Mindestlänge von acht Bytes.

A.6.1.4. Checksum

Der *UDP-Header* beinhaltet eine 16-Bit-Prüfsumme über das ganze *UDP-Paket*. Diese Prüfsumme errechnet sich wie bei *TCP* über das Einerkomplement. Zur Berechnung der Prüfsumme wird dem eigentlichen *UDP-Header* ein *Pseudo-Header* vorangestellt. Für den Fall, dass die Anzahl der Bytes der *UDP-Nutzdaten* ungerade ist, wird den Daten noch ein *Nullbyte* als Füllbyte angehängt. Über diesen kompletten Block, wie in Abbildung A.14 dargestellt, wird anschließend die Prüfsumme berechnet. Zur Prüfsummenberechnung wird der komplette Block in 16-Bit große Teile zerlegt, welche anschließend addiert werden. Sollte das Ergebnis einen Überlauf aufweisen, so wird dieser zum 16-Bit-Wert addiert. Das nun erhaltene Ergebnis wird noch invertiert und der daraus resultierende Wert stellt die Prüfsumme dar.

Hierbei ist zu beachten, dass zur Berechnung der Prüfsumme das Prüfsummenfeld zuerst auf *0x0000* gesetzt wird. Bei der Überprüfung eines empfangenen *UDP-Pakets* wird die empfangene Prüfsumme mit eingerechnet. Das Ergebnis der Überprüfung resultiert bei korrekten Daten anschließend im Wert *0xFFFF*.

ANHANG A. NETZWERKGRUNDLAGEN

Nach «Design und Implementierung eines Ethernet-Treibers und TCP/IP-Protokolls für das Java-Betriebssystem JX» [89] ist die Angabe der Prüfsumme optional und kann bei Bedarf durch den Wert null im *Checksum-Feld* ersetzt werden.

A.6.1.5. Data

Bei diesen Daten handelt es sich um die eigentlichen Nutzdaten, welche im *UDP-Paket* transportiert werden.

A.6.2. UDP-Lite

Nach RFC3828 [76] existiert eine «Lite-Version» von *UDP*. Diese vereinfachte Version des Protokolls ist speziell auf zeitkritische Anwendungen ausgelegt, welche nur geringe Verzögerungen erlauben. Hierbei werden Fehler in einem gewissen Rahmen toleriert. *UDP-Lite* ist kompatibel zu *UDP*. Die Vereinfachung besteht darin, dass im Längenfeld des *UDP-Headers* angegeben werden kann, welche Daten mit einer Prüfsumme geprüft werden sollen und welche nicht. Um welche Art von *UDP-Datagramm* es sich handelt, ist daran erkennbar, dass die *Total Length* im *IP-Header* mit der Längenangabe im *UDP-Header* verglichen wird. Wenn $Total\ Length\ (IP-Header) - Header\ Length\ (IP-Header) - Längenangabe\ (UDP-Header) > 0$ Bytes, dann handelt es sich um ein *UDP-Lite* Datagramm, da die Längenangabe im *UDP-Header* nicht über die *UDP-Nutzdaten* hinweg reicht. In diesem Fall wird die *UDP-Prüfsumme* nur über den *UDP-Header*, nicht aber über die *UDP-Nutzdaten* gebildet. Dies hat den Vorteil, dass beispielsweise trotz eines fehlerhaft übertragenen Bits im UDP-Datenbereich der Datenstrom dennoch weiterverarbeitet wird. Dies resultiert bei Audio- oder Videostreaming zwar in einem Qualitätsverlust bei der Übertragung, der Datenfluss bricht aber nicht ab. Der *UDP-Header* wird weiterhin mit der Prüfsumme kontrolliert, damit fehlgeleitete Pakete erkannt werden.

A.6.3. Flusssteuerung

Das eigentliche *User Datagram Protocol* besitzt keinerlei Vorkehrungen für eine Flusssteuerung. Somit haben Netzwerkgeräte wie beispielsweise *Router* die Aufgabe, UDP-Verbindungen zu bremsen, wenn diese das Netzwerk zu stark auslasten. Für Anwendungen, in welchen dennoch großer Wert auf eine UDP-Flusssteuerung gelegt wird, existieren Abwandlungen von *UDP*. Zwei dieser Abwandlungen sind das *Datagram Congestion Control Protocol (DCCP)* nach RFC4340 [73] für eine gesteuerte

Netzwerkauslastung und das *Reliable Datagram Protocol (RDP)* nach RFC908 [132] und RFC1151 [98] zur Einhaltung der korrekten Paketreihenfolge.

A.7. HyperText Transfer Protocol

Anwendungsschicht		HTTP	SNMP	
Transportschicht		TCP	UDP	
Internetschicht	ICMP			
Internetschicht	IP			ARP
Netzzugangsschicht	Ethernet			

Tabelle A.21.: Position von HTTP im TCP/IP-Referenzmodell

Beim *Hypertext Transfer Protocol (HTTP)* nach RFC1945 [11] und RFC2616 [36] handelt es sich um ein Protokoll der *Anwendungsschicht*, also der Schicht sieben im *OSI-Modell*. Es wird dazu verwendet, um Webseiten (HTML-Seiten, Grafiken,...) und andere Daten aus einem *Intranet* oder dem *World Wide Web (WWW)* zu einem *Webbrowser* zu übertragen.

Bei *HTTP* handelt es sich außerdem um ein Protokoll, in welchem die Informationen nicht mehr in einem festen *Header* als Bits dargestellt werden, wie zum Beispiel bei *IP*, *TCP* oder *UDP*. Stattdessen überträgt *HTTP* die Daten als lesbaren Text mit einem Zeilenumbruch <CR><LF> am Ende jeder Zeile. Weiterhin ist *HTTP* ein verbindungsloses Protokoll, bei welchem die Verbindung nach Ende der Übertragung nicht aufrechterhalten wird. Sollen anschließend weitere Daten übertragen werden, muss eine neue Verbindung aufgebaut werden. Zur Kommunikation ist *HTTP* auf ein darunter liegendes, zuverlässiges Transportprotokoll, wie zum Beispiel *TCP*, angewiesen.

A.7.1. Protokollversionen

Es existieren mehrere Versionen des *Hypertext Transfer Protokolls*. Angefangen bei *HTTP/0.9* aus dem Jahre 1991 und *HTTP/1.0* von 1996, über *HTTP/1.1* von 1999, welches heutzutage am weitesten verbreitet ist, bis hin zum *Hypertext Transfer Protocol Next Generation (HTTP-NG)*, welches sich noch in der Entwicklung befindet. Die wichtigsten Eigenschaften dieser Protokollversionen werden nachfolgend kurz erläutert:

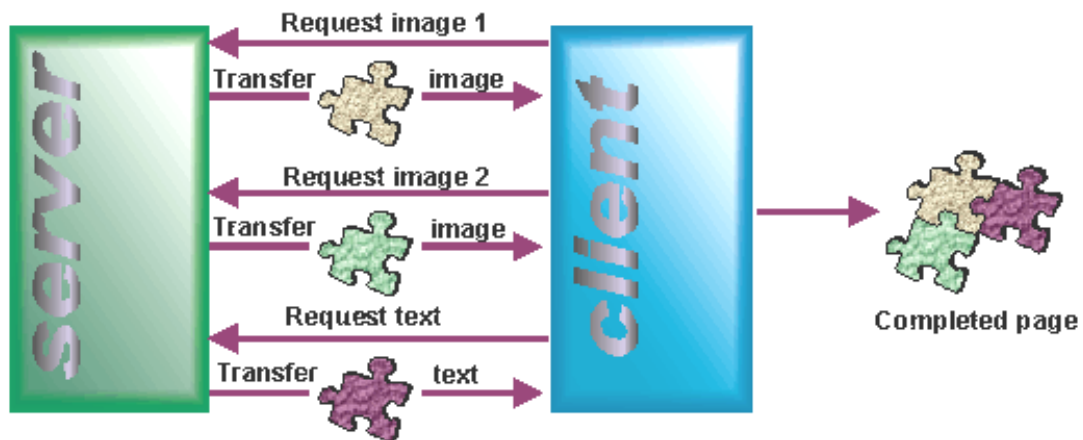


Abbildung A.15.: Anforderung einer Ressource mit HTTP/1.0 [41]

A.7.1.1. HTTP/0.9

Die Version 0.9 des *Hypertext Transfer Protokolls* wird heutzutage nur noch sehr selten verwendet. Es handelt sich um die einfachste Implementierung von *HTTP* und ist durch das *W3C* [133] beschrieben. Version 0.9 unterstützt lediglich einen *Simple-Request*, bestehend aus einem *GET-Request*. Der *Request* von Version 0.9 wurde so einfach wie möglich gewählt - ohne zusätzliche Parameter - um die zur Entstehungszeit von Version 0.9 noch langsamen Netzwerkverbindungen nicht unnötig zu belasten. Ein großer Nachteil von *HTTP/0.9* ist heutzutage, dass diese Protokollversion nur reinen Text, nicht aber zum Beispiel Grafiken, übertragen kann. Dieser *Simple-Request* von *HTTP/0.9* muss von *HTTP/1.0* auch unterstützt werden.

A.7.1.2. HTTP/1.0

Diese Version ist, genau wie die Vorgängerversion *HTTP/0.9*, in der RFC1945 [11] spezifiziert und heute noch weit verbreitet. Der große Nachteil dieser Version liegt jedoch darin, dass mit jeder Verbindung jeweils nur eine Ressource übertragen werden kann. Dies bedeutet, dass für eine Website mit zwei Bildern drei TCP-Verbindungen (eine Verbindung für die HTML-Seite und zwei Verbindungen für die Bilder) benötigt werden, wie Abbildung A.15 verdeutlicht. Dies ist sehr ineffektiv, da jeder Aufbau einer TCP-Verbindung einen Overhead erzeugt und bei jeder Verbindung mit dem *Slow Start Algorithmus* begonnen wird. Dies resultiert in einer großen Menge unnötigem Traffic und die maximale Datenübertragungsrate wird nur bei großen Dateien erreicht.

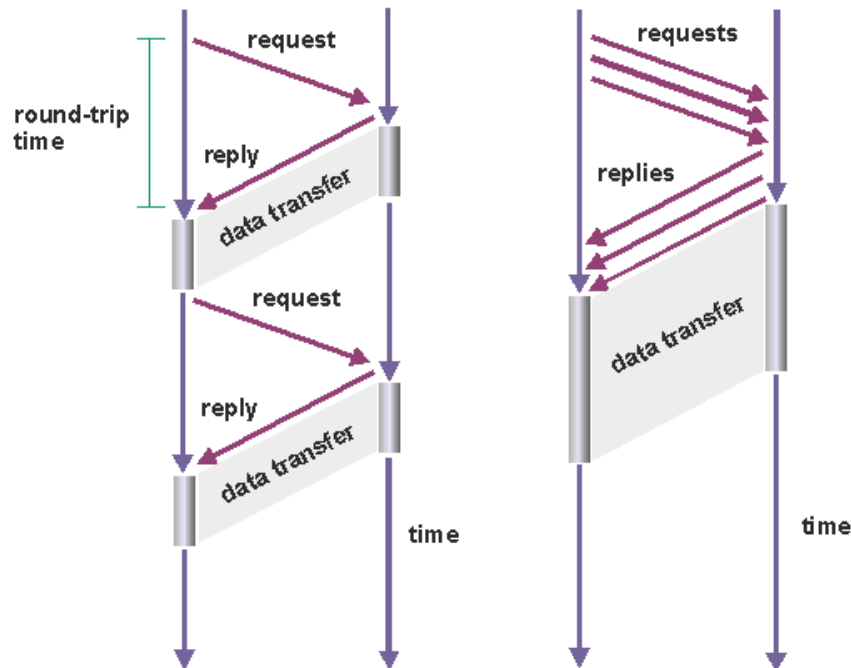


Abbildung A.16.: Pipelining bei HTTP/1.0 [41]

Ein anderer großer Nachteil von Version 1.0 liegt darin, dass nur ein *Webserver* pro *IP-Adresse* unterstützt wird. Es ist mit *HTTP/1.0* also nicht möglich, mehrere *Webserver* unter einer einzigen *IP-Adresse* zu betreiben. In Anbetracht der begrenzten Anzahl von *IP-Adressen* ist dies eine starke Einschränkung.

A.7.1.3. HTTP/1.1

Die oben genannten Einschränkungen von *HTTP/1.0* wurden mit Version 1.1 behoben. Das heißt, dass *HTTP/1.1* persistente Verbindungen unterstützt. Dadurch wird für das genannte Beispiel der Website mit zwei Bildern nur eine einzige TCP-Verbindung benötigt. Somit fällt der Overhead zum Verbindungsaufbau nur einmal an und der *Slow Start Algorithmus* wird nur einmal ausgeführt.

Eine weitere wichtige Erweiterung von *HTTP/1.1* ist die Unterstützung von *Pipelining*. Hierbei kann der Client mehrere *Requests* absenden, ohne vorher eine *Response* erhalten zu haben. Dadurch muss zwischen zwei Anfragen nicht immer die *Round Trip Time* abgewartet werden. Abbildung A.16 verdeutlicht den Vorteil des *Pipelining*s.

ANHANG A. NETZWERKGRUNDLAGEN

Ein weiteres wichtiges Feature der Version 1.1 ist die Unterstützung von «*Multi-Homed Servern*», also das Betreiben mehrerer *Webserver* auf einer *IP-Adresse*. Dies wird durch die *Host-Angabe* im *HTTP-Header* ermöglicht, welche später noch genauer behandelt wird.

Ab Version 1.1 des Protokolls ist es auch möglich, unterbrochene Datenübertragungen fortzusetzen. Hierzu kann der *HTTP-Header* eine *Range-Anweisung* enthalten.

A.7.1.4. HTTP-NG

Das *Hypertext Transfer Protocol Next Generation* (*HTTP-NG*) befindet sich derzeit noch in der Spezifikationsphase. Der aktuelle Entwicklungsstand ist beim *W3C* [134] einsehbar. Dieses Protokoll soll *HTTP/1.0* und *HTTP/1.1* ablösen, da diese unter anderem für sicherheitskritische Webanwendungen, wie elektronischen Handel, nicht geeignet sind. Dieses Problem soll bei *HTTP-NG* durch eine sichere Kommunikation, mittels Authentifizierung und Datenverschlüsselung, behoben werden.

Weiterhin verfügt *HTTP-NG* über Änderungen zur Verringerung der Netzwerklast. Hierzu werden zum Beispiel, wie in Abbildung A.17 gezeigt, über eine einzige TCP-Verbindung mehrere virtuelle Kanäle realisiert. Bei diesen kann dann beispielsweise ein Kanal zur Steuerung verwendet werden und ein anderer Kanal zur eigentlichen Datenübertragung.

A.7.1.5. HTTPS

Da das normale *HTTP* keine Datenverschlüsselung vorsieht, ist es leicht möglich, übertragene Daten abzuhehren. Um diese Schwäche zu umgehen, wurde das *Hypertext Transfer Protocol Secure* (*HTTPS*) entwickelt. Es ist unter der RFC2818 [105] spezifiziert und verwendet eine zusätzliche Schicht zur Verschlüsselung und Authentifizierung zwischen *TCP* und *HTTP*. Die Verschlüsselung basiert dabei auf *SSL/TLS* nach RFC2246 [28], RFC2712 [88], RFC2817 [72], RFC3268 [21], RFC4346 [29] und RFC4366 [12].

A.7.2. Funktionsweise

A.7.2.1. Adressierung einer Ressource

Jede Ressource im *WWW* oder einem *Intranet* besitzt eine eindeutige Adresse, den *Uniform Resource Identifier* (*URL*). Diese *URL* hat eine festgelegte Form nach Abbildung A.18 und darf eine

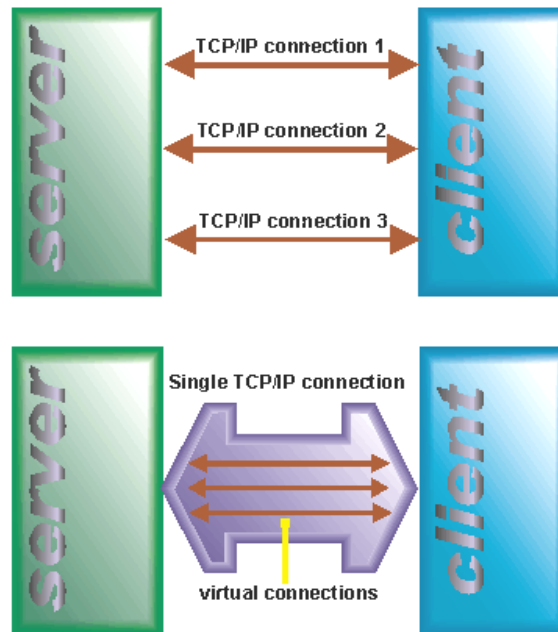


Abbildung A.17.: Virtuelle Kanäle bei HTTP-NG [41]

http://hans:geheim@www.beispiel.de:123/demo/exempel.cgi?land=de&stadt=b#abschnitt1

Protokoll Benutzer Passwort Domain Port Pfad Query Anker

Abbildung A.18.: Form einer URL [137]

Länge von 255 Zeichen nicht überschreiten.

Eine gültige *URL* muss dabei mindestens aus *Protokoll*, *Domain* und *Pfad* bestehen. Die einzelnen Teile einer *URL* haben die nachfolgenden Bedeutungen:

A.7.2.1.1. Protokoll Diese Angabe definiert das verwendete Netzwerkprotokoll. Mögliche Angaben sind beispielsweise «http», «https» oder «ftp».

A.7.2.1.2. Benutzer/Passwort Es besteht die Möglichkeit, mit einer *URL* zusätzlich Authentifizierungsinformationen zu übergeben. Der Benutzername und das Passwort sind hierbei durch einen Doppelpunkt getrennt und am Ende mit einem @-Zeichen mit dem Rest der *URL* verbunden.

ANHANG A. NETZWERKGRUNDLAGEN

A.7.2.1.3. Domain Die Domain bezeichnet den eigentlichen Server im Netzwerk und kann aus einem Bezeichner wie zum Beispiel «www.example.org» oder einer *IP-Adresse* bestehen.

A.7.2.1.4. Port Mit der Angabe eines *Ports* kann auf dem Server die gewünschte Applikation gewählt werden. Es gibt sogenannte *Standard-Ports*, wie zum Beispiel *Port* 80 für *HTTP* oder *Port* 443 für *HTTPS*.

A.7.2.1.5. Pfad Die Pfadangabe kennzeichnet die genaue Position der Ressource innerhalb des Serversystems. Dabei kann die Pfadangabe Verzeichnisse und Dateinamen beinhalten. Die Minimalangabe für den Pfad ist «/» und bezeichnet das *Home-Verzeichnis*.

A.7.2.1.6. Query Wie später gezeigt, können einer Anfrage an einen Server zusätzliche Parameter, bestehend aus Variablenname und Wert, angehängt werden. Dies kann bei einer *GET-Anfrage* direkt in der *URL* erfolgen. Hierzu werden die Parameter durch ein Fragezeichen getrennt an die eigentliche Adresse angehängt. Mehrere Parameter sind durch ein «&»-Zeichen getrennt.

A.7.2.1.7. Anker HTML-Dokumente können sogenannte *Anker* enthalten. Diese bilden eine Sprungmarke innerhalb des HTML-Dokuments. Durch die Anker-Angabe in der *URL*, welche durch ein «#»-Zeichen angehängt wird, können diese Sprungmarken in einem Dokument angesprungen werden.

A.7.2.2. Die HTTP-Übertragung

Möchte ein Client eine Ressource in einem Netzwerk anfordern, so geschieht dies über einen *Request*.

In den nachfolgenden Erläuterungen wird dabei nur auf *Requests* von *HTTP/1.0* und *HTTP/1.1* eingegangen. Die Auswahl wurde so getroffen, da *HTTP/0.9* nur noch selten und *HTTP-NG* noch nicht verwendet wird. *HTTPS* dagegen wird nicht näher behandelt, da es im Rahmen dieser Diplomarbeit nicht näher betrachtet wird.

Ein *Request* besteht aus einem *Request-Header*, gefolgt von den eigentlichen Nutzdaten. Der Server nimmt diesen *Request* entgegen und antwortet darauf mit einer *Response*, welche aus dem *Response-Header* und den angehängten Daten besteht.

ANHANG A. NETZWERKGRUNDLAGEN

Die *Request*- und *Response-Header* müssen hierbei eine festgelegte Norm einhalten. Die angehängten Nutzdaten dagegen unterliegen keiner Vorschrift.

A.7.3. Aufbau des HTTP-Headers

Der *HTTP-Header* enthält die Informationen zur angeforderten Ressource. Direkt danach kommt eine Leerzeile, gefolgt von den Nutzdaten. Hierbei sind die *Header* der Version 1.0 von denen der Version 1.1 zu unterscheiden.

Ein minimaler *HTTP/1.0-Standard-Header* besteht aus einer Zeile, gefolgt von einer Leerzeile. Der *Header* kann beispielsweise folgendermaßen aussehen:

```
GET [URL] HTTP/1.0
```

Ein Minimalheader der Version 1.1 hat hingegen den nachfolgenden Aufbau, wieder gefolgt von einer Leerzeile:

```
GET [URL] HTTP/1.1
Host: [Host]
```

Die Angaben [URL] adressieren hierbei die Ressource auf dem Zielsystem. Es sind dabei alle möglichen «Ausbauformen» der *URL* erlaubt. Es ist also möglich, entweder nur eine Pfadangabe anzugeben oder eine komplette *URL*, zum Beispiel bestehend aus Protokoll, Domain, Pfad und Query. Die zusätzliche Angabe *Host* muss bei *HTTP/1.1* vorhanden sein. Sie enthält die *Domain*, an welche die Anfrage gerichtet ist. Mit Hilfe dieser Angabe ist es bei *HTTP/1.1* möglich, mehrere *Webserver* auf einer einzigen *IP-Adresse* zu betreiben. Sollte diese Angabe bei einem *HTTP/1.1-Request* fehlen, antwortet der Server mit dem HTTP-Statuscode *400 Bad Request*.

Wie schon erwähnt, handelt es sich bei diesen beiden Beispielen um die minimalen Standardversionen der *HTTP-Header*. Alle möglichen *HTTP-Request-Methoden* sind der nachfolgenden Auflistung zu entnehmen:

- GET

Die *GET-Anfrage* kann eine Ressourcenanfrage mit zusätzlichen Parametern übermitteln. Diese Parameter werden an die *URL* angehängt und sind daher leicht einsehbar.

ANHANG A. NETZWERKGRUNDLAGEN

- POST

Die *POST-Anfrage* fordert ebenfalls eine Ressource an, wobei zusätzliche Parameter übermittelt werden können. Diese werden jedoch nicht als Teil der *URL* übertragen, sondern im *Body* der Nachricht übermittelt. Hiermit sind größere Datenmengen für die Parameter möglich und die Daten nicht so leicht einsehbar.

- HEAD

Mit diesem Anfragetyp liefert der Server nur den *Header*, nicht aber die eigentlichen Daten, zurück. Diese Form der Anfrage kann dazu verwendet werden, die Dateien im Browsercache auf aktuellere Versionen zu überprüfen. Die Antwort auf einen *HEAD-Request* ist dieselbe wie auf einen *GET-Request*, mit dem einzigen Unterschied, dass die Antwort auf den *HEAD-Request* keinen *Body* enthält.

Die bisher genannten *Request-Methoden* werden sowohl von *HTTP/1.0*, als auch von *HTTP/1.1* unterstützt. Die folgenden Methoden werden nur von *HTTP/1.1* unterstützt:

- PUT

Mit diesem *Request-Type* können Dateien auf das Zielsystem hochgeladen werden. Diese Form der Datenübertragung wird aber nur noch selten verwendet.

- DELETE

Durch diesen *Request* können Dateien auf dem Zielsystem gelöscht werden. Diese Funktion wird aber, genau wie *PUT*, nur noch selten verwendet.

- TRACE

Auf eine *TRACE-Anfrage* sendet der Server die Daten so zurück wie er sie erhalten hat. Damit kann überprüft werden wie die Daten beim Empfänger ankommen.

- OPTIONS

Mit Hilfe der *OPTIONS-Methode* können alle unterstützten Methoden des Empfängers abgefragt werden. Hierzu sendet der *Client* eine Anfrage nach folgendem Schema:

```
OPTIONS * HTTP/1.1  
Host: www.example.org
```

Daraufhin erhält er eine Antwort, welche die gewünschten Informationen enthält; im folgenden Beispiel also GET, HEAD, OPTIONS und TRACE:

ANHANG A. NETZWERKGRUNDLAGEN

HTTP/1.1 200 OK
Date: Wed, 12 Nov 1997 12:46:24 GMT
Server: Apache/1.3b3-dev
Content-Length: 0
Allow: GET, HEAD, OPTIONS, TRACE
Connection: close

- CONNECT

Die *CONNECT-Methode* wird verwendet, um über einen Proxy einen Tunnel zu schalten. Hierzu werden mit der *CONNECT-Methode* das Zielsystem und der Zielport angegeben.

A.7.3.1. Übermittlung von Argumenten

Wie schon gezeigt, können mit einem *Request* auch zusätzliche Parameter zum Server übertragen werden. Die dabei zu übertragenden Argumente können auf zwei Arten angehängt werden:

A.7.3.1.1. Argumente in einem GET-Request Die einfachste Art, um die Argumente zu übertragen, ist die *GET-Anfrage*. Bei ihr werden die Argumente durch ein «?»-Zeichen getrennt an die *URL* angehängt. Eine somit generierte Anfrage kann beispielsweise das folgende Format haben:

```
GET http://www.google.de/search?hl=de&q=Prozessor&btnG=Google-Suche&
meta= HTTP/1.1
Host: www.google.de
```

Die hierbei übertragenen Wertepaare sind:

```
hl=de
q=Prozessor
btnG=Google-Suche
meta=
```

Ein Vorteil der *GET* Anfrage ist, dass die *URL* zusammen mit den angehängten Argumenten zum Beispiel in den Favoriten des Browsers abgespeichert werden kann.

A.7.3.1.2. Argumente in einem POST-Request Bei der *POST-Anfrage* hingegen werden die Argumente nicht an die *URL* angehängt. Stattdessen stehen die Argumente im *Body* der Nachricht. Die folgende Nachricht stellt ein Beispiel für eine typische *POST-Anfrage* dar.

POST http://www.google.de/search HTTP/1.1

Host: www.google.de

hl=de&q=Mikrocontroller&btnG=Google-Suche&meta=

Die dabei übertragenen Wertepaare sind die gleichen wie im oberen Beispiel. Jedoch werden diese Argumente zum Beispiel nicht mehr mit abgespeichert, wenn die *URL* den Favoriten hinzugefügt wird.

A.7.3.2. Anweisungen im HTTP-Header

Im *HTTP-Header* können zusätzliche Angaben zu den Daten und der Verbindung übermittelt werden. Hierzu stehen mehrere Anweisungen zur Verfügung. Diese lassen sich in die nachfolgenden Gruppen unterteilen:

A.7.3.2.1. Metainformationen zum Body Bei den Metainformationen handelt es sich um Zusatzinformationen, welche die Daten im Body, also im Normalfall die POST- oder Dateidaten, näher beschreiben. Die möglichen Metainformationen werden nachfolgend aufgeführt:

- Allow

Durch die *Allow-Direktive* teilt der Server dem Client mit, welche *Request-Methoden* die angefragte Ressource unterstützt. Die erlaubten Parameter werden dabei durch Kommata getrennt angegeben.

Ein Beispiel einer solchen *Allow-Response* ist:

Allow: GET, HEAD, POST

- Content-Encoding

Mit dieser Anweisung gibt der Server an, welche Kodierungsverfahren bei den Daten verwendet werden. Die möglichen Angaben sind *gzip*, *x-gzip*, *compress*, *deflate* und *identity*. Die Angabe *identity* gibt dabei an, dass kein spezielles Kodierungsverfahren verwendet wird. Weiterhin können mehrere dieser Angaben durch Kommata getrennt kombiniert werden.

ANHANG A. NETZWERKGRUNDLAGEN

Ein Beispiel hierzu:

Content-Encoding: gzip, deflate

- Content-Language

Mit Hilfe der *Content-Language* kann die Sprache des HTTP-Nachrichteninhalts angegeben werden. Als Parameter wird die Abkürzung für die gewünschte Sprache oder eine durch Kommata getrennte Liste der Sprachkürzel nach RFC3282 [6] angegeben. Eine gültige Verwendung der *Content-Language* ist:

Content-Language: de, fr, en

- Content-Length

Mit der *Content-Length* wird die Länge des *Nachrichten-Body* in Bytes angegeben. Ein mögliches Beispiel hierzu:

Content-Length: 5834

- Content-Location

Die *Content-Location-Direktive* gibt eine alternative Adresse für die aktuelle Ressource an. Die Adressangabe kann dabei absolut oder relativ sein. Ein mögliches Beispiel einer solchen alternativen Adressangabe ist:

Content-Location: <http://www.example.org/info.html>

- Content-MD5

Mit der *Content-MD5-Anweisung* wird eine *MD5-Checksumme* nach RFC1321 [107] und RFC-1864 [92] angegeben. Die Prüfsumme erstreckt sich dabei über den kompletten *Nachrichten-Body*. Eine mögliche Angabe für *Content-MD5* ist:

Content-MD5: d41d8cd98f00b204e9800998ecf8427e

- Content-Range

Bei mehrteiligen Nachrichten kann mit *Content-Range* angegeben werden, welcher Teil im aktuellen *Nachrichten-Body* übertragen wird. Der Parameter setzt sich dabei folgendermaßen zusammen: Zuerst kommt das Wort «bytes», gefolgt von einem Leerzeichen. Anschließend kommt eine Angabe über die aktuell übermittelten Bytes in der Form «[Anfangsbyte]-[Endbyte]/[Gesamtlänge]». Das erste Byte besitzt den Index null.

ANHANG A. NETZWERKGRUNDLAGEN

Werden zum Beispiel die ersten 500 Bytes einer Nachricht in drei Teilnachrichten gesendet, so können die drei *Content-Range-Angaben* folgendermaßen aussehen:

Content-Range: bytes 0-189/500

Content-Range: bytes 190-356/500

Content-Range: bytes 357-499/500

- Content-Type

Die Angabe *Content-Type* gibt den Dateityp der Nachrichtendaten an. Der sogenannte *Mediatyp* besteht dabei nach RFC1700 [106] aus zwei Angaben, dem Haupt- und Untertyp. Die beiden Angaben sind durch einen Schrägstrich (*Slash*, «/») getrennt. Bei Bedarf kann durch ein Semikolon getrennt noch ein Parameter, wie zum Beispiel der verwendete Zeichensatz, angehängt werden. Ein Beispiel einer solchen *Content-Type-Angabe* ist:

Content-Type: text/html; charset=ISO-8859-4

- Expires

Diese Direktive gibt den Zeitpunkt im HTTP-Datumsformat an, ab welchem die übermittelte Ressource verfallen ist. Die Angabe ist dazu gedacht, *Proxy-Servern* mitzuteilen, ab wann die Nachricht nicht mehr aktuell ist. Die folgende Angabe stellt ein Beispiel einer *Expires-Direktive* dar:

Expires: Tue, 24 Apr 2007 19:36:28 GMT

- Last-Modified

Mit dieser Angabe wird dem Empfänger mitgeteilt, wann die angeforderte Ressource das letzte Mal geändert wurde. Die Angabe erfolgt im HTTP-Datumsformat. Wurde die angeforderte Ressource beispielsweise das letzte Mal am 24. April 2007 um 19:36 Uhr geändert, sieht die Angabe wie folgt aus:

Last-Modified: Tue, 24 Apr 2007 19:36:28 GMT

A.7.3.2.2. Allgemeine Anweisungen Die neun allgemeinen Anweisungen können sowohl bei *Requests*, als auch bei *Responses* verwendet werden.

- Cache-Control

Die Anweisung *Cache-Control* steuert die Cache-Verwaltung, also das Zwischenspeichern von Ressourcen. Für diese Anweisung gibt es mehrere Request- und Responsewerte. Diese können durch Kommata getrennt kombiniert werden:

Die Requestwerte lauten:

- no-cache

Der Client fordert mit diesem Wert eine aktuelle, nicht zwischengespeicherte Ressource an.

- no-store

Der Cache wird hiermit angewiesen, alle Verbindungsdaten nach dem Versenden der Antwort zu löschen.

- max-age=[Sekunden]

Der Client fordert mit *max-age* eine Ressource an, welche bisher maximal die angegebene Anzahl von Sekunden im Cache liegen darf. Sollte die angeforderte Ressource schon länger im Cache liegen, muss sie vom Server erneut angefordert werden.

- max-stale=[Sekunden]

Mit diesem Parameter kann dem Cache erlaubt werden, eine bereits «veraltete» Ressource auszuliefern. Die Angabe der Sekunden ist dabei optional. Wird die Anzahl der Sekunden angegeben, so darf die Ressource jedoch nicht länger als die angegebene Zeit verfallen sein.

- min-fresh=[Sekunden]

Mit *min-fresh* kann der Client Daten vom Cache anfordern, welche mindestens noch die angegebene Anzahl von Sekunden «aktuell» sein müssen. Das heißt, dass die Lebensdauer des Cacheeintrags größer sein muss als die bisherige Lebensdauer plus die angegebene Anzahl von Sekunden.

ANHANG A. NETZWERKGRUNDLAGEN

- only-if-cached

Mit diesem Parameter darf der Cache nur selbst zwischengespeicherte Daten ausliefern.

Für die Responsewerte sind die folgenden Angaben möglich:

- public

Der Server teilt den Empfängern mit diesem Wert mit, dass jeder Empfänger die empfangenen Daten zwischenspeichern darf.

- private

Mit dieser Angabe darf die Ressource nur von privaten, also «nicht-öffentlichen», Systemen gespeichert werden.

- no-cache

Der Server teilt dem Cache hiermit mit, dass die empfangene Ressource zwar zwischengespeichert werden darf, aber vor jeder Auslieferung überprüft werden muss.

- no-store

Durch *no-store* wird der Cache angewiesen, nach jeder Verbindung alle Verbindungsdaten zu löschen.

- no-transform

Der Parameter *no-transform* zeigt an, dass der Nachrichtenbody vom Empfänger nicht verändert werden soll.

- must-revalidate

Hiermit wird der Cache angewiesen, den Status der gespeicherten Ressource vor der Übertragung zu prüfen. Sollte die Ressource abgelaufen sein, muss diese zuerst wieder validiert werden. Dieser Parameter gilt nur für den Browser-Cache.

- proxy-revalidate

Dieser Parameter bewirkt das Gleiche wie *must-revalidate*, gilt jedoch für Proxies.

ANHANG A. NETZWERKGRUNDLAGEN

- max-age=[Sekunden]

Mit *max-age* kann der Server dem Cache das maximal erlaubte Alter der Ressource mitteilen. Sobald die Ressource im Cache älter als das erlaubte Alter in Sekunden ist, so muss die Ressource neu vom Server angefordert werden.

- s-maxage=[Sekunden]

Der *s-maxage-Parameter* gilt nur für öffentliche Caches. Er gibt das maximal erlaubte Alter einer Ressource in Sekunden an. Wenn zusätzlich zum *s-maxage-Parameter* der Parameter *max-age* oder die *Expires-Anweisung* angegeben sind, so werden diese durch *s-maxage* überschrieben.

Eine gültige *Cache-Control* Angabe kann beispielsweise folgendermaßen aussehen:

Cache-Control: no-cache

- Connection

Mit dieser Anweisung wird angegeben, ob nach der Datenübertragung die Verbindung aufrecht erhalten oder beendet werden soll. Bei *HTTP/1.0* wird eine Verbindung standardmäßig nach jeder Übertragung abgebaut. Es kann hier jedoch mit dem Parameter *keep-alive* ein Aufrechterhalten der Verbindung angefordert werden.

HTTP/1.1 hingegen verwendet standardmäßig persistente Verbindungen. In diesem Fall kann die Verbindung nach der Datenübertragung jedoch mit dem Parameter *close* geschlossen werden.

Eine mögliche Anweisung wäre zum Beispiel:

Connection: close

- Date

Mit der *Date-Anweisung* wird das aktuelle Datum im *Header* mit übertragen. Das dabei vorwiegend verwendete Datum hat die Form wie in RFC1123 [14] beschrieben. Eine solche, nach RFC1123 gültige Datumsangabe, kann wie folgt aussehen:

Date: Tue, 06 Apr 2004 21:45:13 GMT

ANHANG A. NETZWERKGRUNDLAGEN

- Pragma

Die *Pragma-Direktive* gibt an wie die Nachricht von Caches verarbeitet werden soll. Mit dem Parameter *no-cache* kann angegeben werden, dass die Nachricht nicht zwischengespeichert werden soll. Weitere Parameter existieren für diese Anweisung nicht. Die gültige Angabe zum Verbot von Cache-Funktionen sieht folgendermaßen aus:

Pragma: no-cache

- Trailer

Die *Trailer-Anweisung* zeigt, dass noch weitere *Header-Angaben* in der Nachricht folgen und gibt diese an. Somit kann eine Nachricht aus mehreren Teilen bestehen und auch nach den eigentlichen Daten noch *Header-Anweisungen* beinhalten. Die Anweisungsnamen *Transfer-Encoding*, *Content-Length* und *Trailer* sind als Parameter für die *Trailer-Anweisung* jedoch nicht erlaubt.

Das folgende ist ein mögliches Beispiel für eine HTTP-Nachricht mit *Trailer-Angabe*:

HTTP/1.1 200 OK

Server: Apache/1.3.27

Transfer-Encoding: chunked

Trailer: Cache-Control

f4b

[Die ersten 3915 Zeichen der Datei]

9e3

[Weitere 2531 Zeichen der Datei]

4e8

[Die letzten 1256 Zeichen der Datei]

0

Cache-Control: no-cache

In diesem Beispiel wird im *Header* angegeben, dass noch eine *Cache-Control* Angabe folgt. Nach dem *Header* folgen anschließend die eigentlichen Daten und am Ende folgt die bereits angekündigte *Cache-Control-Angabe*.

ANHANG A. NETZWERKGRUNDLAGEN

- Transfer-Encoding

Mit der *Transfer-Encoding-Direktive* wird angegeben, wie die Nachricht kodiert ist. Dabei sind die Verfahren *gzip*, *x-gzip*, *compress*, *x-compress* und *deflate* erlaubt. Wenn die Daten nicht speziell kodiert sind, wird *identity* angegeben.

Für den Fall, dass mehrere Kodierungsverfahren zum Einsatz kommen, müssen diese durch Kommata getrennt und in der richtigen Reihenfolge als Parameter angegeben werden.

Als weiteren Parameter gibt es außerdem noch *chunked*. Hierbei handelt es sich allerdings nicht um eine Kodierung der Daten, sondern die Angabe, dass die Nachricht in mehreren Teilen übertragen wird.

Wenn mehrere Kodierungsverfahren nach *Transfer-Encoding* angegeben werden, so muss *chunked* als letztes genannt werden. Weiterhin ist darauf zu achten, dass bei einem Kodierungsverfahren ungleich *identity*, die Angabe *Content-Length* im *Header* nicht vorkommen darf beziehungsweise andernfalls ignoriert wird.

Ein Beispiel zum *chunked-Verfahren* ist im Beispielheader der *Trailer-Direktive* zu finden. Hier wird das *chunked-Verfahren* im *Header* mit *Transfer-Encoding: chunked* definiert. Anschließend wird der *Body* der Nachricht aus einzelnen Teilen (*Chunks*) zusammengesetzt. Jeder dieser Teile besteht aus einer Längenangabe im Hexadezimalformat, welche die Anzahl Bytes des Nachrichtenteils angibt. Danach folgt ein Zeilenumbruch, bestehend aus `<CR><LF>`. Anschließend folgen die eigentlichen Daten, welche wieder mit einem Zeilenumbruch abgeschlossen werden. Um das Ende der Daten anzuzeigen, wird als Längenangabe eine null übermittelt, gefolgt von einem Zeilenumbruch.

Das *chunked-Verfahren* eignet sich für Ressourcen, welche nicht sofort komplett zur Verfügung gestellt werden können. In diesem Fall kann immer wieder ein Teil der Daten übertragen werden.

- Upgrade

Mit der *Upgrade-Anweisung* kann der Client versuchen, die momentane Datenübertragung auf ein anderes Protokoll umzuschalten. Somit ist es möglich, eine bestehende *HTTP/1.0-Kommunikation* auf *HTTP/1.1* umzustellen. Der Client sendet hierzu die *Upgrade-Anweisung*, gefolgt von einer mit Kommata getrennten Liste der Protokolle, welche er bevorzugt verwenden würde. Wenn der Server die Protokolländerung akzeptiert, sendet er den *Response-Code* 101 zurück, gefolgt von einer eigenen *Upgrade-Anforderung* mit dem gewünschten Protokoll. Anschließend wird das Übertragungsprotokoll direkt nach dieser Angabe umgestellt.

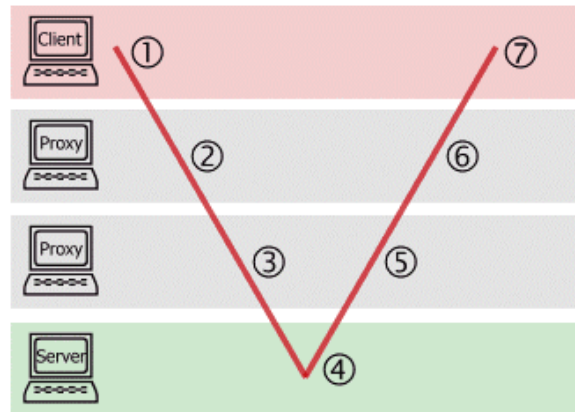


Abbildung A.19.: Beispiel für einen TRACE-Request [140]

Ein Beispiel für eine solche *Upgrade-Anfrage* ist:

Upgrade: HTTP/1.0, SHTTP/1.1

- Via

Die *Via-Direktive* dient der Verfolgung des Nachrichtenweges einer HTTP-Nachricht. Wenn von einem Client eine *TRACE-Anfrage* gesendet wird, so sind alle *Proxies*, welche von der HTTP-Nachricht passiert werden, angewiesen, die *Via-Direktive* zu ergänzen. Jeder Proxy fügt dabei Informationen, bestehend aus Protokollinformation, einem Leerzeichen, Proxyname und eventueller Portangabe, an das Ende der *Via-Direktive* an. Die Protokollinformation hat dabei entweder das Format «Protokollname/Protokollversion» oder nur «Protokollversion». Ein Proxyserver kann weiterhin bei Bedarf einen durch Leerzeichen getrennten Kommentar an das Ende seiner Angaben anhängen.

Sollte ein Proxy eine bereits gesetzte *Via-Direktive* empfangen, so kann er seine Informationen durch ein Komma getrennt an die bereits vorhandenen Informationen anhängen.

Ein Beispiel für einen *TRACE-Request* ist in Abbildung A.19 dargestellt.

Die *HTTP-Header* an den einzelnen Stellen der Übertragung haben dabei nachfolgende Formen:

An Stelle 1:

TRACE / HTTP/1.1

Host: server.de

Max-Forwards: 5

ANHANG A. NETZWERKGRUNDLAGEN

An Stelle 2:

TRACE / HTTP/1.1

Host: server.de

Max-Forwards: 4

Via: 1.1 Proxy1

An Stelle 3:

TRACE / HTTP/1.1

Host: server.de

Max-Forwards: 3

Via: 1.1 Proxy1, 1.1 Proxy2

An Stelle 4 bis 7:

HTTP/1.1 200 OK

Content-Type: message/http

Content-Length: 81

TRACE / HTTP/1.1

Host: server.de

Max-Forwards: 3

Via: 1.1 Proxy1, 1.1 Proxy2

- Warning

Die *Warning-Direktive* enthält zusätzliche Statusinformationen. Die Parameter haben dabei die Form «Warnungscode Servername[:Port] “Warnungstext“ [“Datum“]».

Hierbei sind folgende Warnungscodes definiert:

– 110

Die *Response-Daten* sind veraltet.

– 111

Die *Response-Daten* können veraltet sein. Der Cache konnte die Aktualität der Daten nicht überprüfen.

– 112

Der Cache hat keine Verbindung zum Netzwerk.

ANHANG A. NETZWERKGRUNDLAGEN

– 113

Die zwischengespeicherten Daten sind älter als 24 Stunden.

– 199

Es handelt sich um zusätzliche Informationen. Das System, welches diese Meldung empfängt, muss auf diese *Warning* nicht reagieren. Es kann den Warnungstext jedoch in einem Logfile ablegen oder dem Benutzer eine Fehlermeldung anzeigen.

– 214

Der *Cache* oder *Proxy* mussten die Kodierung oder den Medientyp der Daten ändern.

– 299

Es handelt sich um zusätzliche, persistente Informationen. Das System, welches diese Meldung empfängt, muss auf diese *Warning* nicht reagieren. Es kann den Warnungstext jedoch in einem Logfile ablegen oder dem Benutzer eine Fehlermeldung anzeigen.

Den Warnungscodes folgt der Servername und eventuell eine Portangabe. Anschließend folgt, getrennt durch ein Leerzeichen und in einfache Anführungszeichen gesetzt, der Warnungstext. Diesem kann bei Bedarf noch eine Datumsangabe in doppelten Anführungszeichen folgen.

Ein Beispiel einer Warnung ist:

Warning: 110 proxy.com:88 "Response is stale" "Tue, 06 Apr 2004 21:45:13 GMT"

A.7.3.2.3. Request-Anweisungen Die folgenden *Request-Anweisungen* werden bei Anfragen vom Client an den Server verwendet.

- Accept

Mit dieser Anweisung kann der Client bei der Anforderung einer Ressource angeben, welche *Mediatypen* er als Antwort auf diese Anfrage erwartet. Der *Mediatyp* setzt sich aus dem Haupt- und Untertyp, durch einen Schrägstrich (*Slash*, «/») getrennt, zusammen. Anstatt einem Untertyp kann auch ein Stern (*) angegeben werden. Dies bedeutet dann, dass der Client alle möglichen Untertypen akzeptiert. Bei Bedarf kann durch ein Semikolon getrennt ein Qualitätsfaktor angehängt werden. Dieser Faktor gibt an, wie sehr der angegebene *Mediatyp* gewünscht wird und kann im Bereich von null bis eins liegen. Die Zwischenwerte werden mit einem Punkt als Dezimaltrennzeichen geschrieben. Sollte kein Qualitätsfaktor angegeben

ANHANG A. NETZWERKGRUNDLAGEN

werden, wird die Angabe mit dem Faktor $q=1$ interpretiert. Mehrere *Mediatypen* können durch Kommata getrennt angegeben werden.

Mit dem nachfolgenden Beispiel teilt der Client dem Server mit, dass er Daten vom Typ «text/html» erwartet. Sollte dieses Datenformat nicht möglich sein, darf aber auch jeder beliebige Untertyp des Haupttyps «text» geliefert werden.

Accept: text/html, text/*; q=0.9

- Accept-Charset

Mit *Accept-Charset* gibt der Client an, welche Zeichensätze er laut *IANA* [50] für das aktuell angeforderte Dokument erwartet. Sollen mehrere mögliche Zeichensätze angegeben werden, so werden diese durch Kommata getrennt. Die Angabe eines zusätzlichen Qualitätsfaktors ist genau wie bei der *Accept-Direktive* möglich. Eine mögliche Verwendung von *Accept-Charset* ist:

Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7

- Accept-Encoding

Mit dieser Angabe gibt der Client an, welche Kodierungsverfahren er unterstützt. Die möglichen Angaben sind *gzip*, *x-gzip*, *compress*, *x-compress* und *deflate*. Soll keine spezielle Kodierung möglich sein, so kann dies mit dem Parameter *identity* signalisiert werden. Mehrere Kodierungsverfahren werden durch Kommata getrennt und bei Bedarf kann zusätzlich ein Qualitätsfaktor, wie bei der *Accept-Direktive*, angegeben werden.

Ein Beispiel zu *Accept-Encoding*:

Accept-Encoding: gzip

- Accept-Language

Mit *Accept-Language* kann der Client die von ihm bevorzugten Sprachen angeben. Die Angabe erfolgt mit einem Sprachkürzel nach RFC3282 [6]. Bei mehreren Sprachangaben werden die Sprachkürzel durch Kommata getrennt. Die Angabe eines Qualitätsfaktors ist wie bei den vorherigen Direktiven möglich. Eine mögliche *Accept-Language-Angabe* kann die folgende Form haben:

Accept-Language: de, en, fr

ANHANG A. NETZWERKGRUNDLAGEN

- Authorization

Die *Authorization-Direktive* wird dazu verwendet, den Client gegenüber dem Server zu identifizieren, um somit an geschützte Daten zu gelangen. Die Authorisierungsverfahren von *HTTP/1.0* und *HTTP/1.1* unterscheiden sich hier jedoch.

HTTP/1.0 unterstützt nur das *Basic-Schema* zur Authentifizierung. Hierbei wird als Parameter das Wort «Basic», gefolgt von einem Leerzeichen, angegeben. Anschließend folgen die *Base64* kodierten Angaben zu Benutzername und Passwort. Der mit *Base64* kodierte String hat die Form «[Benutzername]:[Passwort]».

Da die *Base64-Kodierung* der Daten jedoch sehr unsicher ist, wurde mit *HTTP/1.1* ein zweites Verfahren namens *Digest* eingeführt. Dieses Kodierungsverfahren schützt den Benutzernamen und das Passwort bei der Übertragung besser als das einfachere *Basic-Verfahren*. Beide Kodierungsverfahren sind in RFC2617 [40] beschrieben. *Digest* verwendet zur Verschlüsselung der Benutzerdaten mehrere Parameter, wie einen Zufallswert, Benutzername, Passwort, die *HTTP-Methode* und die angeforderte *URL*.

Bei einem Verbindungsaufbau zu einer geschützten Ressource sendet der Client zunächst einen ganz normalen *Request*. Stellt der Server daraufhin fest, dass die angeforderte Ressource geschützt ist, so weist er die Anfrage mit dem Statuscode «401 Unauthorized» ab. Dies bedeutet für den Client, dass er entweder nicht weiter versucht, diese Ressource zu erhalten, oder dass er stattdessen nun mit der *Authorization-Direktive* versucht, Zugang zu der Ressource zu bekommen.

Mögliche *Authorization-Requests* können zum Beispiel folgendermaßen aussehen:

- Basic

Authorization: Basic SBRNTHdvrmxkOgludjVybnV2

- Digest

Authorization: Digest username="Benutzername",
realm="testrealm@host.com",
nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
uri="/dir/index.html",
qop=auth,
cnonce="0a4f113b",
response="6629fae49393a05397450978507c4ef1",
opaque="5ccc069c403ebaf9f0171c9517f40e41"

ANHANG A. NETZWERKGRUNDLAGEN

- Cookie

Bei *Cookies* ist zu erwähnen, dass diese kein offizieller Bestandteil des *HTTP-Headers* sind. Da *Cookies* aber sehr weit verbreitet sind, werden sie hier dennoch erwähnt.

Bei einem *Cookie* handelt es sich um eine kleine Textdatei, welche durch den *Server* auf dem *Client* gespeichert wird. Der *Server* weist den *Client* mit dem Befehl *Set-Cookie* an, eine Zuweisung der Form «Cookiename=Cookiewert» abzuspeichern. Weiterhin wird zu jedem Cookie die zugehörige *URL* abgelegt.

Sobald der *Client* zu einem späteren Zeitpunkt die gleiche *URL* anfordert, sendet er mit der *Cookie-Anweisung* im *Header* das zuvor abgespeicherte Wertepaar mit zum *Server*. Der *Server* erhält dadurch clientspezifische Daten, die er dort zuvor deponiert hat, und kann mit diesen Werten weiterarbeiten. Mehrere *Cookies* werden hierbei durch Kommata getrennt übertragen. Weiterhin ist es möglich, den *Cookie* je nach Pfadangabe für mehrere *URLs* zu verwenden.

Eine mögliche Anweisung zum Übertragen eines *Cookies* an den Server ist zum Beispiel:

Cookie: Autofarbe=blau

- Expect

Mit der *Expect-Anweisung* kann der *Client* gewisse Erwartungen seinerseits an den *Server* senden. Sollte der *Server* diese Erwartungen nicht erfüllen können, antwortet er mit dem Fehlercode «417 Expectation Failed».

- From

Mit der *From-Angabe* wird die E-Mail-Adresse des Benutzers angegeben, welcher den *Client* betreut. Eine hierbei mögliche Angabe kann das nachfolgende Format haben:

From: name@domain.de

- Host

Die *Host-Angabe* gibt den Hostnamen des *Servers* an. Dieser Wert ist bei *HTTP/1.0* optional, bei *HTTP/1.1* hingegen handelt es sich um eine Pflichtangabe. Da *HTTP/1.1* «*Multi-Homed Server*» unterstützt, muss mit dieser Angabe der gewünschte *Server* spezifiziert werden. Für den Fall, dass der gewünschte *Server* auf einem Port ungleich 80 läuft, muss dies nach dem Hostnamen durch einen Doppelpunkt getrennt angegeben werden.

Als Beispiel hierzu:

ANHANG A. NETZWERKGRUNDLAGEN

Host: www.example.org:1234

- If-Match

Die *If-Match-Anweisung* weist den *Server* an, die Ressource nur zu übermitteln, wenn der angegebene *Entity Tag (ETag)* mit dem *ETag* der Ressource auf dem *Server* übereinstimmt. Bei den *ETags* handelt es sich um Prüfsummen, welche der *Server* für jede Ressource erstellt und dem *Client* mitliefert. Die Prüfsummen werden dazu verwendet, die Identität von Dokumenten zu überprüfen. Der Berechnungsalgorithmus dieser Prüfsummen ist allerdings nicht fest vorgeschrieben und kann je nach Belieben auf dem Server implementiert werden.

Sollen bei der *If-Match-Direktive* mehrere *ETags* angegeben werden, so werden diese durch Kommata getrennt. Alternativ kann auch ein Stern «*» angegeben werden; in diesem Fall sind dann alle Dateien erlaubt.

Mit Hilfe der *If-Match-Anweisung* kann ein *Client* eine Anfrage mit einem bestimmten *ETag* an den *Server* senden. Der *Server* überprüft daraufhin den *ETag* und sendet bei Übereinstimmung die komplette Ressource erneut an den *Client*.

Ein Beispiel zur *If-Match* Anweisung:

If-Match: "2c986a-b20-fc93242e"

- If-Modified-Since

Mit *If-Modified-Since* kann der *Server* angewiesen werden, die Ressource nur zu senden, wenn diese seit dem angegebenen Datum geändert wurde. Die Datumsangabe geschieht dabei im *HTTP-Datumsformat*. Ein Beispiel hierzu:

If-Modified-Since: Tue, 10 Apr 2007 18:25:11 GMT

- If-None-Match

Die Direktive *If-None-Match* entspricht dem Gegenteil der *If-Match-Anweisung*. Der *Server* sendet die Ressource nur, wenn diese dem angegebenen *ETag* nicht entspricht. Sollte die Ressource dagegen dem angegebenen *Entity-Tag* entsprechen, sendet der *Server* nur einen *Header* mit dem Statuscode «304 Not Modified» und ohne nachfolgenden *Nachrichten-Body*. Eine solche *If-Not-Match-Anfrage* kann folgendermaßen aussehen:

If-None-Match: "2c986a-b20-fc93242e"

ANHANG A. NETZWERKGRUNDLAGEN

- If-Range

Beim Anfordern von fehlenden Teilen einer mehrteiligen Nachricht kann mit der *If-Range-Anweisung* sichergestellt werden, dass sich die Datei seit dem Empfang der ersten Nachrichtenteile nicht verändert hat. Sollte sich die Datei geändert haben, werden nicht die noch fehlenden Nachrichtenteile ausgeliefert, sondern die komplette Ressource wird neu übertragen.

Zur Überprüfung, ob sich an den Daten etwas geändert hat, wird entweder eine *HTTP-Datumsangabe* oder ein *Entity-Tag* verwendet. Der *Server* entscheidet dadurch, welche Aufgabe er auszuführen hat.

Eine mögliche *If-Range-Anweisung* mit einer *HTTP-Datumsangabe* ist in folgendem Beispiel zu finden:

If-Range: Tue, 10 Apr 2007 18:25:11 GMT

- If-Unmodified-Since

Mit dieser Direktive kann der *Client* vom *Server* eine Ressource anfordern, wenn diese seit einem gewissen Datum nicht verändert wurde. Dieses Datum wird im *HTTP-Datumsformat* angegeben.

If-Unmodified-Since: Tue, 10 Apr 2007 18:25:11 GMT

- Max-Forwards

Mit dieser Angabe kann festgelegt werden, wie oft eine *HTTP-Nachricht* von *Proxy-Servern* weitergeleitet werden darf. *Max-Forwards* wird nur bei *Requests* vom Typ *TRACE* und *OPTIONS* verwendet. Jeder *Proxy*, welcher von dieser *HTTP-Nachricht* passiert wird, dekrementiert die *Max-Forwards-Angabe* um eins und leitet das Paket weiter. Erhält ein *Proxy* eine Nachricht mit *Max-Forwards: 0*, so leitet er diese Nachricht nicht weiter, sondern sendet sie zurück an den *Client*.

Ein Beispiel zu einer vom *Client* generierten *Max-Forwards* Angabe:

Max-Forwards: 6

- Proxy-Authorization

Diese Direktive wird dazu verwendet, einen *Client* beziehungsweise dessen Benutzer, an einem *Proxy-Server* zu authentifizieren. Wie bei der *Authorization-Direktive* werden hierzu die Verfahren *Basic* und *Digest* verwendet.

ANHANG A. NETZWERKGRUNDLAGEN

Im Normalfall nimmt der erste *Proxy* die Zugangsdaten entgegen und wertet diese aus. Für den Fall, dass mehrere *Proxy-Server* hintereinander geschaltet sind, kann der erste *Proxy* die Zugangsdaten zusätzlich an einen weiteren *Proxy* weiterreichen.

Eine *Proxy-Authorization* kann zum Beispiel wie folgt aussehen:

Proxy-Authorization: Basic SBRNTHdvrnmxkOgludjVybnV2

- Range

Mit Hilfe der *Range-Anweisung* kann ein *Client* gezielt nur bestimmte Bytes einer Datei vom *Server* anfordern. Dies kann beispielsweise dazu verwendet werden, einen abgebrochenen Download fortzusetzen.

Als Parameter wird der *Range-Anweisung* zuerst der String «bytes=» übergeben. Dahinter folgt das Startbyte mit einem anschließenden Bindestrich. Der Index der Bytes beginnt bei 0. Nach dem Bindestrich kann optional das letzte gewünschte Byte angegeben werden. Wird auf diese zusätzliche Angabe verzichtet, werden alle Bytes vom Startbyte bis zum Dateiende übertragen. Mehrere Bytebereiche können mit Kommata getrennt angegeben werden.

Das nachfolgende Beispiel fordert den Bereich von Byte 19 bis 53, den Bereich von Byte 98 bis 103 und alle Bytes ab Byte 154 an:

Range: bytes=19-53,98-103,154-

- Referer

Die Angabe *Referer* gibt die *URL* an, von welcher auf die aktuelle Seite verwiesen wurde. Diese Angabe kann vom *Server* beispielsweise dazu verwendet werden, festzustellen, von welchen Webseiten die Besucher auf die aktuelle Seite kamen. Der Betreiber einer Seite erhält dadurch Auskunft über diejenigen Webseiten, welche einen *Link* auf seine Seite enthalten.

Die Angabe der *URL* kann bei der *Referer-Direktive* absolut oder relativ sein. Eine mögliche Angabe ist:

Referer: http://www.example.org

- TE

Mit der *TE-Angabe* gibt der *Client* an, welche Kodierungsverfahren er für die aktuelle *HTTP-Nachricht* zulässt, und ob er *Trailer* bei mehrteiligen Nachrichten erlaubt. Die möglichen Angaben für diese Anweisung sind: *gzip*, *x-gzip*, *compress*, *x-compress*, *deflate* sowie *identity*,

ANHANG A. NETZWERKGRUNDLAGEN

wenn keine besondere Kodierung verwendet wird. Um anzuzeigen, dass *Trailer* bei mehrteiligen Nachrichten erlaubt sind, kann *trailers* angegeben werden.

Mehrere Angaben als Parameter werden durch Kommata getrennt und es kann, wie zum Beispiel bei der *Accept-Anweisung*, ein Qualitätsfaktor angegeben werden. Als mögliches Beispiel für die Anwendung der *TE-Direktive* ergibt sich somit:

TE: trailers, gzip;q=0.9

- User-Agent

Über diese Anweisung hat der *Client* die Möglichkeit, dem *Server* Informationen über sich selbst zukommen zu lassen. Es können also beispielsweise der Name und die Version des *Webrowsers* oder auch Informationen über das Betriebssystem angegeben werden. Der *Server* kann diese Informationen für statistische Zwecke verwenden. Es besteht auch die Möglichkeit, mit Hilfe dieser Informationen die zu sendenden Daten an den Webbrowser anzupassen. Dies wird häufig bei *HTML* verwendet, da sich nicht alle *Browser* 100%ig an den *W3C-Standard* halten.

Mögliche Beispiele von *User-Agent-Angaben* sind:

- User-Agent: Mozilla/4.1
- User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 2.0.50727; .NET CLR 1.1.4322)
- User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; de; rv:1.8.1.3) Gecko/20070309 Firefox/2.0.0.3
- User-Agent: Mozilla/5.0 Galeon/1.2.7 (X11; Linux i686; U;) Gecko/20030131
- User-Agent: Opera/7.51 (X11; SunOS sun4u; U) [de]

A.7.3.2.4. Response-Anweisungen Die folgenden *Response-Anweisungen* können vom *Server* bei einer Antwort mit zum *Client* gesendet werden:

- Accept-Ranges

Mit dieser Anweisung kann der *Server* dem *Client* mitteilen, dass clientseitige *Range-Anfragen* erlaubt oder verboten sind. Die dabei möglichen Angaben sind «bytes» für erlaubte *Range-Anfragen* beziehungsweise «none», wenn keine solchen Anfragen unterstützt werden.

ANHANG A. NETZWERKGRUNDLAGEN

Ein Beispiel hierzu:

Accept-Ranges: bytes

- Age

Mit *Age* kann der *Server* das Alter der angeforderten Ressource in Sekunden angeben. Existiert die Ressource zum Beispiel seit 1384 Sekunden, so signalisiert der Server dies durch folgende Angabe:

Age: 1384

- ETag

Mit *ETag* gibt der *Server* den eindeutigen *Entity-Tag* der angeforderten Ressource an. Mit dieser Angabe kann der Client danach *If-Match*-, *If-None-Match*- und *If-Range-Anfragen* stellen. Enthält eine solche *ETag-Angabe* ein vorangestelltes «W/», handelt es sich um einen sogenannten *Weak Entity Tag*. Dies bedeutet, dass der *Server* eine zweite gleichwertige Datei besitzt, welche er senden kann.

Zwei Beispiele für eine Anfrage mit *Entity Tag* und *Weak Entity Tag*:

ETag: «htm769x»

ETag: W/«htm769x»

- Location

Mit der *Location-Direktive* teilt der *Server* dem *Client* die neue *URL* der angeforderten Ressource mit. Den genauen Grund der Adressänderung der Ressource kann dem mitgesendeten *Statuscode* entnommen werden. Ein Ressource, welcher eine neue Adresse zugewiesen wurde, wird beispielsweise durch einen *3xx-Statuscode* signalisiert. Eine Ressource, welche soeben dynamisch von einem Skript erstellt wurde, erhält dagegen den *Statuscode* «201 Created». Bei der angegebenen Adresse handelt es sich um eine absolute Angabe, wie zum Beispiel:

Location: http://www.example.org/beispiel.html

- Proxy-Authenticate

Sobald ein *Client* eine Anfrage an einen *Proxy* sendet, welcher eine Authentifizierung benötigt, wird diese Anfrage zuerst vom *Proxy* mit dem *Statuscode* «407 Proxy Authentication Required» abgewiesen. Außerdem erhält die *Response-Nachricht* die Anweisung *Proxy-Authenticate*, mit welcher der *Proxy* dem *Client* mitteilt, dass eine Authentifizierung benötigt wird.

ANHANG A. NETZWERKGRUNDLAGEN

Der Parameter der Anweisung *Proxy-Authenticate* setzt sich aus dem Kodierungsschema und der Bereichsbeschreibung zusammen. Als Kodierungsschema sind die Angaben *Basic* und *Digest* möglich. Die Bereichsbeschreibung besteht aus «realm=»[Bereich]«. Die Angabe «[Bereich]» ist dabei ein frei wählbarer String. Der *Client* erkennt an dieser Bereichsangabe, für welchen Bereich er sich authentifizieren soll.

Eine mögliche Anforderung zur Authentifizierung kann beispielsweise wie folgt aussehen:

Proxy-Authenticate: Basic realm=»Der private Bereich der Website«

- **Retry-After**

Mit *Retry-After* kann der *Server* die Anfrage des *Clients* abweisen und den *Client* gleichzeitig dazu auffordern, nach einer gewissen Zeit eine erneute Anfrage zu senden. Dieses Vorgehen kann beispielsweise dazu verwendet werden, einen *Request* abzuweisen, weil der *Server* momentan gewartet wird oder dieser ausgelastet ist.

Als Parameter von *Retry-After* kann entweder eine Zeit in Sekunden oder ein gültiges *HTTP-Datum* angegeben werden. Möchte der *Server* beispielsweise nach 160 Sekunden eine erneute Anfrage vom *Client* erhalten, so signalisiert er dies mit:

Retry-After: 160

- **Server**

Mit der *Server-Direktive* kann der *Server* dem *Client* Informationen über sich selbst mitsenden. Diese Informationen können beispielsweise Angaben über den *Webserver*, das verwendete *Serverbetriebssystem* oder die *PHP-Version* enthalten.

Diese Angaben über den *Server* und dessen Software können unter Umständen ein Sicherheitsrisiko darstellen, da potentielle Angreifer Informationen über das verwendete System und somit über eventuell enthaltene Schwachstellen erhalten.

Ein *Server* mit dem Betriebssystem *Windows 2000 Server*, dem Webserver *Apache 2.0.53* und *PHP 5.0.0* liefert beispielsweise die Angaben:

Server: Apache/2.0.53 (Win32) mod_ssl/2.0.53 OpenSSL/0.9.7e PHP/5.0.0

- **Set-Cookie**

Da es sich bei *HTTP* um ein zustandsloses Protokoll handelt, können Sitzungsdaten nicht direkt verwaltet werden. Hierzu wurden von der *Netscape Communications Corporation* [25] die *Cookies* nach RFC2109 [74] und RFC2965 [75] als Erweiterung des *HTTP-Headers* entwickelt.

ANHANG A. NETZWERKGRUNDLAGEN

Mit Hilfe der *Cookies* kann der *Server* mit der *HTTP-Response* zusätzliche Cookie-Informationen mit zum Browser senden. Diese werden auf dem Client lokal gespeichert. Anschließend überprüft der *Webbrowser* bei jedem *HTTP-Request*, ob ein *Cookie* von der entsprechenden *Website* existiert und sendet diesen gegebenenfalls mit.

Ein solcher *Cookie* besteht aus reinem Text und sollte eine Länge von vier Kilobytes nicht überschreiten. Jeder *Cookie* wird als kleine Textdatei auf dem *Client* gespeichert und auch von diesem verwaltet. Dies bedeutet auch, dass ein *Client* selbst entscheiden kann, welche *Cookies* er annimmt und wie er diese behandelt. Ob ein *Cookie* vom *Client* gespeichert wurde, kann ein *Webserver* nur durch einen weiteren *Request* erkennen.

Jeder *Cookie* besteht aus einem Namen und einem zugehörigen Wert. Bei Bedarf können außerdem noch zusätzliche Parameter angegeben werden. Alle Cookieangaben sind nachfolgend aufgelistet:

- Name

Der Name besteht aus einer beliebigen Kombination aus *ASCII-Zeichen*. Dieser Name identifiziert den angegebenen Wert eindeutig.

- Wert

Der Wert repräsentiert den eigentlichen Wert der «Variable» mit dem Namen «Name».

- Version

Diese Angabe gibt die *Cookie-Management-Spezifikation* in einer Dezimalzahl an.

- Expires

Mit dem *Expires-Parameter* gibt der *Server* bei *HTTP/1.0* den Zeitpunkt an, zu dem der *Client* den *Cookie* wieder löschen soll. Die Zeitangabe erfolgt dabei im Format *Universal Time Coordinated (UTC)* wie im Artikel «Koordinierte Weltzeit» [138] beschrieben.

- Max-age

Mit *Max-age* definiert der *Server* bei *HTTP/1.1* die Lebensdauer des *Cookies* in Sekunden. Mit der Angabe «0» wird der *Cookie* sofort gelöscht.

- Domain

Mit Hilfe der *Domain-Angabe* wird spezifiziert, für welche *Domain* der gesetzte *Cookie* gilt.

ANHANG A. NETZWERKGRUNDLAGEN

- Path

Mit *Path* kann die Gültigkeit des *Cookies* auf einen bestimmten Teil der *URL* beschränkt werden.

- Port

Durch die Angabe eines oder mehrerer Ports kann die Verwendung des *Cookies* weiter eingeschränkt werden.

- Comment

Durch die Angabe eines Kommentars mit *Comment* wird der *Cookie* näher beschrieben.

- CommentURL

Unter der *CommentURL* kann eine detailliertere Beschreibung des *Cookies* abgelegt werden.

- Secure

Mit *Secure* kann angegeben werden, dass der *Cookie* nur über eine sichere Verbindung übertragen werden soll. Die meisten *Clients* senden einen als «sicher» gekennzeichneten *Cookie* über eine *HTTPS-Verbindung* zurück.

- Discard

Durch *Discard* wird der *Webbrowser* angewiesen, den *Cookie* beim Beenden sofort zu löschen.

Ein mögliches Setzen eines *Cookies* kann mit Hilfe der oben genannten Parameter schließlich wie folgt aussehen:

```
Set-Cookie:  Farbe="rot"; Expires=Tue, 29-Mar-2005 19:30:42 GMT; Max-Age=
2592000; Path=/cgi-bin/auswertung.php; Version="1";
```

Bei einem anschließenden *Request* des *Client* wird den zusätzlichen Parametern ein «\$» vorangestellt. Die *Cookie-Anweisung* im *Request-Header* kann dann beispielsweise folgende Form haben:

```
Cookie: Farbe="rot"; $Path=/cgi-bin/auswertung.php; $Version="1";
```

Da die Cookie-Informationen beim Setzen und Lesen jeweils im *Header* übertragen werden, ist es auch möglich einen *Cookie* beispielsweise beim Übertragen einer Grafik mitzusenden.

ANHANG A. NETZWERKGRUNDLAGEN

Möchte ein *Server* einen *Cookie* gezielt löschen, so ist dies dem *Server* nur dadurch möglich, dass er den *Cookie* mit leeren Daten überschreibt.

Seit RFC2965 [75] existieren auch die Anweisungen *Cookie2* und *Set-Cookie2*, welche den älteren «Netscape-Standard» ablösen sollen.

- Vary

Durch die *Vary-Direktive* kann der *Server* dem *Client* mitteilen, dass von dem aktuellen Dokument noch weitere Versionen existieren. Diese zusätzlichen Versionen können sich beispielsweise in der Sprache oder der Zeichenkodierung von der aktuell angeforderten Version unterscheiden. Durch die zusätzliche Angabe von möglichen Anweisungen teilt der *Server* dem *Client* mit, wie die weiteren Versionen angefordert werden können. Der *Server* kann zum Beispiel folgende Zeile übertragen:

Vary: Accept-Language, Accept-Encoding

Der *Client* erkennt daran nun, dass noch andere Versionen des aktuellen Dokuments existieren, welche sich in Sprache und Zeichenkodierung unterscheiden. Durch die gezielte Angabe einer *Accept-Language* kann der *Client* anschließend nach der Version des Dokuments in einer anderen Sprache fragen.

- WWW-Authenticate

Durch *WWW-Authenticate* kann der *Server* dem *Client* mitteilen, dass für die aktuell angeforderte Ressource eine Authentifizierung notwendig ist. Die möglichen Parameter sowie die eigentliche Authentifizierung verlaufen dabei analog zu den unter *Proxy-Authenticate* vorgestellten Angaben und Verfahren.

Eine mögliche *WWW-Authenticate-Anweisung* kann schließlich wie folgt aussehen:

WWW-Authenticate: Basic realm="Der private Bereich der Website"

A.7.3.2.5. Response-Codes Durch die sogenannten *Response-Codes* kann der *Server* dem *Client* eine Statusmeldung über die aktuelle Operation senden. Bei den *Response-Codes* handelt es sich um dreistellige Zahlen im Bereich von 100 bis 599. Die einzelnen *Response-Codes* befinden sich in bestimmten Bereichen (1xx, 2xx, 3xx, 4xx, 5xx). Jeder dieser Bereiche stellt dabei eine bestimmte Grundinformation über den Status zur Verfügung. Neben den im Folgenden genannten Statuscodes kann ein *Server* auch eigene Statuscodes verwenden. In diesem Fall muss aber sichergestellt werden, dass die beteiligten *Clients* diese Statuscodes ebenfalls unterstützen.

ANHANG A. NETZWERKGRUNDLAGEN

- 1xx - Informationen

Die *Response-Codes* im Bereich 100 bis 199 enthalten lediglich allgemeine Informationen.

- 100 Continue

Durch *100 Continue* teilt der *Server* dem *Client* mit, dass der aktuelle Teil einer mehrteiligen Nachricht erfolgreich empfangen wurde. Der *Client* kann daraufhin die weiteren Teile der Nachricht senden.

- 101 Switching Protocols

Hiermit wird dem *Client* mitgeteilt, dass die verwendete Protokollversion erfolgreich gewechselt wird.

- 2xx - Erfolgreiche Operation

Durch die Statuscodes 200 bis 299 teilt der *Server* mit, dass der angeforderte *Request* erfolgreich war.

- 200 OK

Die Angabe *200 OK* ist die Standardantwort bei erfolgreichem Ausliefern einer Ressource.

- 201 Created

Mit dieser Statusmeldung wird signalisiert, dass der aktuelle *Request* erfolgreich war und die angeforderte Ressource erstellt wurde. Zusätzlich zu dieser Statusmeldung wird immer die Headeranweisung *Location* mit angegeben, um dem *Client* die *URL* der neuen Ressource mitzuteilen.

- 202 Accepted

Dieser *Response-Code* teilt dem *Client* mit, dass der *Request* akzeptiert wurde, aber die Verarbeitung nicht fertiggestellt werden konnte. Weitere Informationen zu dem Problem sollten im anschließenden Nachrichtentext erwähnt werden.

- 203 Non-Authoritative Information

Hiermit wird dem *Client* gemeldet, dass die Daten nicht vom *Server*, sondern von einer lokalen oder sonstigen externen Quelle stammen.

ANHANG A. NETZWERKGRUNDLAGEN

– 204 No Content

Durch diesen *Response-Code* wird signalisiert, dass die Anfrage vom *Server* zwar verarbeitet wurde, die aktuelle Nachricht aber keine Daten des *Body* beinhaltet.

– 205 Reset Content

Mit *205 Reset Content* erhält der *Client* die Information, dass die Daten der Anfrage verarbeitet wurden und er sie zurücksetzen soll. Diese Information wird zum Beispiel nach dem Übertragen von Formulardaten verwendet.

– 206 Partial Content

Mit dieser Meldung teilt der *Server* dem *Client* mit, dass der angeforderte Teil (*Range*) der Daten ausgeliefert wird.

• 3xx - Weiterleitung

Die Statuscodes im Bereich 300 bis 399 teilen dem *Client* mit, dass zur erfolgreichen Bearbeitung der Anfrage weitere Schritte seitens des *Client* notwendig sind. Dies kann beispielsweise der Fall sein, wenn sich die Adresse einer Ressource auf dem *Server* geändert hat und der *Client* dadurch seine Anfrage an einen anderen Pfad richten muss.

– 300 Multiple Choices

Durch Statuscode 300 zeigt der *Server* an, dass es für die aktuelle Ressource mehrere Quellen gibt. Im weiteren Teil der Nachricht werden hierzu die möglichen Alternativ-Ressourcen angegeben.

– 301 Moved Permanently

Durch *301 Moved Permanently* erfährt der *Client*, dass sich die Adresse der angeforderten Ressource permanent geändert hat. Um dem *Client* die aktuelle Adresse zu übergeben, wird zusätzlich die *Location-Anweisung* verwendet.

– 302 Found

Durch diese Statusmeldung zeigt der *Server* an, dass die angeforderte Ressource temporär unter einer anderen *URL* zu finden ist. Die neue Position wird wieder durch die *Location-Direktive* übergeben.

ANHANG A. NETZWERKGRUNDLAGEN

– 303 See Other

Hiermit teilt der *Server* mit, dass die aktuelle Anfrage entweder an eine andere Adresse gestellt oder eine andere Anfragemethode verwendet werden soll.

– 304 Not Modified

304 Not Modified teilt dem *Client* mit, dass die Ressource seit dem mit *If-Modified-Since* oder *If-None-Match* angegebenen Datum nicht verändert wurde und daher nicht gesendet wird. Stattdessen wird der *Client* aufgefordert, die Ressource aus dem lokalen *Cache* zu laden.

– 305 Use Proxy

Hiermit kann der *Server* den *Client* auffordern, die aktuelle Anfrage über einen bestimmten *Proxy-Server* zu stellen. Mit Hilfe der *Location-Anweisung* wird im *Header* der gewünschte *Proxy* angegeben.

– 307 Temporary Redirect

Durch die Statusmeldung *307 Temporary Redirect* zeigt der *Server* an, dass die angeforderte Ressource temporär unter einer anderen *URL* zu finden ist. Zusätzlich wird die *Location-Direktive* angegeben, um dem *Client* die neue Adresse mitzuteilen.

Der *Client* soll die neue *URL* jedoch nur für diese Anfrage verwenden, für zukünftige Anfragen soll weiterhin die «alte» *URL* verwendet werden.

• 4xx - Client-Fehler

Die Codes 400 bis 499 zeigen an, dass der *Client* einen Fehler verursacht hat.

– 400 Bad Request

Mit dieser Meldung wird dem *Client* angezeigt, dass der *Request* einen Syntaxfehler enthielt.

– 401 Unauthorized

401 Unauthorized zeigt an, dass der *Client* sich nicht authentifiziert hat. Dieser Statuscode wird beispielsweise zusammen mit der *WWW-Authenticate-Anweisung* verschickt.

– 402 Payment Required

Dieser Statuscode ist bisher nur reserviert.

ANHANG A. NETZWERKGRUNDLAGEN

– 403 Forbidden

Mit dieser Statusmeldung kann ein *Server* eine Anfrage ablehnen.

– 404 Not Found

Bei Statuscode *404 Not Found* handelt es sich wohl um den bekanntesten *HTTP-Statuscode*. Er teilt dem *Client* mit, dass die angeforderte Ressource nicht existiert.

– 405 Method Not Allowed

Mit Hilfe dieses Statuscodes teilt der *Server* mit, dass die vom *Client* angegebene Methode bei der aktuellen Ressource nicht erlaubt ist. Zusätzlich sendet der *Server* eine *Allow-Anweisung* mit.

– 406 Not Acceptable

Diese Statusmeldung zeigt an, dass die angeforderte Datei zwar auf dem *Server* existiert, diese aber nicht in dem vom *Client* angeforderten Format vorliegt. Der *Server* sendet zusätzlich zu dieser Statusmeldung die Anweisungen *Content-Language*, *Content-Encoding* und *Content-Type*.

– 407 Proxy Authentication Required

Sobald ein nicht authentifizierter *Client* versucht, auf einen *Proxy* zuzugreifen, erhält er diesen Statuscode zurück. Zusätzlich zum Statuscode wird in diesem Fall die *Proxy-Authenticate-Anweisung* verschickt.

– 408 Request Timeout

Wenn der *Server* innerhalb der zur Verfügung stehenden Zeit nicht antworten kann, teilt er dies dem *Client* durch *408 Request Timeout* mit.

– 409 Conflict

Durch diesen *Response-Code* zeigt der *Server* an, dass es durch die Anfrage zu einem Konflikt innerhalb des *Servers* kam und die Anfrage daher nicht bearbeitet werden konnte.

– 410 Gone

Durch diese Meldung erhält der *Client* die Information, dass die angeforderte Ressource nicht mehr existiert und auch in Zukunft nicht mehr existieren wird.

ANHANG A. NETZWERKGRUNDLAGEN

– 411 Length Required

Dieser Code teilt dem *Client* mit, dass der *Server* den *Request* nicht ohne die Angabe des *Content-Length-Headers* akzeptiert.

– 412 Precondition Failed

Wenn der *Client* eine If-Anfrage an den *Server* stellt und der *Server* keine Datei mit dem passenden *ETag* findet, so antwortet der *Server* mit dieser Statusmeldung.

– 413 Request Entity Too Large

Der *Server* antwortet mit diesem *Response-Code*, wenn er die Anfrage nicht verarbeiten konnte, weil der *Body* der Nachricht zu groß war.

– 414 Request-URI Too Long

Diesen Statuscode erhält der *Client*, wenn die *URL* in seiner Anfrage zu lang war und der *Request* daher nicht bearbeitet werden konnte.

– 415 Unsupported Media Type

Wenn der *Server* eine Anfrage mit einem nicht unterstützten *Mediatyp* erhält, antwortet er dem *Client* mit *415 Unsupported Media Type*.

– 416 Requested Range Not Satisfiable

Stellt ein *Client* eine Anfrage nach einem nicht vorhandenen Byte-Bereich (*Range*), so signalisiert ihm der *Server* dies durch den Statuscode 416.

– 417 Expectation Failed

Für den Fall, dass ein *Server* die Anforderungen der *Expect-Anweisung* im *Request-Header* nicht erfüllen kann, teilt er dies dem *Client* durch diese Meldung mit.

• 5xx - Server-Fehler

Mit diesen Statuscodes werden dem *Client* Serverfehler signalisiert.

– 500 Internal Server Error

Interne Fehler im *Server* werden durch den Statuscode 500 angezeigt.

ANHANG A. NETZWERKGRUNDLAGEN

– 501 Not Implemented

Wenn der *Server* eine vom *Client* angeforderte Aktion nicht ausführen kann, weil diese im *Server* nicht implementiert ist, antwortet er mit *501 Not Implemented*.

– 502 Bad Gateway

Diese Statusmeldung zeigt an, dass der *Server* oder *Proxy* von einem anderen *Server* oder *Proxy* eine ungültige *Response* erhalten hat.

– 503 Service Unavailable

Möchte der *Server* mit der *Retry-After-Anweisung* anzeigen, dass er momentan nicht erreichbar ist, es aber demnächst wieder sein wird, so sendet er dem *Client* diesen Statuscode.

– 504 Gateway Timeout

Mit dieser Meldung wird dem *Client* signalisiert, dass ein *Proxy* oder ein *Server* den *Request* nicht rechtzeitig verarbeiten konnte.

– 505 HTTP Version Not Supported

Dieser Statuscode zeigt an, dass der *Server* die verwendete *HTTP-Version* nicht unterstützt.

A.8. Simple Network Management Protocol

Anwendungsschicht		HTTP	SNMP	
Transportschicht		TCP	UDP	
Internetschicht	ICMP			
Internetschicht	IP			ARP
Netzzugangsschicht	Ethernet			

Tabelle A.22.: Position von SNMP im TCP/IP-Referenzmodell

Das *Simple Network Management Protocol*, kurz *SNMP*, ist ein Netzwerkprotokoll zum Steuern und Überwachen von Netzwerkgeräten, wie *Routern*, *Servern*, *Switches* oder *Druckern*, von einer

ANHANG A. NETZWERKGRUNDLAGEN

zentralen Stelle aus. Detaillierte Informationen zum Protokoll sind unter «SNMP Simple Network Management Protocol» [34], «Entwurf und Implementierung eines Netzwerk-Management-Systems auf Basis von SNMP» [31] und «The Simpleweb - Tutorial slides in PDF format» [117] zu finden. Die erste Version des Protokolls wurde 1987 als *Simple Gateway Management Protocol* in der RFC1028 [26] veröffentlicht. Die Bezeichnung enthält in dieser frühen Version noch das Wort «Gateway», da diese Verbindungsknoten damals das Hauptproblem in Netzwerken waren. Im Jahr 1988 erschienen dann die Dokumente «Structure and Identification of Management Information for TCP/IP-based internets» RFC1065 [111], «Management Information Base for Network Management of TCP/IP-based internets» RFC1066 [110] und «A Simple Network Management Protocol» RFC1067 [18]. 1989 erhielt *SNMP* schließlich den Status «recommended» und es wurden die ersten Anwendungen auf Messen vorgestellt. Ein Jahr später erschienen die Dokumente «Structure and Identification of Management Information for TCP/IP-based Internets» RFC1155 [112], «A Simple Network Management Protocol (SNMP)» RFC1157 [19] und das Dokument «Management Information Base for Network Management of TCP/IP-based internets» RFC1156 [86], welches die *MIB-I* beschreibt. Wiederum ein Jahr später erschien die Managed Information Base 2 (MIB-II) in der RFC1213 [87] und im Jahr 1992 begann die Entwicklung von *SNMPv2*.

Der Begriff *SNMP* ist bei dieser Managementlösung eigentlich nur der Name einer Komponente und zwar des Protokolls. Der Begriff wird jedoch oft als Bezeichnung für die gesamte Lösung, also auch für das Konzept und das Verfahren, verwendet. In dieser Ausarbeitung wird von diesem Umstand ebenfalls Gebrauch gemacht und die gesamte Managementlösung wird als *SNMP* bezeichnet.

Die Aufgaben eines solchen Netzwerk-Managements sind laut dem Buch «SNMP - Konzepte, Verfahren, Plattformen» [64] von *Rainer Janssen* und *Wolfgang Schott* wie folgt:

- Gewährleistung einer konstant hohen Dienstgüte (quality of service) des Netzes
- flexible Anpassung des Netzes an veränderte Anforderungen
- effiziente Nutzung der vorhandenen Netzressourcen zur Erreichung eines optimalen Kosten-Nutzen-Verhältnisses

Diese Ziele können nur durch Überwachung und Steuerung der Ressourcen und Austausch von Managementinformationen erreicht werden, wie Abbildung A.20 zeigt.

Die Zielsetzung von *SNMP* wird schon durch das Wort «Simple» im Namen beziehungsweise das *Fundamental Axiom* aus «The Simple Book» [113] ausgedrückt:

«Die Auswirkung von Netzmanagement auf die zu verwaltenden Netzelemente muß gering bleiben und sich am kleinsten gemeinsamen Nenner orientieren.»

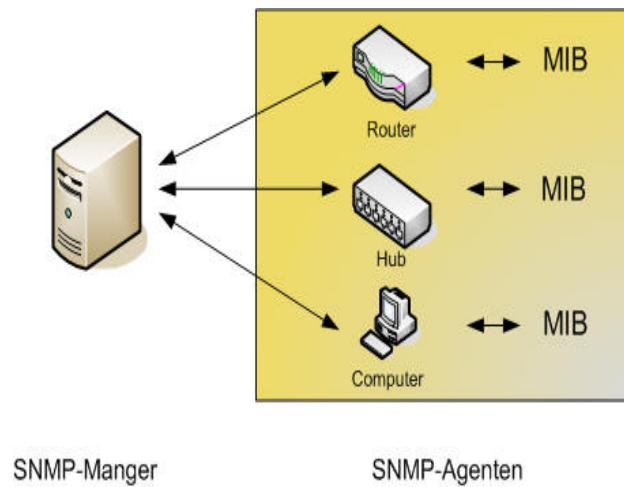


Abbildung A.20.: Aufbau eines SNMP-Systems [42]



Abbildung A.21.: Dr. Jeffrey D. Case, alias «Dr. SNMP» [118]

Dr. Jeffrey D. Case (Abbildung A.21), Mitbegründer von *SNMP* und auch als *Dr. SNMP* bekannt, hat dies sehr schön durch folgenden Satz ausgedrückt:

«I want my dog hunting raccoon, not skinning them !»

Es handelt sich bei *SNMP* um ein symmetrisches, asynchrones Request-Response-Protokoll. Dies bedeutet:

- Nachrichten werden spontan gesendet, ohne vorherige Ankündigung
- Jeder *Request* wird mit einer *Response* beantwortet (*TRAP-Nachrichten* bilden hierbei die einzige Ausnahme)

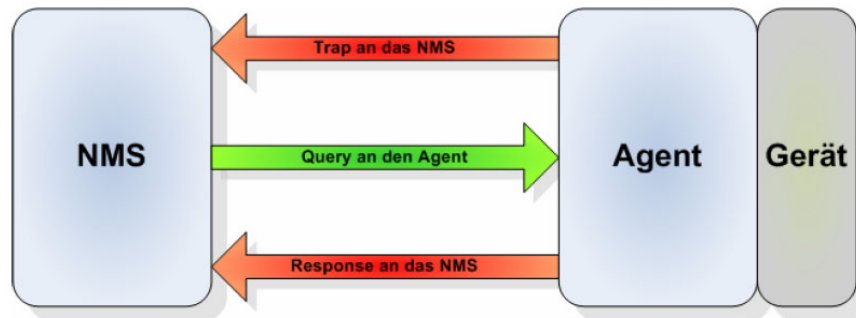


Abbildung A.22.: Beispiel eines aufgesetzten SNMP-Agents [34]

- Es können mehrere Anfragen auf einmal versendet werden. Die Antworten werden danach durch den *Request Identifier*, also die *Anfrage ID* korrekt zugeordnet
- Nachrichten können verloren gehen
- Jede *SNMP-Entity* kann grundsätzlich alle *PDU-Typen* (*Personal Data Unit (PDU)*) senden und empfangen. Daher wird oft angegeben, ob eine *Entity* als *Manager* oder *Agent* arbeitet

SNMP sollte auf einem verbindungslosen Protokoll basieren, so dass im Fehlerfall das Netzwerk nicht noch durch einen zusätzlichen Overhead des Protokolls belastet wird. Da *SNMP* erlaubt, dass Nachrichten verloren gehen dürfen, ist es kein Problem, *SNMP* auf einem Protokoll, wie beispielsweise *UDP*, basieren zu lassen. *UDP* ist das meistgenutzte Transportprotokoll für *SNMP-Nachrichten*, obwohl auch andere Protokolle hierfür verwendet werden können.

A.8.1. Aufbau eines SNMP-Systems

Ein solches *SNMP-System* besteht aus einem *SNMP-Manager* und *SNMP-Agents*. Der *Manager* läuft hierbei auf der *Network-Management-Station (NMS)*, die *Agents* laufen auf den *Managed Network Entity (MNE)* beziehungsweise den sogenannten *Managed Nodes*. Die *Agents* haben die Aufgabe, in den Netzressourcen Informationen zu sammeln und diese dem *Manager* bei Bedarf zur Verfügung zu stellen. Abbildung A.22 verdeutlicht diesen Zusammenhang. Verfügt ein Gerät über keinen eigenen *SNMP-Agent*, so kann es über einen sogenannten *Proxy-Agent* verwaltet werden (Abbildung A.23).

Eine vom *Agent* verwaltete Variable wird als *Managed Object (MO)* bezeichnet. Bei *SNMPv1* stehen einem *SNMP-Manager* insgesamt drei Befehle zur Verfügung, um Anfragen an einen *SNMP-Agent* zu richten, um dessen *Managed Objects* auslesen und verändern zu können. Die Anfragen werden an den Port 161 des *Agent* gerichtet.

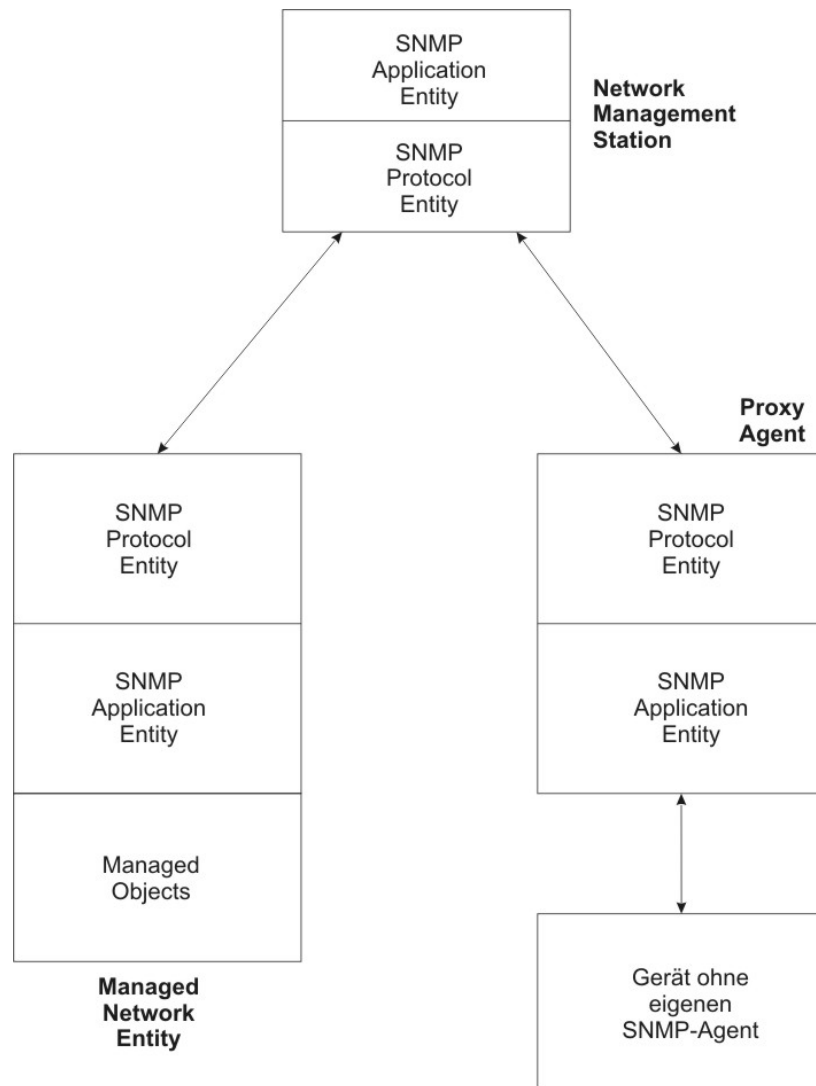


Abbildung A.23.: Verwendung eines Proxy-Agent

ANHANG A. NETZWERKGRUNDLAGEN

- GET

Mit *GET* kann der *Manager* einen Datensatz vom *Agent* anfordern.

- GETNEXT

Mit *GETNEXT* fordert der *Manager* den nächsten vorhandenen Datensatz an.

- SET

Mit diesem Befehl können Datensätze geändert werden. Es ist dabei auch möglich, gleichzeitig mehrere Datensätze zu modifizieren.

Ein *SNMP-Agent* dagegen besitzt zwei Möglichkeiten, um einem *Manager* Daten zukommen zu lassen.

- RESPONSE

Mit *RESPONSE* antwortet ein *Agent* auf eine Anfrage. Die Daten werden hierbei an den *Port* gerichtet, von dem aus auf der *Manager-Seite* die Anfrage gesendet wurde.

- TRAP

Mit *TRAP-Nachrichten* können *Agents* unaufgefordert Nachrichten an *Network-Management-Stations* senden, um diesen über ein Ereignis zu benachrichtigen. *TRAP-Nachrichten* werden an *Port 162* des *SNMP-Managers* gesendet.

Ein *Manager* bekommt seine Informationen also entweder über eine *RESPONSE*- oder eine *TRAP-Nachricht*. Bei den *RESPONSE-Nachrichten* muss der *SNMP-Manager* die Werte per *Polling* abfragen. Die *TRAPs* hingegen werden vom *Agent* angetriggert. Bei einer Implementierung von *Polling* und *TRAPs* sollte darauf geachtet werden, dass beide Möglichkeiten im passenden Verhältnis verwendet werden. Würde ein *Manager* beispielsweise zu viel *Polling* verwenden, würde das gegen das *Fundamental Axiom* verstoßen.

Die Anwendungen, welche auf den *Network-Management-Stations* und den *Managed-Network-Entities* laufen, sind die *SNMP Application Entities*. Diese gehören jeweils einer Gruppe, der sogenannten *SNMP-Community* an. Diese *Communities* besitzen eindeutige Namen und haben das Datenformat «String». Durch diese Gruppennamen können nur *Application Entities*, welche zur gleichen *Community* gehören, miteinander kommunizieren, wie auch Abbildung A.24 verdeutlicht.

Da es sich bei *SNMP* um ein verbindungsloses Protokoll handelt, muss der *Community Name* in jeder Nachricht mit übertragen werden.

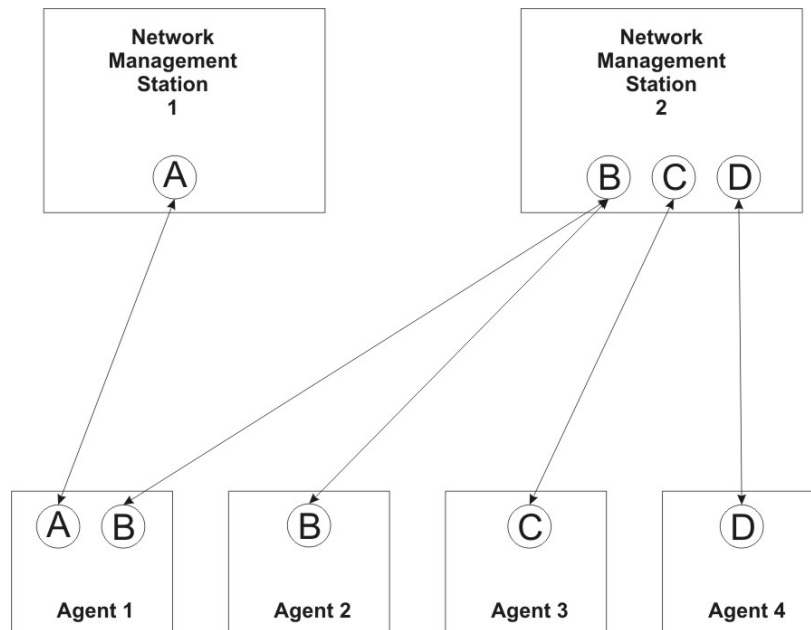


Abbildung A.24.: Kommunikation verschiedener SNMP-Communities

A.8.2. Managed Information Base

Die auf einem *Agent* vorhandenen Werte, welche ausgelesen und teilweise verändert werden können, werden als *Managed Objects* bezeichnet. Diese *Managed Objects* werden durch ein Datenmodell mit der Bezeichnung *Managed Information Base* (MIB) in der RFC1156 [86] beschrieben. Dieses Datenmodell wird in der Beschreibungssprache *Structure of Management Information (SMI)* geschrieben, welche auf der *Abstract Syntax Notation One (ASN.1)* basiert. Von der *SMI* existieren inzwischen zwei Versionen, die *SMIv1* nach RFC1065 [111] und RFC1155 [112] sowie die *SMIv2* nach RFC2578 [84], RFC2579 [85] und RFC2580 [83].

Die *MIB* weist eine Baumstruktur auf, in welcher die Zweige durch Nummern und alphanumerische Bezeichnungen dargestellt sind. Abbildung A.25 zeigt eine solche Baumstruktur mit Angabe der Nummern und alphanumerischen Bezeichnungen.

Jedes Objekt eines *SNMP-Agent*, also jede ausles- bzw. änderbare Variable, besitzt einen eindeutigen Namen, den *Object Identifier (OID)*. Durch diese Adresse wird das Objekt eindeutig in der *MIB-Baumstruktur* identifiziert beziehungsweise adressiert. Die *sysUpTime* kann auf diese Weise beispielsweise durch die Adresse «1.3.6.1.2.1.1.3» oder durch «iso.org.dod.internet.mgmt.mib.system.sysUpTime» angesprochen werden. Der Punkt am Anfang einer *OID* gibt an, dass diese *OID* als Referenz den Wurzelknoten (*root*) besitzt. Bei *OIDs* ohne führenden Punkt handelt es sich um relative *OIDs*, welche in den *RFCs* allerdings nicht offiziell vorgesehen sind. Viele *SNMP-Manager*

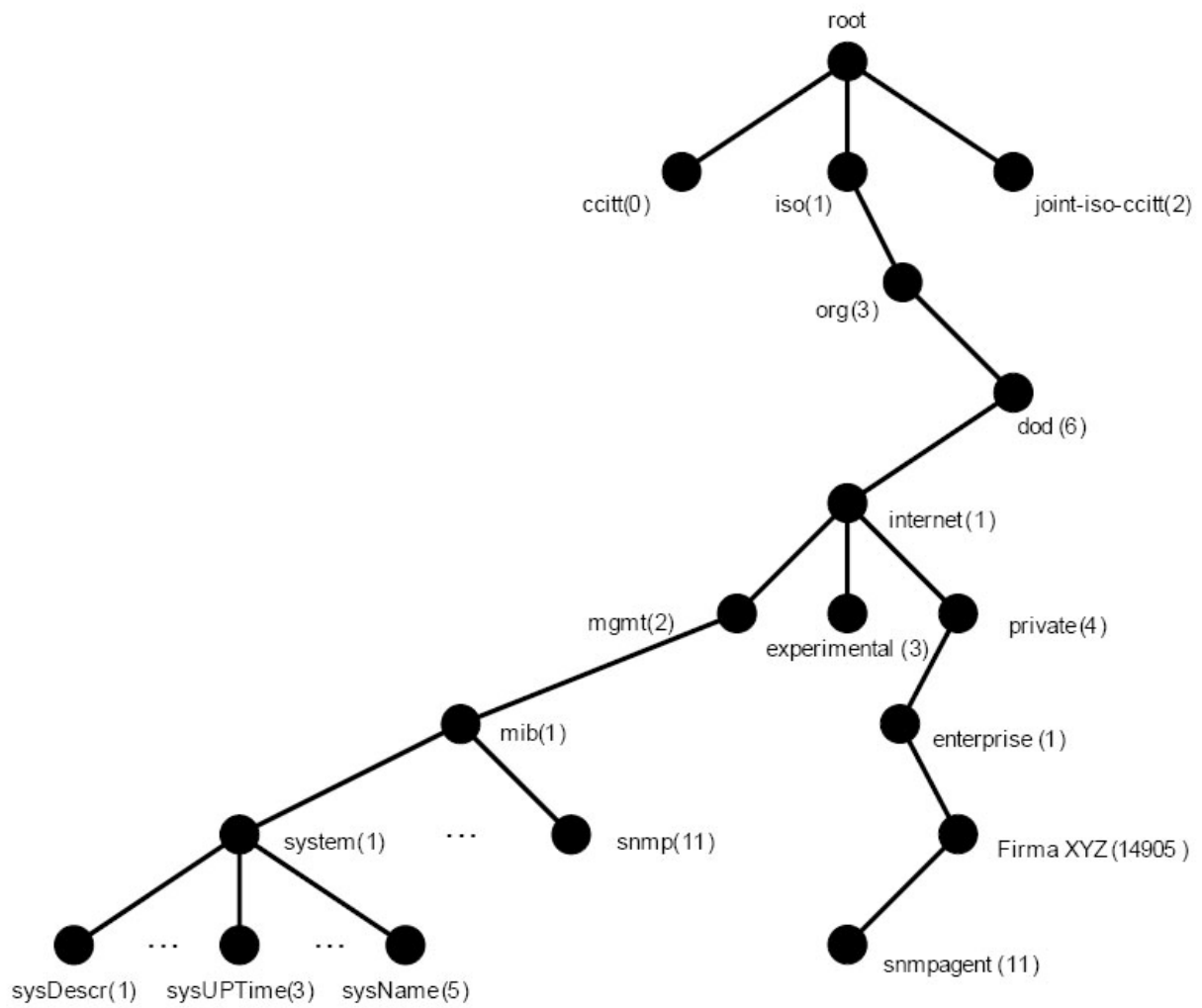


Abbildung A.25.: Struktur des MIB-I-Baums [7]

ANHANG A. NETZWERKGRUNDLAGEN

unterstützen diese relativen Adressen dennoch. Bei der Angabe einer relativen *OID* kann die absolute *OID* durch die Form «*Global OID* + *Relative OID* = *Absolute OID*» bestimmt werden. Da die relativen Adressierungen kein offizieller Standard sind, gibt es auch keine Vorschrift, welche Form die *Global OID* haben muss. Die meisten *SNMP-Implementierungen* verwenden jedoch für die *Global OID* entweder die Adresse *.1.3.6.1* oder die *.1.3.6.1.2.1*. Die relative Angabe *2.1.1.1.0* adressiert bei einer *Global OID* von *.1.3.6.1* also ebenso die *sysDescr* wie auch die absolute Adressangabe *.1.3.6.1.2.1.1.0*. Das Verwenden von relativen Adressen ist mit den alphanumerischen Bezeichnern ebenso möglich wie mit den numerischen Bezeichnern.

A.8.2.1. Systeminformationen

Zu den Anfangszeiten von *SNMP* hatte der *MIB-Tree* die Form nach *MIB-I*, welche in RFC1156 [86] beschrieben ist. Da mit *SNMPv2* und *SNMPv3* viele neue Parameter hinzukamen, welche auch verwaltet werden müssen, musste die Baumstruktur zu *MIB-II* erweitert werden. Beide dieser Strukturversionen enthalten unter «*.1.3.6.1.2.1*» einen Eintrag, welcher das Zielsystem beschreibt. Der Eintrag hat unter *MIB-I* den Namen «*mib*», unter *MIB-II* heißt er «*mib-2*». Dieser Knoten enthält wiederum Unterknoten. Die folgende Aufstellung zeigt einen kleinen Ausschnitt dieser Unterknoten:

- System

Dieser Unterknoten enthält allgemeine Informationen über das System, wie beispielsweise den Ansprechpartner, den Standort oder eine kurze Systembeschreibung.

- Interface

Dieser Zweig stellt Informationen über die vorhandenen Schnittstellen, deren Status und beispielsweise deren Übertragungsgeschwindigkeit dar.

- IP

Über diesen Unterpunkt können Informationen über das *Internet Protokoll* abgerufen werden. Diese Informationen enthalten Angaben wie beispielsweise die Anzahl der gesendeten Pakete oder die Anzahl der aufgetretenen Fehler.

- TCP

Dieser Eintrag enthält Informationen über alle *TCP-Verbindungen*.

ANHANG A. NETZWERKGRUNDLAGEN

- UDP

In diesem Unterknoten sind die Details zu den *UDP-Verbindungen* zu finden.

Jeder *SNMP-Agent* muss den Zweig *MIB-I* beziehungsweise *MIB-II* implementiert haben, um Informationen über das System zur Verfügung stellen zu können.

A.8.2.2. Private OIDs

Die bisher vorgestellten Zweige enthalten allgemeine Informationen zu den Systemen. Viele Geräte enthalten jedoch mehr Optionen als durch diese Standardzweige abgedeckt werden. Hierzu gibt es unter «1.3.6.1.4» beziehungsweise unter «iso.org.dod.internet.private» einen Zweig, der für eigene Aufgaben nach Bedarf gestalten werden kann. Unterhalb des *private-Knotens* existiert ein Knoten mit dem Namen *enterprises*. In diesem können sich Firmen und Einrichtungen einen eigenen Unterknoten reservieren lassen, in welchem sie anschließend ihre einzelnen Geräte auflisten und deren Parameter darstellen können. Die Anmeldung eines eigenen Unterknotens kann über das Onlineformular «Private Enterprise Number (PEN) Request Template» [53] der IANA [51] erfolgen. Einige der bekanntesten Firmen beziehungsweise Einrichtungen sind unter den folgenden *OIDs* zu finden:

- IBM «1.3.6.1.4.1.2»
- Cisco «1.3.6.1.4.1.9»
- Audi AG «1.3.6.1.4.1.3195»
- Intel Corporation «1.3.6.1.4.1.343»
- AMD «1.3.6.1.4.1.3704»
- Microsoft «1.3.6.1.4.1.311»
- NASA Ames Research Center «1.3.6.1.4.1.324»
- DeTeMobil Deutsche Telekom MobilNet GmbH «1.3.6.1.4.1.6490»
- Fachhochschule Heilbronn «1.3.6.1.4.1.4766»

Eine Liste aller *Private Enterprise Numbers* ist auf der Website «PRIVATE ENTERPRISE NUMBERS» [54] der IANA [51] zu finden.

A.8.3. Aufbau der Beschreibungssprache ASN.1 und den «Basic Encoding Rules» (BER)

Da die lokalen Daten auf den Netzwerkteilnehmern oft in unterschiedlichen Formaten vorliegen³, muss eine formale Beschreibungssprache eingeführt werden, die von allen Geräten verstanden wird. Diese abstrakte Syntax wird in der Beschreibungssprache *Abstract Syntax Notation One* (ASN.1) dargestellt.

Diese Syntax definiert allerdings nur eine abstrakte Syntax, welche von allen Geräten verstanden wird. Es muss aber zusätzlich noch definiert werden, wie die Daten dieser Beschreibungssprache auf Bitebene übertragen werden. Für diese sogenannte *Transfer Syntax* werden die *Basic Encoding Rules* (BER) von *ASN.1* verwendet, welche bei der *International Telecommunication Union (ITU)* unter X.690 [130] nachgelesen werden können.

Bei diesen Regeln wird jede Variable durch drei Größen definiert:

1. Tag (Variablentyp)
2. Length (Länge des Value-Feldes)
3. Value (Variablenwert)

Durch die Angaben *Tag*, *Length* und *Value* wird diese Kodierung auch *TLV-Kodierung* genannt. Ein Variablenname ist hierbei nicht vorhanden. Nach den *BER* werden die Daten mit dem *MSB First* übertragen, also das höchstwertigste Bit zuerst.

Die Felder der *TLV-Kodierung* werden wie folgt erklärt verwendet:

A.8.3.1. Tag-Feld

Das *Tag-Feld* setzt sich nach Abbildung A.26 zusammen. Die *Tag-Klasse* ist hierbei durch Tabelle A.23 definiert. Bit sechs gibt an, ob es sich um einen einfachen (primitive, Bit 6 = 0) oder zusammengesetzten (constructed, Bit 6 = 1) Typ handelt. Einfache Datentypen sind *INTEGER*, *OCTET STRING*, *OBJECT IDENTIFIER* und *NULL*. Zusammengesetzte Datentypen sind dagegen *SEQUENCE* und *SEQUENCE OF*. Die niederwertigsten fünf Bits des *Tag-Feldes* repräsentieren die *Typkennung*, welche einen Wert von 1 bis 30 annehmen kann.

³Diese unterschiedlichen lokalen Datenformaten resultieren beispielsweise aus unterschiedlichen Betriebssystemen, wie Windows oder Linux, auf den Zielsystemen.

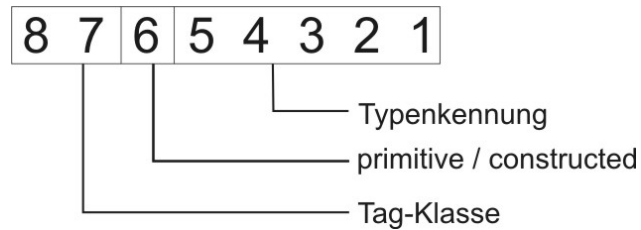


Abbildung A.26.: Aufbau des Tag-Felds

	Bit 8	Bit 7
UNIVERSAL	0	0
APPLICATION	0	1
Context-Specific	1	0
PRIVATE	1	1

Tabelle A.23.: Auflistung der Tag-Klassen

Einige mögliche *ASN.1-Kennungen* der SNMP-Datentypen für das *Tag-Feld* sind Tabelle A.24 zu entnehmen.

A.8.3.2. Length-Feld

Das *Length-Feld* gibt die Länge des nachfolgenden *Value-Feldes* an. Nach *ASN.1* sind hierbei zwei Arten der Längenangabe möglich: Die bestimmte (*definite*) Form und die unbestimmte (*indefinite*) Form. Da bei *SNMP* allerdings nur die bestimmte Form erlaubt ist, wird folgend nur auf diese näher eingegangen.

Diese bestimmte Form der Längenangabe wird wiederum in zwei Typen unterteilt, die *Short-Form* für Längenangaben von 0 bis 127 Bytes und die *Long-Form* für Längenangaben über 127 Bytes.

Die *Short-Form* besteht nur aus einem Byte für die Längenangabe. Das höchstwertigste Bit wird bei dieser Form auf null gesetzt und die restlichen sieben Bits enthalten die Längenangabe von 0 bis 127. Die *Long-Form* hingegen besteht aus mehreren Bytes zur Längenangabe. Beim ersten Byte wird dazu das höchstwertigste Bit auf eins gesetzt, was die *Long-Form* signalisiert. Mit den restlichen sieben Bits des ersten Bytes wird die Anzahl der Nachfolgebytes zur Längenangabe definiert. Der Wertebereich reicht hierbei von 0 bis 126, da der Wert 127 nicht erlaubt ist. Die anschließende Folge von Bytes gibt die eigentliche Längenangabe an. Der Wert wird dabei so dargestellt, dass die Bytes

ANHANG A. NETZWERKGRUNDLAGEN

Kennung	Bedeutung
0x02	INTEGER
0x04	OCTET STRING
0x05	NULL
0x06	OID
0x30	SEQUENZ
0x40	IP ADRESSE
0x43	TIME STAMP

Tabelle A.24.: Einige ASN.1-Kennungen der SNMP-Datentypen

Short Form:

Bits							
8	7	6	5	4	3	2	1
0	Längenangabe 0 - 127						

Long Form:

Bits							
8	7	6	5	4	3	2	1
1	Anzahl der Folgebytes 0 - 126						

Bits							
8	7	6	5	4	3	2	1
Eigentliche Längenangabe							

Beispiel für den Wert 201:

Bits							
8	7	6	5	4	3	2	1
1	0	0	0	0	0	0	1

Bits							
8	7	6	5	4	3	2	1
1	1	0	0	1	0	0	1

Abbildung A.27.: Verschiedenen Formen der Längenangabe

einfach aneinandergereiht werden. Abbildung A.27 zeigt jeweils ein Beispiel für die verschiedenen Formen der Längenangabe.

A.8.3.3. Value-Feld

Im *Value-Feld* wird der Wert der Variablen definiert. Das Feld darf leer sein (Länge = 0) oder aus beliebig vielen Bytes bestehen.

A.8.3.4. Beschreibung der ASN.1-Variablentypen

Die wichtigsten Variablentypen sind:

ANHANG A. NETZWERKGRUNDLAGEN

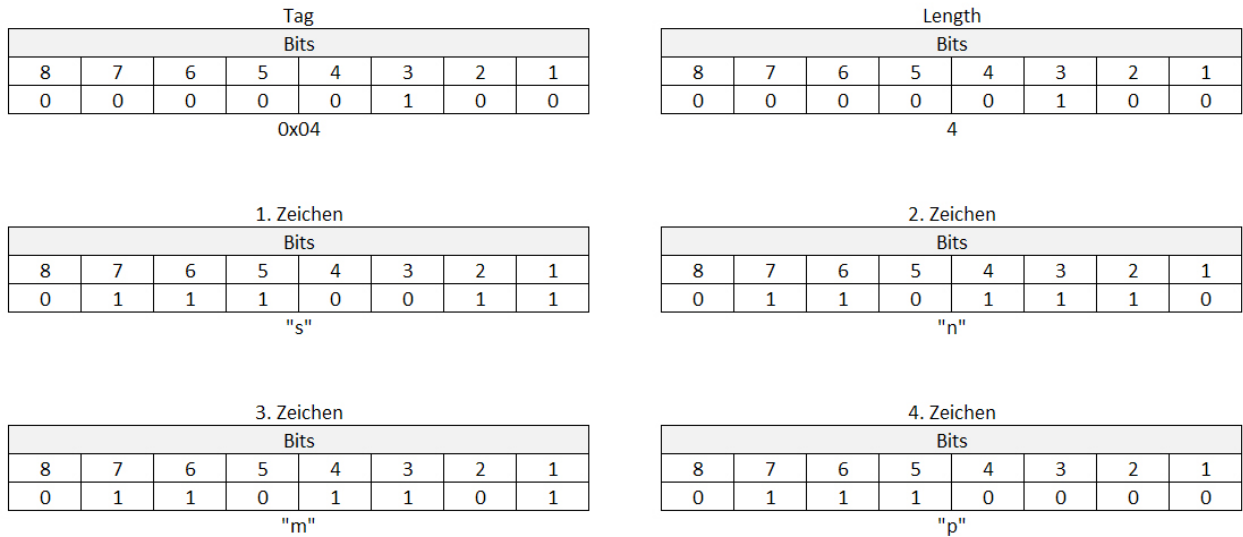


Abbildung A.28.: Beispiel für den Datentyp OCTET STRING

A.8.3.4.1. INTEGER Der *INTEGER-Typ* bekommt im *Tag-Feld* den Wert 0x02. Die Längenangabe variiert je nach der darzustellenden Zahl. Der Wert der Variablen wird im Zweierkomplement⁴ angegeben. Somit sind Ganzzahlen im Wertebereich von $-2^{n-1} \leq x \leq 2^{n-1} - 1$ möglich.

Bei der Darstellung des *INTEGER-Wertes* dürfen die ersten neun Bits nicht den gleichen Wert haben. Es darf also beispielsweise nicht vorkommen, dass die Bits 8 - 1 des ersten Bytes und das Bit 8 des zweiten Bytes alle auf «1» gesetzt sind.

A.8.3.4.2. OCTET STRING Mit diesem Datentyp können Stringvariablen dargestellt werden. Das *Tag-Feld* wird hierfür mit dem Wert 0x04 beschrieben. Das darauf folgende *Length-Feld* gibt die Stringlänge an und dahinter folgt der eigentliche String. Bei diesem steht der *ASCII-Code* für jedes Zeichen in einem eigenen Byte, wie Abbildung A.28 zu entnehmen ist.

A.8.3.4.3. NULL Bei *NULL* handelt es sich um eine leere Variable. Diese hat als *Tag* immer den Wert 0x05, eine Längenangabe von null und kein nachfolgendes *Value-Feld*.

⁴Beim Zweierkomplement werden die Bits des *Value-Feldes* von 0 bis n durchnummeriert, das niederwertigste Bit hat den Index 0. Werden bei den Bits 0 bis n-1 alle gesetzten Bits, ihrer Wertigkeit entsprechend, aufaddiert, resultiert dies in einer Zahl. Ein einzelnes Bit stellt hierbei den Wert 2^{Index} dar. Wenn das Bit n gesetzt ist, so muss von der erhaltenen Zahl 2^n abgezogen werden, um den endgültigen Wert der *INTEGER-Variablen* zu erhalten.

ANHANG A. NETZWERKGRUNDLAGEN

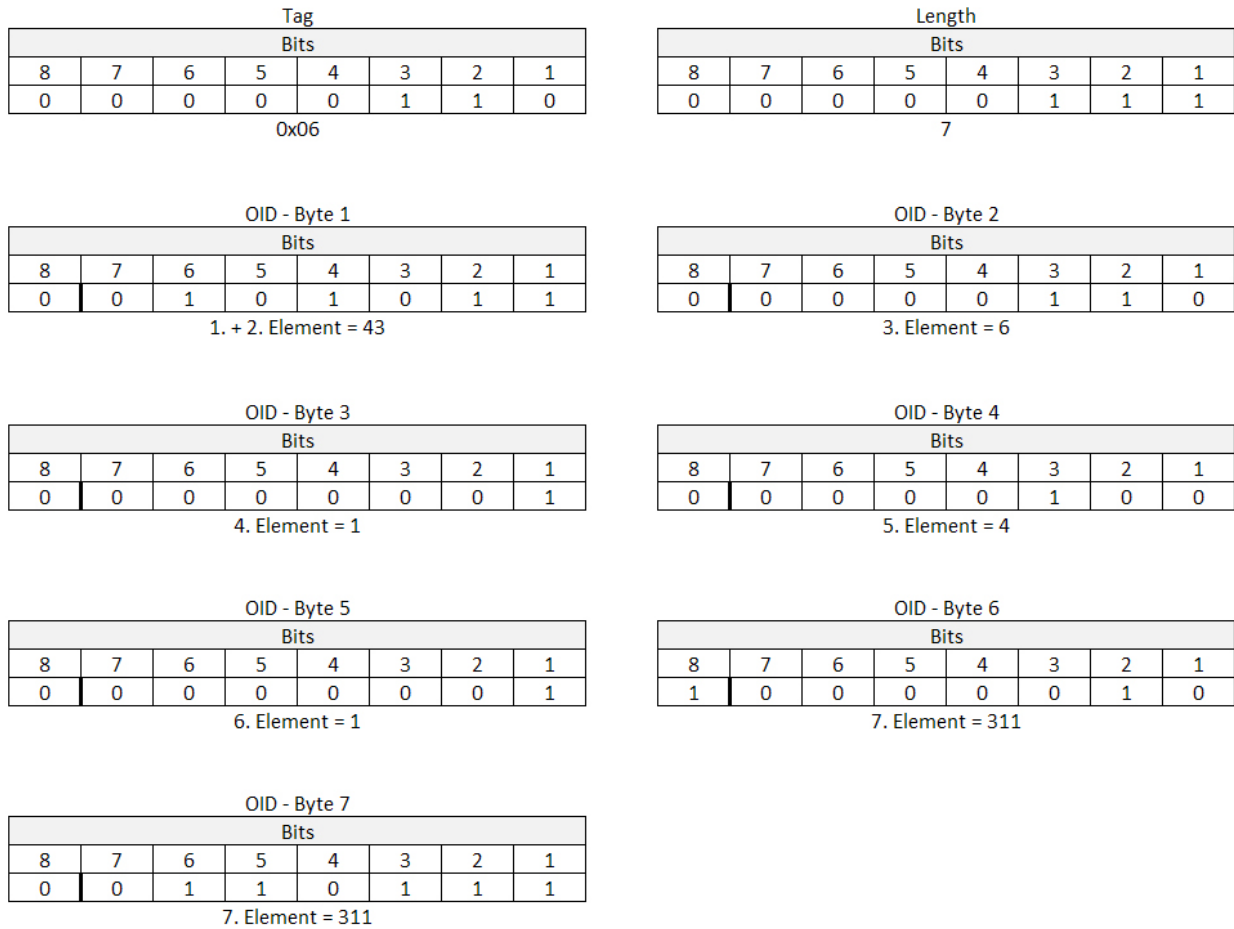


Abbildung A.29.: Beispiel für den Datentyp OBJECT IDENTIFIER

A.8.3.4.4. OBJECT IDENTIFIER Mit diesem Variablentyp kann die *OID*, also die Position eines Objekts im *MIB-Tree*, angegeben werden. Die Wurzel des *MIB-Tree* spaltet sich in die drei Zweige *ccitt(0)*, *iso(1)* und *joint-iso-ccitt(2)* auf. Heutzutage müssen die Zweige korrekterweise *itu-t(0)*, *iso(1)* und *joint-iso-itu-t(2)* heißen, da das *Comité Consultatif International Télégraphique et Téléphonique (CCITT)* am 1. März 1993 in *ITU Telecommunication Standardization Sector (ITU-T)* [129] umbenannt wurde.

Nach einer Definition der *CCITT* und der *International Organization for Standardization (ISO)* [62] sind in den Zweigen null und eins nie mehr als 40 Knoten (0-39) erlaubt. Diese Definition wird in den *BER* ausgenutzt, indem das erste Byte der *OBJECT IDENTIFIER* Variable als $40 \cdot e_1 + e_2$ definiert wird. e_1 und e_2 stehen hierbei für die Stellen eins und zwei der *OID*. Auf diese Weise können zwei Verzweigungen des *MIB-Baums* mit nur einem Wert ausgedrückt werden. Alle weiteren Elemente e_k der *OID* werden separat in einem oder mehreren Bytes kodiert.

Als Wert für das *Tag-Feld* erhält dieser Datentyp den Wert 0x06. Die Längenangabe im *Length-Feld*

ANHANG A. NETZWERKGRUNDLAGEN

richtet sich nach der Anzahl der nachfolgenden Bytes der *OID*. Bei den Bytes, welche zur Darstellung der *OID* verwendet werden, sind nur die Bits eins bis sieben zur Angabe des Wertes verfügbar. Bit acht hingegen wird dazu verwendet, anzuzeigen, ob der Wert in nur einem Byte steht (Bit 8 = 0) oder ob der Wert größer 127 ist und daher auf mehrere Bytes verteilt werden muss. Muss der Wert verteilt werden, enthält nur das letzte Byte des Werts eine Null an der Stelle des höchstwertigsten Bits. Alle vorigen Bytes zeigen mit einer Eins an dieser Stelle an, dass noch weitere Bytes folgen.

Abbildung A.29 zeigt zur Verdeutlichung die Darstellung der *OID* .1.3.6.1.4.1.311 («iso.org.dod.internet.private.enterprise.microsoft»). Der Teil «1.3» wird hierbei über $40 \cdot 1 + 3 = 43$ zusammengefasst. Die einzelnen Stellen der Angabe «6.1.4.1» werden danach in je einem Byte dargestellt und schließlich folgt der Adressteil «311», welcher in zwei Bytes aufgeteilt werden muss. Bei dieser Aufteilung ist zu erkennen, dass bei dem ersten Byte das höchstwertigste Bit gesetzt ist und somit Nachfolgebytes anzeigt.

Bei dem ersten Element der *OID* können auch andere Werte als null oder eins verwendet werden. In diesem Fall existiert keine Begrenzung auf 40 Unterknoten. Dennoch wird zur Angabe der *OID* weiterhin die Form $40 \cdot e_1 + e_2$ verwendet, was unter Umständen darin resultiert, dass das Ergebnis nicht mehr mit einem Byte ausgedrückt werden kann und in mehrere Bytes zerlegt werden muss.

A.8.3.4.5. SEQUENCE Der Datentyp *SEQUENCE* ist vergleichbar mit einer Struktur in der Programmiersprache C. In ihm können mehrere Datentypen zu einem neuen zusammengesetzt werden.

Ein Beispiel eines solchen Datentyps ist:

```
Adresse::=SEQUENCE {  
    ort OCTET STRING,  
    plz INTEGER }
```

Dieser neue Typ mit dem Namen «Adresse» besteht nun aus einem Teil «ort» vom Typ *OCTET STRING* und einem Teil «plz» vom Typ *INTEGER*. Werden für das Beispiel der Ort «Berlin» mit der Postleitzahl «13405» gewählt, enthält der *BER-kodierte* Datenstrom die Daten wie in Abbildung A.30 gezeigt.

Bei diesem Beispiel sind die Typen *OCTET STRING* und *INTEGER* in die *SEQUENCE* eingebunden. Die Längenangabe der *SEQUENCE* erstreckt sich dabei über den ganzen Block, bestehend aus den beiden eingebetteten Variablen.

ANHANG A. NETZWERKGRUNDLAGEN

SEQUENCE

Tag							
Bits							
8	7	6	5	4	3	2	1
0	0	1	1	0	0	0	0

0x30

Length							
Bits							
8	7	6	5	4	3	2	1
0	0	0	0	1	1	0	0

0x0C

Ort

Tag							
Bits							
8	7	6	5	4	3	2	1
0	0	0	0	0	1	0	0

0x04

Length							
Bits							
8	7	6	5	4	3	2	1
0	0	0	0	0	1	1	0

0x06

Value - Byte 1							
Bits							
8	7	6	5	4	3	2	1
0	1	0	0	0	0	1	0

"B"

Value - Byte 2							
Bits							
8	7	6	5	4	3	2	1
0	1	1	0	0	1	0	1

"e"

Value - Byte 3							
Bits							
8	7	6	5	4	3	2	1
0	1	1	1	0	0	1	0

"r"

Value - Byte 4							
Bits							
8	7	6	5	4	3	2	1
0	1	1	0	1	1	0	0

"j"

Value - Byte 5							
Bits							
8	7	6	5	4	3	2	1
0	1	1	0	1	0	0	1

"i"

Value - Byte 6							
Bits							
8	7	6	5	4	3	2	1
0	1	1	0	1	1	1	0

"n"

PLZ

Tag							
Bits							
8	7	6	5	4	3	2	1
0	0	0	0	0	0	1	0

0x02

Length							
Bits							
8	7	6	5	4	3	2	1
0	0	0	0	0	0	1	0

0x02

Value - Byte 1							
Bits							
8	7	6	5	4	3	2	1
1	1	1	0	1	0	0	0

Wert = 13405

Value - Byte 2							
Bits							
8	7	6	5	4	3	2	1
0	1	0	1	1	1	0	1

Wert = 13405

Abbildung A.30.: Beispiel für den Datentyp SEQUENCE

A.8.3.4.6. SEQUENCE OF Eine *SEQUENCE* kann aus mehreren verschiedenen Datentypen zusammengesetzt sein. Der Datentyp *SEQUENCE OF* unterscheidet sich dadurch, dass er nur Strukturen bestehend aus einem einzigen Datentyp erzeugen kann. Es können also zum Beispiel nur «*INTEGER*-» oder nur «*OCTET STRING*-Typen eingebaut werden, nicht aber beide Typen gleichzeitig.

A.8.3.4.7. Tagged Types Mit *Tagged Types* können neue Datentypen definiert werden und nicht nur, wie bei *SEQUENCE* und *SEQUENCE OF*, aus bereits bestehenden Typen zusammengesetzt werden. Dieser Datentyp wird jedoch im Rahmen dieser Diplomarbeit nicht näher betrachtet und an dieser Stelle daher nur auf Kapitel 4.4.3.4 des Buches «SNMP - Konzepte, Verfahren, Plattformen» [64] verwiesen.

A.8.4. Beschreibung der «Managed Information Base» (MIB)

Bei der *MIB* handelt es sich um ein Datenmodell, welches die *Managed Objects* beschreibt. Dieses Datenmodell wird in der Syntax *SMI* geschrieben und in einer *MIB-Datei*, wie unter «The Net-SNMP Programming Guide» [108] beschrieben, abgelegt. Der Aufbau dieser Datei wird in diesem Unterkapitel anhand eines Beispiels erläutert. Für dieses Beispiel wird die im Rahmen dieser Diplomarbeit verwendete *MIB-Datei* «STK-LAN.mib» herangezogen. Im Mikrocontrollerprogramm konnten jedoch aus Speicherplatzgründen nicht alle der *Managed Objects* implementiert werden.

Der Inhalt der «STK-LAN.mib» lautet::

```
-- SNMP-Agent on the STK-LAN
--
-- Author Date
-- =====
-- Thomas Finke 19.05.2007
--
--
STK-LAN DEFINITIONS ::= BEGIN

IMPORTS
enterprises, IpAddress, Gauge, TimeTicks FROM RFC1155-SMI
OBJECT-TYPE FROM RFC-1212
TRAP-TYPE FROM RFC-1215;
```

ANHANG A. NETZWERKGRUNDLAGEN

Fachhochschule-Heilbronn OBJECT IDENTIFIER ::= { enterprises 4766 }

STK-LAN OBJECT IDENTIFIER ::= { Fachhochschule-Heilbronn 3 }

ADC OBJECT IDENTIFIER ::= { STK-LAN 1 }

HID OBJECT IDENTIFIER ::= { STK-LAN 2 }

ADC-Value OBJECT-TYPE

SYNTAX INTEGER (0..255)

ACCESS read-only

STATUS mandatory

DESCRIPTION

"The value of the A/D conversion of the AVR's internal ADC."

::= { ADC 1 }

STK500-Switches OBJECT-TYPE

SYNTAX INTEGER (0..255)

ACCESS read-only

STATUS mandatory

DESCRIPTION

"The value of the eight switches mounted on the STK500."

::= { HID 1 }

STK500-LEDs OBJECT-TYPE

SYNTAX INTEGER (0..255)

ACCESS read-write

STATUS mandatory

DESCRIPTION

"The value of the eight LEDs mounted on the STK500."

::= { HID 2 }

STK500-LCD OBJECT-TYPE

SYNTAX OCTET STRING (SIZE(0..32))

ACCESS read-write

STATUS mandatory

DESCRIPTION

ANHANG A. NETZWERKGRUNDLAGEN

```
"The text to show on the LCD."
```

```
::= { HID 3 }
```

```
Trapswitch TRAP-TYPE
```

```
ENTERPRISE Fachhochschule-Heilbronn
```

```
VARIABLES { STK500-Switches }
```

```
DESCRIPTION "This TRAP appears if the Trapswitch is pressed."
```

```
::= 1
```

```
coldStart TRAP-TYPE
```

```
ENTERPRISE snmp
```

```
DESCRIPTION "This TRAP appears after each Reset of the microcontroller."
```

```
::= 0
```

```
END
```

Mit den doppelten Bindestrichen «--» wird in der *MIB-Datei* ein Kommentar eingeleitet. Danach wird der restliche Text in dieser Zeile bis zum Zeilenende nicht interpretiert. Mit «*STK-LAN DEFINITIONS ::= BEGIN*» wird die eigentliche Beschreibung des Datenmodells «STK-LAN» begonnen, welche bis zu «*END*» reicht. Die Angabe «*STK-LAN*» definiert hierbei den Namen der *Managed Node*.

Mit dem Schlüsselwort «*IMPORTS*» können Informationen von anderen *MIBs* eingebunden werden. Nach diesem Importierungsvorgang erfolgt der eigentliche Aufbau des *MIB-Baums*. Die Zeile «*Fachhochschule-Heilbronn OBJECT IDENTIFIER ::= { enterprises 4766 }*» erzeugt den Zweig «Fachhochschule-Heilbronn», mit der Teil-OID 4766 als Unterzweig zum Knoten «enterprises». Diesem neuen Knoten «Fachhochschule-Heilbronn» wird mit «*STK-LAN OBJECT IDENTIFIER ::= { Fachhochschule-Heilbronn 3 }*» ein weiterer Zweig namens «STK-LAN» hinzugefügt mit der Teil-OID 3. Anschließend erhält der Knoten «STK-LAN» zwei Subknoten «ADC» und «HID» (*Human Interface Device (HID)*).

Zum Schluss werden diesen Subknoten «ADC» und «HID» die *Managed Objects* des Systems zugeordnet. In diesem Fall ist dies die nur lesbare (*read-only*) Variable «ADC-Value» vom Typ *INTEGER*, welche die Werte 0 bis 255 annehmen kann und den übergeordneten Knoten «ADC» besitzt. Die zwei Variablen «STK500-Switches» und «STK500-LEDs» hingegen sind dem Knoten «HID» untergeordnet. Zusätzlich existiert in der *MIB* eine Definition für das *Managed Object* mit dem Namen «*STK500-LCD*». Durch dieses *Managed Object* kann ein *LCD* am *STK500* angesprochen werden.

ANHANG A. NETZWERKGRUNDLAGEN

Die möglichen Werte für die *ACCESS-Angabe* sind *read-only*, *read-write* und *not-accessible*. Der *STATUS* kann die Werte *mandatory*, *optional*, *deprecated* und *obsolete* annehmen.

Zusätzlich zu den vom *SNMP-Manager* abfragbaren Variablen können in der *Managed Information Base* auch *TRAP-Nachrichten*, wie in RFC1215 [109] und «Defining traps» [142] beschrieben, definiert werden. Von diesen gibt es zwei Typen: Die *Enterprise-specific-Traps* und die *Generic-Traps*. In der Beispiel-MIB wird eine Nachricht vom Typ *Enterprise-specific-Trap* mit dem Namen «*Trapswitch*» definiert, welche dem privaten Knoten «*Fachhochschule-Heilbronn*» angehört. Durch *VARIABLES* werden diejenigen *Managed Objects* angegeben, welche bei dem jeweiligen *TRAP-Ereignis* mit zum *Manager* übertragen werden. Es ist hierbei darauf zu achten, dass die Reihenfolge der Variablenauflistung mit der realen Variablenreihenfolge im *SNMP-Paket* übereinstimmt. Durch *DESCRIPTION* kann zusätzlich eine Information angegeben werden, welche später im *Manager* abgerufen werden kann. Die Zahl nach dem «*::=*» am Ende einer *Enterprise-specific-Trap* gibt den *Enterprise-Trap-Typ* an.

Laut der Definition von *TRAP-Nachrichten* sollen, wenn möglich, nur *Generic-Traps* verwendet werden. Mit den hierzu zur Verfügung stehenden Mitteln können jedoch auch *Enterprise-specific-Traps* erstellt werden.

Nach der Definition von *Trapswitch* wird zusätzlich noch eine *TRAP-Nachricht* vom Typ *Generic-Trap* genauer spezifiziert. Dies ist im Normalfall nicht nötig, da jeder *SNMP-Teilnehmer* die *Generic-Trap-Typen* kennen sollte. In der Beispiel-MIB wird die Definition dazu verwendet, dem *coldStart-Trap* eine *DESCRIPTION* hinzuzufügen. *Generic-Traps* bekommen als *ENTERPRISE* die Angabe *snmp*. Die Angabe *::= 0* am Ende der Definition gibt den für *coldStart* üblichen *Generic-Trap-Typ* null an.

Eine solche *MIB-Datei* kann mit jedem beliebigen Texteditor erstellt werden. Für dieses Beispiel wurde der *Crimson Editor* [67] verwendet.

Anschließend wird diese *MIB-Datei* mit dem Programm *MG-SOFT MIB Compiler* [24] kompiliert. Hierzu wird das Programm standardmäßig über «Start - Programme - MG-SOFT MIB Browser - MIB Compiler» gestartet. Nach dem Schließen des «Tip of the Day»-Fensters kann über das Menü «File - Compile» der Öffnen-Dialog zum Laden der *MIB-Datei* aufgerufen werden (Abbildung A.31). In diesem wird die gewünschte *MIB-Datei* ausgewählt und mit einem Klick auf «Öffnen» übernommen.

Anschließend startet die Kompilierung, welche eine Reihe von Meldungen im unteren Statusfenster anzeigt. Sobald die Meldung «Finished.» erscheint, ist die Kompilierung abgeschlossen. Nach erfolgreicher Kompilierung öffnet sich das Fenster «Compiled MIB Modules» wie in Abbildung A.32.

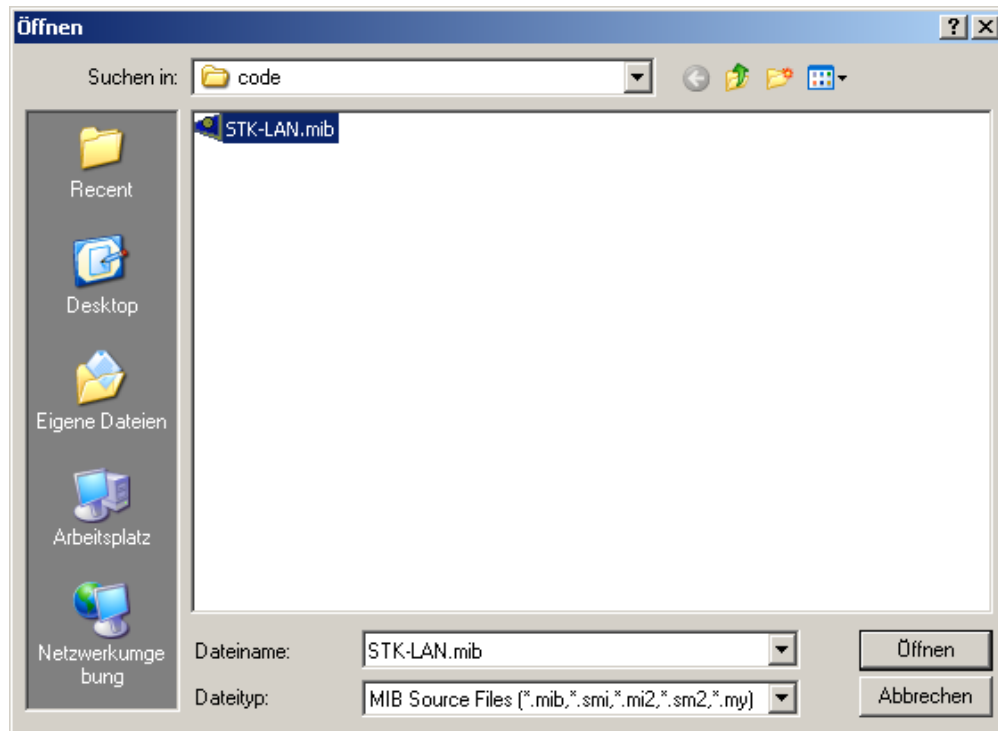


Abbildung A.31.: Öffnen einer Datei im MG-SOFT MIB-Compiler

In diesem kann das kompilierte Modul «STK-LAN» nun abgespeichert werden; der Standarddateiname für das *Database File* lautet «STK-LAN.smidb». Nachdem diese Datei abgespeichert wurde (am besten unter dem standardmäßig vorgeschlagenen Pfad), kann der *MIB Compiler* geschlossen werden.

Das nächste benötigte Programm ist der *MG-SOFT MIB Browser*, welcher sich über den gleichen Pfad im Windows-Startmenü aufrufen lässt. Nach dem Aufruf muss zuerst über die Registerkarte «MIB» das neue «Database File» eingebunden werden. Dies geschieht, indem in der unteren Liste «Module identity» der Eintrag «STK-LAN» ausgewählt und die Auswahl mit einem Klick auf den nach oben zeigenden roten Pfeil übernommen wird (Abbildung A.33).

Nach dem Importieren des Moduls kann wieder auf die Registerkarte «Query» gewechselt werden. Dort sollte nun der in Abbildung A.34 gezeigte *MIB-Tree* angezeigt werden. Wenn diese Baumstruktur wie abgebildet erscheint, kann davon ausgegangen werden, dass das Erstellen des *MIB-Datenmodells* erfolgreich war.

Auf die erstellte Variable «ADC-Value» kann nun vom *MIB-Browser* über *.1.3.6.1.4.1.4766.3.1.1.0* beziehungsweise *.iso.org.dod.internet.private.enterprises.Fachhochschule-Heilbronn.STK-LAN.ADC.ADC-Value.0* zugegriffen werden. Zu beachten ist hierbei das «.0» am Ende des Pfades. Dies

ANHANG A. NETZWERKGRUNDLAGEN

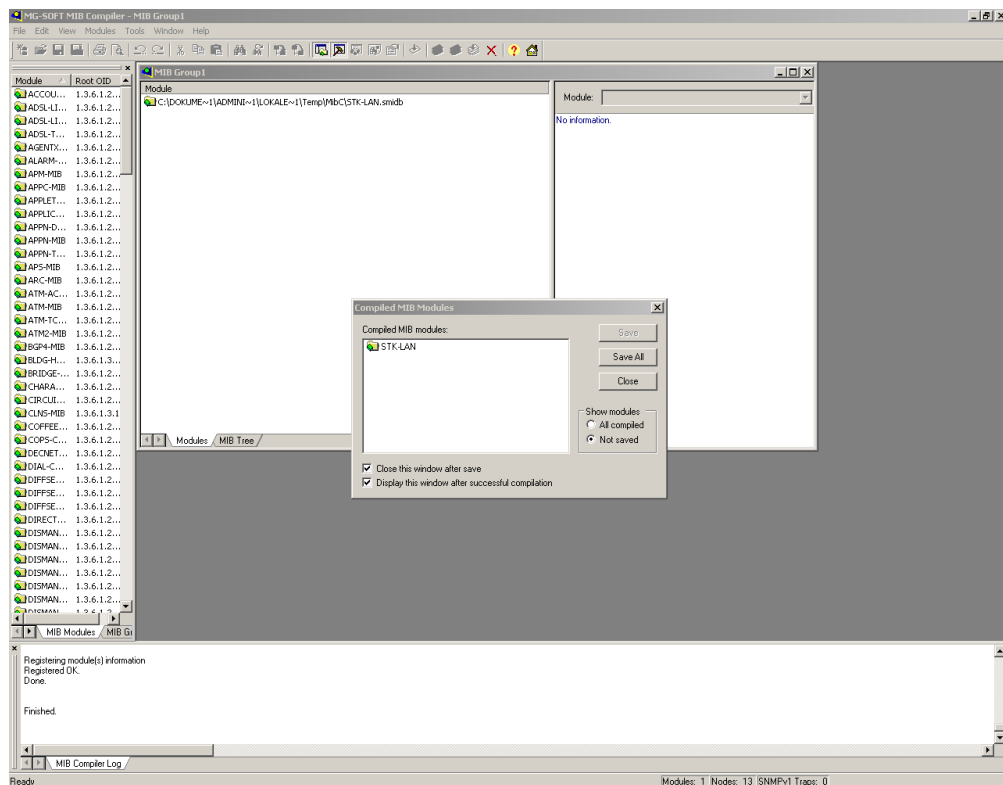


Abbildung A.32.: Die Programmoberfläche des MG-SOFT MIB-Compilers nach erfolgreichem Kompilieren

ANHANG A. NETZWERKGRUNDLAGEN

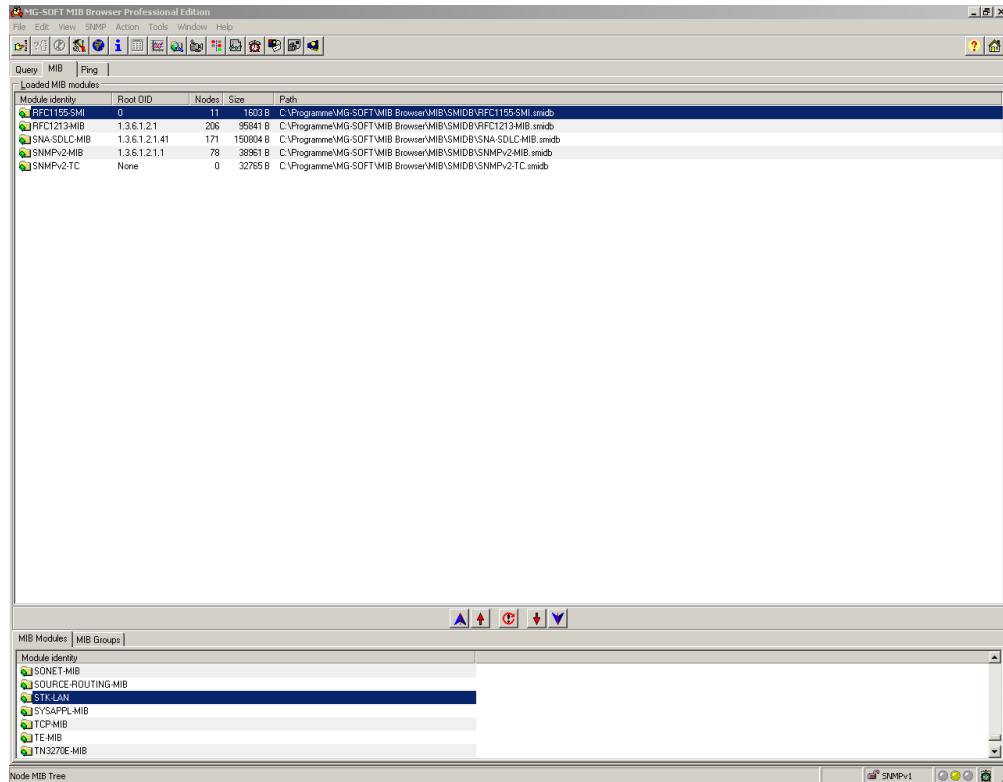


Abbildung A.33.: Importieren des MIB-Moduls

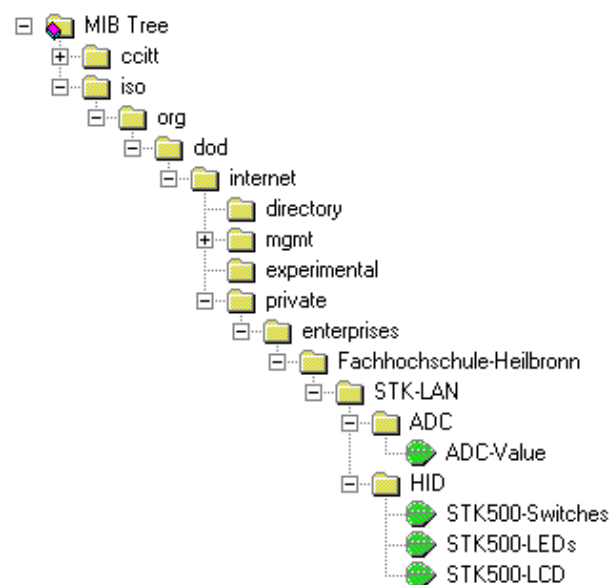


Abbildung A.34.: Ergebnis der MIB-Beschreibungsdatei

signalisiert, dass es sich um die erste Instanz des *MIB-Objekts* handelt. Ein *INTEGER-Datentyp*, wie hier verwendet, kann an dieser Stelle nur eine Null enthalten. Objekte mit mehreren Instanzen, wie zum Beispiel Tabellen, können auch höhere Werte besitzen. Die Anzahl der Instanzen eines tabellarischen *Managed Objects* ist nicht festgelegt und kann während der Laufzeit verändert werden.

A.8.5. SNMP-Nachrichtentypen

Bei einer *SNMP-Kommunikation* stehen mehrere Nachrichtentypen zum Datenaustausch zur Verfügung. Die Nachrichtentypen einer *SNMPv1-Kommunikation* sind:

A.8.5.1. GET

Mit einem *GET-Request* (PDU-Kennung = 0) kann der *Manager* ein oder mehrere *Managed Objects* vom *Agent* anfordern. Tritt dabei ein Fehler auf, wird nicht nur die fehlerhafte Variable nicht ausgeliefert, sondern die gesamte Anfrage als fehlerhaft abgewiesen. Die hierbei zur Verfügung stehenden Fehlertypen sind:

- Unbekannte OID

Ist dem *Agent* die *OID* einer Variablen nicht bekannt, so antwortet dieser mit einer der *GET-Nachricht* identischen *RESPONSE*, in welcher «Error Status» auf *noSuchName* und der «Error Index» auf die fehlerhafte Variablenbindung gesetzt ist.

- Die Antwortnachricht ist zu groß

Sobald die *Response-Nachricht* die maximal zulässige Nachrichtengröße überschreitet, wird dies durch die Fehlermeldung *tooBig* im «Error Status Feld» gemeldet. Der «Error Index» wird in diesem Fall auf null gesetzt und eine der *GET-Nachricht* identische Antwort erstellt.

- Andere Fehler

Tritt ein anderer Fehler als die oben genannten auf, antwortet der *Agent* auch mit einer der *GET-Anfrage* identischen *Response*, mit «Error Status» gleich *genErr* und einem «Error Index», welcher auf die fehlerhafte Variablenbindung zeigt.

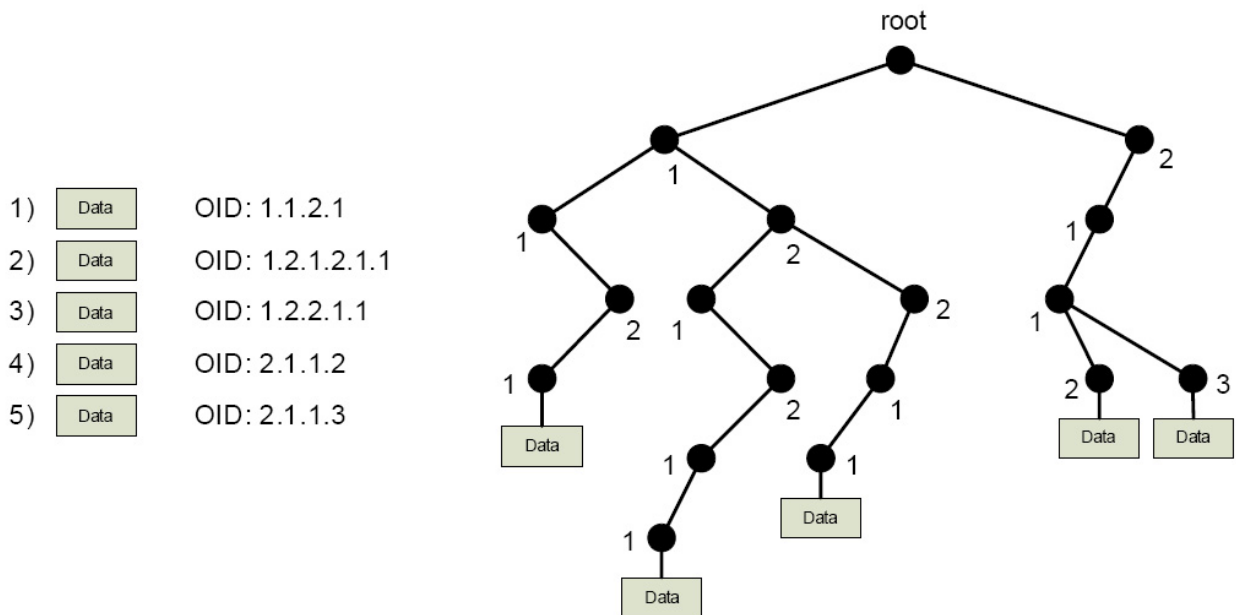


Abbildung A.35.: Beispiel einer lexikografischen Sortierung [7]

A.8.5.2. GETNEXT

Die bisher bekannte *GET-Anfrage* hat einen entscheidenden Nachteil: Sobald die exakte Struktur der *MIB* nicht bekannt ist, kann keine gezielte *GET-Anfrage* gestellt werden. Nun kann argumentiert werden, dass die *MIB* einem *SNMP-Manager* immer bekannt sein muss. Sobald der *Manager* aber beispielsweise versucht, die verschiedenen Instanzen einer Tabelle auszulesen, die sich während der Laufzeit in ihrer Anzahl ändern dürfen, oder die *MIB* dem *Manager* wirklich nicht bekannt ist, wird diese Argumentation nichtig. Für diesen Fall existiert neben der *GET-Anfrage* noch die *GETNEXT-Anfrage* (PDU-Kennung = 1).

Mit diesem Nachrichtentyp kann eine Anfrage nach einem beliebigen Knoten im *MIB-Baum* gestellt werden. Ist dieser Knoten nicht bekannt, kommt keine *noSuchName-Fehlermeldung* wie bei einem *GET-Request*, sondern der *SNMP-Agent* liefert das nächste vorhandene, lexikografisch sortierte Objekt.

Dem *MIB-Baum* aus Abbildung A.35 sind am Ende der Zweige mehrere *Managed Objects* zu entnehmen, welche nebenstehend lexikografisch sortiert aufgelistet sind.

Die Adressen der einzelnen *Managed Objects* sind hierbei aufsteigend sortiert, was sich im *MIB-Baum* so äußert, dass die Objekte von links nach rechts abgearbeitet werden.

ANHANG A. NETZWERKGRUNDLAGEN

Stellt ein *SNMP-Manager* nun beispielsweise eine Anfrage an die *OID* *.1.2*, so liefert der *Agent* das nächste vorhandene Objekt zurück, welches in diesem Beispiel *.1.2.1.2.1.1.0* (*sysDescr.0*) ist. Auf diese Weise kann ein *Manager* seine Anfrage bei einem unbekannten *MIB-Tree* an ein Objekt am Anfang des *MIB-Baums* richten und erhält daraufhin automatisch das erste vorkommende Objekt zurückgeliefert. Der *Manager* kann daraufhin seine Anfrage so modifizieren, dass er mit *GETNEXT* nach einem «tiefer» liegenden Objekt fragt, also beispielsweise nach *.1.2.1.2.1.1.1* (*sysDescr.1*). Darauf erhält der *Manager* keine Fehlermeldung, obwohl das angefragte Objekt nicht existiert. Stattdessen bekommt er das nächste vorhandene Objekt zurückgeliefert, in diesem Fall *.1.3.6.1.2.1.1.2.0* (*sysObjectID.0*). Auf diese Weise kann die *NMS* nun den kompletten *MIB-Baum* durchlaufen und somit alle *Managed Objects* abfragen.

Wird ein *GETNEXT-Request* an die exakte Adresse eines bestehenden Objekts gerichtet, also beispielsweise an *.1.3.6.1.2.1.1.1.0* (*sysDescr.0*), so wird dieses vorhandene Objekt nicht ausgeliefert, sondern das darauf folgende «nächste» Objekt. Es ist auch möglich, mehrere *GETNEXT-Requests* in einer *SNMP-Nachricht* zu übermitteln.

Bei *GETNEXT-Anfragen* können die folgend genannten Fehlerfälle auftreten:

- Es existieren keine weiteren Objekte

Wenn eine *GETNEXT-Anfrage* an das letzte vorhandene Objekt des *MIB-Baums* oder eine noch «tiefer» liegende Ressource gerichtet wird, antwortet der *Agent* mit einer der *GETNEXT-Anfrage* identischen *Response* mit geändertem Nachrichtentyp, dem «Error Status» *noSuchName* und dem «Error Index», welcher auf die problematische Variablenbindung zeigt.

- Überschreitung der maximalen Nachrichtenlänge

Wird beim Generieren der Antwort die maximale Nachrichtenlänge überschritten, antwortet der *SNMP-Agent* mit «Error Status» gleich *tooBig* und «Error Index» gleich null.

- Andere Fehler

Bei allen anderen Fehlerarten erstellt der *Agent* eine der *GETNEXT-Anfrage* identische *Response*, wieder mit geändertem Nachrichtentyp. Zusätzlich wird das Feld «Error Status» auf den Wert *genErr* gesetzt und der «Error Index» zeigt auf die Variablenbindung, welche den Fehler ausgelöst hat.

A.8.5.3. SET

Durch einen *SET-Request* (PDU-Kennung = 3) kann ein *Manager* einer oder mehreren Variablen auf dem *Agent* neue Werte zuweisen. Dabei gilt, dass entweder alle oder gar keine Variablen des *SET-Requests* geändert werden. Um dies sicher zu stellen, muss der *SNMP-Agent* vor der Änderung der Daten überprüfen, ob auch wirklich alle Variablen geändert werden dürfen beziehungsweise können. Fällt diese Überprüfung positiv aus, werden die gewünschten *Managed Objects* aktualisiert und anschließend eine dem *SET-Request* identische Nachricht mit geändertem Nachrichtentyp als *Response* zurückgesendet.

Ist während der Verarbeitung der Daten allerdings ein Fehler aufgetreten, äußert sich dies in einer der folgenden Fehlermeldungen:

- Unbekannte OID

Dem *Managed Node* ist die angefragte *OID* unbekannt. In diesem Fall wird mit einer der Anfrage identischen *Response* geantwortet, welche im «Error Status» den Wert *noSuchName* und im «Error Index» einen Zeiger auf die fehlerhafte Variablenbindung enthält.

- Falscher Datentyp oder Wertebereich

Wird bei einem *SET* ein falscher Datentyp oder ein falscher Wertebereich angegeben, so antwortet der *Agent* wie bei «Unbekannte OID», der «Error Status» steht allerdings auf *badValue*.

- Überschreitung der maximalen Nachrichtenlänge

Wird beim Generieren der *Response-Nachricht* die maximal zulässige *SNMP-Nachrichtenslänge* überschritten, wird wie bei den anderen bisher vorgestellten Fehlern eine dem *Request* identische *Response* mit geändertem Nachrichtentyp erstellt. Der «Error Status» wird auf *tooBig* und der «Error Index» auf null gesetzt.

- Keine Zugriffsrechte

Besteht auf eine oder mehrere Variablen kein Zugriffsrecht, so wird dies mit einem «Error Status» *readOnly* und einem «Error Index» entsprechend der Variablenbindung signalisiert.

- Andere Fehler

Alle anderen Fehler werden durch eine der *GET-Anfrage* identischen *Response* mit «Error Status» *genErr* angezeigt. Der «Error Index» gibt die problematische Variablenbindung an.

A.8.5.4. RESPONSE

Mit einer *RESPONSE-Nachricht* (PDU-Kennung = 2) überträgt der *Agent* die vom *Manager* angeforderten Variablenbindungen oder nötige Fehlermeldungen. Im erstgenannten Fall sind die beiden Fehlerfelder auf den Wert null gesetzt.

A.8.5.5. TRAP

Ein *SNMP-Agent* kann bei gewissen Ereignissen selbstständig, also ohne ein *Request* vom *Manager*, Nachrichten versenden. Jede dieser *TRAP-Nachrichten* (PDU-Kennung = 4) kann keine, eine oder auch mehrere Variablenbindungen enthalten.

A.8.6. Die SNMP-Versionen

Es existieren inzwischen mehrere Versionen von *SNMP*, welche sich in der Sicherheit und zusätzlichen Features unterscheiden. Die existierenden Versionen sind:

- SNMP Version 1 (SNMPv1)

SNMPv1 ist die erste Version des *Simple Network Management Protocols* und ist in RFC1157 [19] beschrieben. Der Vorteil dieser Version ist die einfache Implementierung, der gleichzeitige Nachteil dabei ist allerdings die geringe Sicherheit, da das Passwort im Klartext übertragen wird. Im Rahmen dieser Diplomarbeit wird nur *SNMPv1* verwendet, da dies für einfache Anwendungen ausreicht und einen Mikrocontroller nicht so sehr belastet wie eine neuere *SNMP-Version* mit zusätzlichen Sicherheitserweiterungen.

- Party-Based SNMP Version 2 (SNMPv2p)

SNMPv2p bietet gegenüber *SNMPv1* sichere Passwörter. Weiterhin ist durch einen sogenannten *INFORM-Request* eine Kommunikation zwischen zwei *SNMP-Managern* möglich und durch eine *GETBULK-Abfrage* können mit einem einzigen *Request* eine beliebige Anzahl von *Managed Objects* abgerufen werden. Mit Hilfe der Kommunikation zwischen zwei *SNMP-Managern* kann mit dieser Version von *SNMP* eine hierarchische Struktur des Netzmanagements realisiert werden. Der *Header* von *SNMPv2-Nachrichten* besitzt im Vergleich zu einem *SNMPv1-Header* zusätzliche Felder. Somit sind diese beiden Versionen nicht direkt zueinander kompatibel. *SNMPv2p* ist in RFC1447 [82] beschrieben und wird nur noch selten verwendet.

ANHANG A. NETZWERKGRUNDLAGEN

SNMP-Version	Erscheinungsjahr	Sicherheit	Erweiterungen
SNMPv1	1988	Community String	Basisversion
SNMPv2	1993	«Party» basierend	GETBULK, INFORM, erweiterte Fehlersignalisierung und Sicherheitskonzepte
SNMPv2u	1995	Benutzerdefiniert	Vereinfachte Konfiguration
SNMPv2c	1995	Community String	SNMPv2 über SNMPv1
SNMPv2*	1995	Benutzerdefiniert	Verbesserte Modularisierung
SNMPv3	1997	Benutzerdefiniert	Verbesserte Modularisierung, abwärtskomptibel, verbesserte Authentifizierungsverfahren und Datenverschlüsselung

Tabelle A.25.: Einige ASN.1-Kennungen der SNMP-Datentypen [7] (modifiziert)

- User-Based SNMP Version 2 (SNMPv2u)

Diese Version von *SNMP* ist in RFC1910 [135] definiert und bietet zusätzlich die Authentifizierung durch einen Benutzernamen. Auch diese Version wird nur noch sehr selten implementiert.

- Community-Based SNMP Version 2 (SNMPv2c)

Community Based SNMP verhält sich von der Sicherheit wie *SNMPv1*, bietet aber zusätzlich *GETBULK-Abfragen* und unterstützt die Kommunikation zwischen zwei *SNMP-Managern*. Aufgrund der geringeren Sicherheit und der daraus folgenden einfacheren Implementierung sowie der Möglichkeit, *GETBULK-Abfragen* und die Kommunikation zwischen zwei *Managern* zu verwenden, ist *SNMPv2c* sehr weit verbreitet. Diese Protokollversion ist in RFC1901 [20] spezifiziert. Wird der Begriff *SNMPv2* erwähnt, ist in den meisten Fällen *SNMPv2c* gemeint.

- SNMP Version 3 (SNMPv3)

Version 3 bietet auf der einen Seite eine sehr gute Sicherheit, dies resultiert aber auf der anderen Seite in einer höheren Komplexität bei der Implementierung. Aus diesem Grund ist *SNMPv3* bei kleineren Geräten noch nicht sehr weit verbreitet. Definiert ist *SNMPv3* in RFC3414 [13], weitere Informationen sind unter «Snm version 3» [116] zu finden.

Eine zusammengefasste Übersicht über die *SNMP-Versionen* ist Tabelle A.25 zu entnehmen.

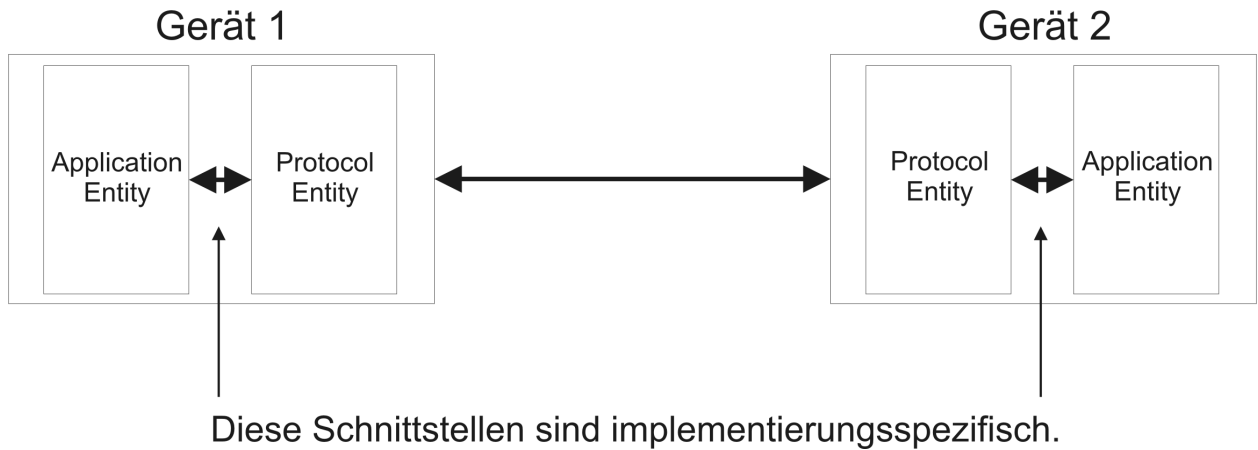


Abbildung A.36.: Kommunikation zwischen SNMP-Entities

Weiterführende Informationen zu den Protokollversionen zwei und drei sind in der Diplomarbeit «Entwurf und Realisierung eines SNMP-Agenten in einer 16-Bit μ Controller-Umgebung» [7] zu finden.

A.8.7. Das Protokoll

Bisher wurde erläutert, wie die Daten bei *SNMP* dargestellt werden. Nun müssen diese Daten aber noch in ein *SNMP-Nachrichtenpaket* verpackt werden, um gesendet werden zu können. Jedes dieser Pakete besteht aus den folgenden zwei Teilen:

1. SNMP-Nachrichten-Präambel
2. SNMP-Protokolldateneinheit (auch als *Protocol Data Unit (PDU)* bezeichnet)

All diese Daten sind nach den *BER* kodiert und werden zwischen den *SNMP Protocol Entities* ausgetauscht, welche mit ihren lokalen *SNMP Application Entities* kooperieren (Abbildung A.36). Die *Protocol Entities* müssen laut Spezifikation mindestens 484 Bytes große *SNMP-Nachrichten* empfangen können. Eine obere Grenze für die Größe der *SNMP-Nachrichten* ist jedoch nicht definiert und höchstens durch die darunterliegenden Netzwerkprotokolle beziehungsweise die *SNMP-Implementierungen* gegeben. Eine Nachricht muss immer in einem einzigen *UDP-Paket* übertragen werden können. Ein typisches *SNMPv1-Nachrichtenpaket* ist nach Abbildung A.37 aufgebaut und enthält die eigentlichen Managementinformationen in der *PDU*.

ANHANG A. NETZWERKGRUNDLAGEN

Nachrichten Länge	SNMP Nachrichten Präambel	Nachrichten Länge
SNMP Version		SNMP Version
Communitiy String		Communitiy String
PDU Typ	PDU Header	PDU Typ
PDU Länge		PDU Länge
Anfrage ID		Hersteller OID
		Agenten IP-Adresse
		Standard Trap Typ
Error Status		Hersteller Trap Typ
Error Index		Zeitstempel
Länge der Variablenbindungen	PDU Body	Länge der Variablenbindungen
Länge der ersten Bindung		Länge der ersten Bindung
OID der ersten Bindung		OID der ersten Bindung
Typ der ersten Bindung		Typ der ersten Bindung
Wert der ersten Bindung		Wert der ersten Bindung
Länge der zweiten Bindung		Länge der zweiten Bindung
OID der zweiten Bindung		OID der zweiten Bindung
Typ der zweiten Bindung		Typ der zweiten Bindung
Wert der zweiten Bindung		Wert der zweiten Bindung
....	
Weitere Variablenbindungen		Weitere Variablenbindungen
....	
GET-REQUEST GET-NEXT-REQUEST SET-REQUEST RESPONSE	Narichtentyp	TRAP

Abbildung A.37.: Aufbau eines SNMPv1-Nachrichtenpakets [7]

A.8.7.1. Aufbau der SNMP-Nachrichten-Präambel

- Nachrichten Länge

Diese Längenangabe gibt die Anzahl der nachfolgenden Bytes der *SNMP-Nachricht* an.

- SNMP Version

Mit Hilfe dieser Angabe wird die Versionsnummer der verwendeten *SNMP-Version* angegeben. Diese ist für *SNMPv1* beispielsweise null und für *SNMPv2* eins.

- Community String

In diesem Feld wird die Gruppenzugehörigkeit der *Managed Node* angegeben, also der sogenannte *Community String*.

A.8.7.2. Aufbau des PDU-Headers

Der Aufbau des *PDU-Headers* bei *TRAP-Nachrichten* unterscheidet sich von dem Aufbau der restlichen Nachrichtentypen. Daher muss zwischen diesen Headertypen unterschieden werden:

A.8.7.2.1. GET, GETNEXT, SET, RESPONSE

- PDU Typ

Mit dem *PDU Typ* wird der verwendete Nachrichtentyp angegeben, also beispielsweise *GET*, *GETNEXT*, *SET* oder *RESPONSE*.

- PDU Länge

Dieses Feld gibt an, wie viele Bytes der eigentlichen *PDU* noch folgen.

- Anfrage ID

Bei dieser *Anfrage ID* handelt es sich um eine 32-Bit-Zahl, welche die Zuordnung einer *Response* zum zugehörigen *Request* erlaubt. Somit kann der *Manager* direkt nacheinander mehrere Anfragen an einen *Agent* senden, welcher daraufhin in einer beliebigen Reihenfolge antwortet. Der *Manager* kann anschließend durch die *Anfrage IDs* wieder alle *Responses* den zugehörigen *Requests* zuordnen.

Ein *Agent* muss dieses Feld nicht verarbeiten, sondern nur die *Request ID* der Anfrage zwischenspeichern, um diese in die zugehörige *Response* wieder einfügen zu können.

ANHANG A. NETZWERKGRUNDLAGEN

- Error Status

SNMP-Nachrichten werden normalerweise mit einem nicht gesetzten *Error Status* versendet. Stellt der *SNMP-Agent* bei der Verarbeitung der Daten jedoch einen Fehler fest, kann er einen der folgenden Fehlertypen in das *Error-Status-Feld* eintragen und den Fehler somit dem *Manager* mitteilen.

- 0: «noError» - die Anfrage wurde erfolgreich verarbeitet
- 1: «tooBig» - die Antwort auf die Anfrage überschreitet die maximal zulässige SNMP-Nachrichtenlänge
- 2: «noSuchName» - das angefragte Objekt ist dem *Agent* nicht bekannt
- 3: «badValue» - der Wert zum Setzen des Objekts ist unzulässig
- 4: «readOnly» - das gewünschte Objekt darf nicht verändert werden
- 5: «genErr» - alle anderen Fehler, welche durch die vorherigen Meldungen nicht abgedeckt sind

- Error Index

Dieses Feld wird nur im Fehlerfall verwendet und hat ansonsten den Wert null. Im Fehlerfall hingegen gibt es den Index der Variablen in der Variablen-Bindungsliste an, welche den Fehler verursacht hat.

A.8.7.2.2. TRAP

- PDU Typ

Der *PDU Typ* gibt hier den Nachrichtentyp *TRAP* an.

- PDU Länge

Mit diesem Feld wird die Anzahl der nachfolgenden Bytes dieser Nachricht spezifiziert.

- Hersteller OID

Mit diesem Wert wird die *Enterprise OID* angegeben, also die herstellerabhängige *Agent-OID*. Bei einer *TRAP-Nachricht* von dem im Weiteren dieser Diplomarbeit behandelten System ist dieser Wert beispielsweise auf *.1.3.6.1.4.1.4766* gesetzt, da es sich um ein Gerät der Hochschule Heilbronn handelt.

ANHANG A. NETZWERKGRUNDLAGEN

- Agenten IP-Adresse

Anhand der *IP-Adresse* kann der *SNMP-Manager* genau erkennen, welcher *SNMP-Agent* die *TRAP-Nachricht* versendet hat.

- Standard Trap Typ

Im *SNMPv1-Protokoll* sind für *TRAP-Nachrichten* einige grundlegende Ereignisse definiert:

- 0: «coldStart» - dieser Fehlertyp signalisiert einen Kaltstart, also einen Hardware-Reset des Zielsystems. Durch einen solchen Reset können sich Variablenwerte geändert haben.
- 1: «warmStart» - mit diesem Fehler wird angezeigt, dass der *Agent* softwaremäßig reinitialisiert wurde. Auch durch diesen Vorgang können sich Variablenwerte im *SNMP-Agent* geändert haben.
- 2: «linkDown» - dieser Fehlertyp wird verwendet, wenn ein Kommunikationsinterface deaktiviert wurde.
- 3: «linkUp» - dieser Status gibt an, dass ein Kommunikationsinterface aktiviert wurde.
- 4: «authenticationFailure» - die Authentifizierung beim *Agenten* ist fehlgeschlagen.
- 5: «egpNeighborLoss» - dieser Fehler zeigt an, dass ein *EGP-Nachbar* deaktiviert wurde.
- 6: «enterpriseSpecific» - dieser Fehlerstatus signalisiert eine herstellerspezifische *TRAP-Nachricht*. Zusätzlich muss dazu dann noch der *Hersteller Trap Typ* angegeben werden.

- Hersteller Trap Typ

Sobald eine herstellerspezifische *TRAP-Nachricht* verwendet wird, zeigt dieses Feld auf den zugehörigen TRAP-Eintrag, wie dieser in der *MIB-Datei* definiert wurde. Ansonsten enthält das Feld den Wert null.

- Zeitstempel

Beim *Zeitstempel* (*Timestamp*) handelt es sich um eine 32-Bit-Variable, welche ab dem Systemstart jede 1/100 Sekunde um eins inkrementiert wird. Mit Hilfe dieser Angabe kann der *SNMP-Manager* die Reihenfolge der *TRAP-Nachrichten* ermitteln und dadurch erkennen, in welcher Reihenfolge die *TRAP-Ereignisse* aufgetreten sind.

A.8.7.3. Der PDU-Body

Der *PDU-Body* enthält die eigentlichen Nutzdaten einer *SNMP-Nachricht*, welche in verschiedenen Variablenbindungen transportiert werden. Jede Variablenbindung repräsentiert hierbei ein *Managed Object* und jede *SNMP-Nachricht* muss ein oder mehrere dieser Variablenbindungen enthalten. Die *TRAP-Nachricht* bildet hierbei eine Ausnahme, da diese auch keine einzige Variablenbindung enthalten darf. Die Felder des *PDU-Bodies* sind wie folgt definiert:

- Länge der Variablenbindungen

Dieses Feld gibt die Länge aller Variablenbindungen in Bytes an.

- Länge der Variablenbindung

Durch dieses Feld wird die Länge der aktuellen Variablenbindung spezifiziert.

- OID der Variablenbindung

In diesem Feld wird die *OID* der Variablenbindung angegeben und somit die verwendete Variable adressiert.

- Typ der Variablenbindung

Mit diesem Wert wird der Datentyp der aktuellen Variable definiert.

- Wert der Variablenbindung

Der *Wert der Variablenbindung* gibt schließlich den eigentlichen Wert der Variablen an.

A.8.8. Analyse einer SNMP-Nachricht

Nachdem nun alle Eigenschaften, Datentypen und Kodierungen eines *SNMP-Pakets* vorgestellt wurden, können diese Informationen am Beispiel in Abbildung A.38 nachvollzogen werden. Bei dem *SNMP-Paket* handelt es sich um eine *SNMPv1-Response-Nachricht*.

ANHANG A. NETZWERKGRUNDLAGEN

Sequenz	0x30	→	- Anfang einer Sequenz
Länge	0x82	→	- Die Länge wird hier in 3-Bytes, in der „Long Form“ kodiert
Länge	0x00	→	0x4D ist die Länge und entspricht 77 Nachfolgenden Bytes
Länge	0x4D	→	
Typ	0x02	→	- Hier wird die SNMP-Protokoll Versionsnummer kodiert
Länge	0x01	→	0x02 steht für ein Integerwert, 0x01 gibt die Anzahl der
Wert	0x00	→	Nachfolgebytes wieder und der Wert 0 steht für SNMPv1
Typ	0x04	→	- Hier wird der Community-String kodiert
Länge	0x06	→	Der String ist von Datentyp „OCTET STRING“, ist 6-Bytes lang.
Wert	0x70	→	Der Community-String lautet: public
Wert	0x75	→	
Wert	0x62	→	
Wert	0x6C	→	
Wert	0x69	→	
Wert	0x63	→	
PDU Typ	0xA2	→	- Der Nachrichtentyp: response
Länge	0x82	→	- Die Länge in der „Long Form“ kodiert
Länge	0x00	→	0x3E ist die Länge und entspricht 62 Nachfolgenden Bytes
Länge	0x3E	→	
Type	0x02	→	- Hier wird die „Request ID“, die vom Typ „INTEGER“ ist in 4-Bytes
Länge	0x04	→	kodiert. Die Beziehungszahl ist: 50
Wert	0x00	→	
Wert	0x00	→	
Wert	0x00	→	
Wert	0x32	→	
Typ	0x02	→	- Error Status ist 0, keine Fehler
Länge	0x01	→	
Wert	0x00	→	
Typ	0x02	→	- Error Index ist 0, keine Fehler
Länge	0x01	→	
Wert	0x00	→	
Sequenz	0x30	→	- Anfang einer Sequenz
Länge	0x82	→	- Die Länge in der „Long Form“ kodiert
Länge	0x00	→	0x2E ist die Länge und entspricht 46 Nachfolgenden Bytes
Länge	0x2E	→	
Sequenz	0x30	→	- Anfang einer Sequenz
Länge	0x82	→	- Die Länge in der „Long Form“ kodiert
Länge	0x00	→	0x2A ist die Länge der Variablenbindung und entspricht 42-Bytes
Länge	0x2A	→	
Typ	0x06	→	- Hier steht die OID der Variablenbindung .
Länge	0x0B	→	1.3.6.1.4.1.14905.11.1.1.0
Wert	0x2B	→	
Wert	0x06	→	
Wert	0x01	→	
Wert	0x04	→	
Wert	0x01	→	
Wert	0xF4	→	
Wert	0x39	→	
Wert	0x0B	→	
Wert	0x01	→	
Wert	0x01	→	
Wert	0x00	→	
Typ	0x04	→	- Hier steht der Wert der Variablenbindung ,
Länge	0x1B	→	es ist ein „OCTET STRING“:
Wert	0x44	→	„Dies ist ein 10 Byte String“
...	...	→	
...	...	→	
...	...	→	
Wert	0x67	→	

Abbildung A.38.: Nachrichtenpaket einer SNMPv1-Response [7]

B. Marktübersicht

Zum Anbinden eines Systems an *Ethernet* sind auf dem Markt zahlreiche Produkte erhältlich. Einige dieser Produkte sind:

- Beck-IPC

Die Firma *Beck IPC GmbH* [43] bietet kleine Module der IPC@CHIP®-Familie an, welche einen integrierten Webserver besitzen. Zusätzlich besitzen diese Module zahlreiche Schnittstellen wie zum Beispiel *Inter-Integrated Circuit (I2C)*, *SPI* oder *Controller Area Network (CAN)*. Mit Hilfe dieser Module können Geräte auf einfachste Weise an ein Netzwerk angebunden werden. Eine Übersicht über die verfügbaren Module liefert Tabelle B.1.

Abbildung B.1 zeigt ein solches *Beck-IPC* System in einem DIL32-Gehäuse (*Dual In-Line (DIL)*).

- easyToWeb

EasyToWeb [128] bietet ab 149,00 € kleine Platinen an, welche mit einem *AVR*- oder *ARM-Mikrocontroller (Advanced RISC Machines (ARM))* und der zusätzlichen Peripherie bestückt sind, um einen kleinen *Embedded Webserver* zu realisieren. Diese Platinen eignen sich vor allem für den Einstieg in eine Ethernetimplementierung, da die Schaltung bei Bedarf modifiziert



Abbildung B.1.: Beck-IPC Modul im DIL32-Gehäuse

ANHANG B. MARKTÜBERSICHT

	SC1x	SC1x3
Prozessor	SC186 20/40 MHz	SC186-EX 96 MHz
RAM	512 KB	8 MB
Flash	512 KB	2 MB (SC123) / 8 MB (SC143)
Anschlüsse	Ethernet, 2xSeriell, I ² C, SPI, 14 I/O, IRQ, Externer DMA, Timer Ein-/Ausgang	2xEthernet, 4xSeriell, 2xCAN 2.0B, USB1.1, I2C, SPI, 31 I/O, IRQ, Externer DMA, Timer Ein-/Ausgang,
Produkte	SC12, SC13, SC13-IEC, SC11, SC11-IEC	SC123, SC123-IEC, SC143, SC143-IEC
Preis	59,63 € - 96,00 €	39,00 € - 129,00 €

Tabelle B.1.: Übersicht über die Beck-IPC Module

werden kann und der komplette *TCP/IP-Stack* frei zur Verfügung steht (der C-Code kann auf der Website heruntergeladen werden).

- *Embedded Computer Module* von Wilke Technology GmbH

Die Firma *Wilke Technology GmbH* [45] bietet kleine *Embedded Webserver* der Serien *ECO-NO-*, *TINY-* und *BASIC-Tiger* an. Die Module sind ab einem Preis von 49 € erhältlich und enthalten unter anderem mehrere digitale I/O-Pins, serielle Schnittstellen sowie mehrere Analog-Digital-Wandler. Diese Module können im Zielsystem programmiert werden.

C. Entwicklungsumgebung

C.1. Hardwareentwicklungstools

Zur Entwicklung und zum Testen der Hardware stehen einige Tools zur Verfügung, welche im Folgenden vorgestellt werden.

C.1.1. CadSoft EAGLE

Bei dem Programm *Einfach Anzuwendender Grafischer Layout Editor (EAGLE)* der *CadSoft Computer GmbH* [44] handelt es sich um eine Kombination aus Schaltplan- und Layouteditor. Mit Hilfe dieses Programms sind der Schaltplan und das Layout des *STK-LAN*, welches in Kapitel 2 näher beschrieben ist, erstellt.

Auf der Website der *CadSoft Computer GmbH* kann eine eingeschränkte Version des Programms, namens *EAGLE Light Edition* kostenlos heruntergeladen werden. Die Einschränkungen dieser Version sind:

- Die nutzbare Platinenfläche ist auf 100 x 80 mm begrenzt.
- Es können nur zwei Layer (Top und Bottom) verwendet werden.
- Der Schaltplan-Editor kann nur eine Seite erzeugen.

Da der im Rahmen dieser Diplomarbeit entwickelte Schaltplan jedoch mehr als eine Seite besitzt und die Platine größer als 100 mm x 80 mm ist, wurde für diese Diplomarbeit eine Vollversion von *EAGLE* verwendet. Die damit erstellten Projektfiles lassen sich jedoch auch mit der *Light Edition* öffnen.

ANHANG C. ENTWICKLUNGSUMGEBUNG

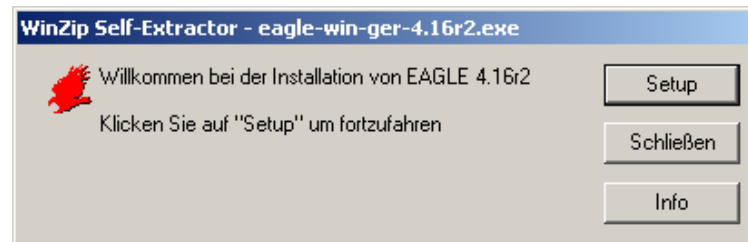


Abbildung C.1.: Startfenster der EAGLE-Installation



Abbildung C.2.: Willkommensbildschirm beim Start der Installation

C.1.1.1. Installation

Bevor das Programm genutzt werden kann, muss es installiert werden. Hierzu wird die *EAGLE Light Edition* von der Website der *CadSoft Computer GmbH* heruntergeladen. Die derzeit aktuelle Version ist die *4.16r2*. Die Installation kann über die heruntergeladene Datei *eagle-win-ger-4.16r2.exe* gestartet werden.

Nach dem Ausführen der Datei erscheint das in Abbildung C.1 gezeigte Fenster, in welchem die eigentliche Installation durch einen Klick auf «Setup» gestartet werden muss. Nach dem Start der Installation erscheint der Willkommensbildschirm (Abbildung C.2), in welchem die Installation durch einen Klick auf die Schaltfläche «Weiter» fortgesetzt wird.

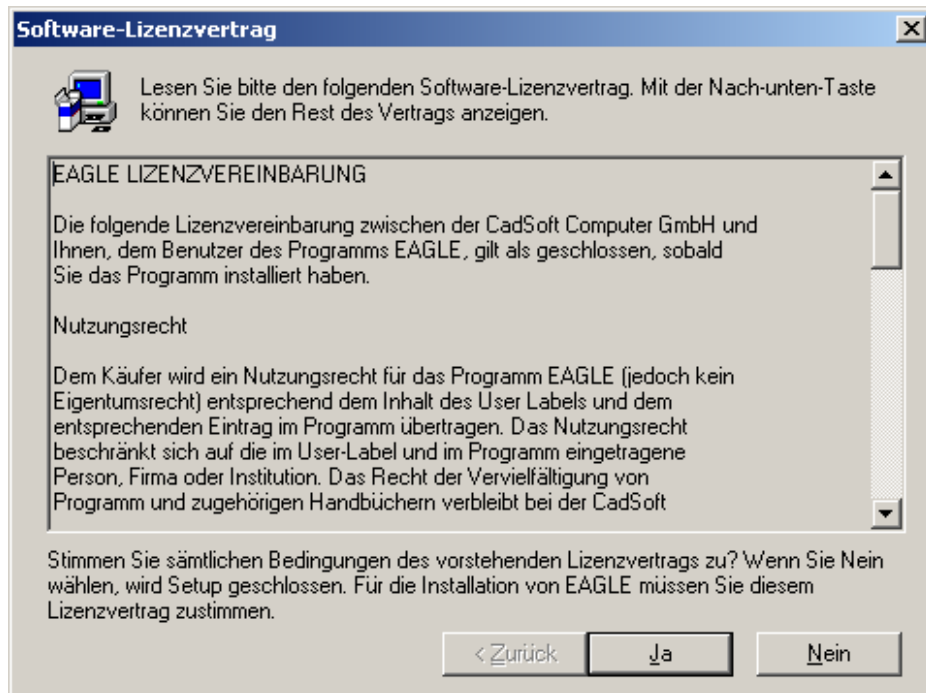


Abbildung C.3.: Zustimmung des Lizenzvertrags

Anschließend muss die Lizenzvereinbarung (Abbildung C.3) durch einen Klick auf «Ja» akzeptiert werden. Im darauf folgenden Fenster (Abbildung C.4) fragt die Installationsroutine nach dem gewünschten Installationspfad. Nachdem dieser eingegeben wurde, kann er mit einem Klick auf «Weiter» übernommen werden.

Nachdem die Installationsroutine alle benötigten Informationen zur Installation gesammelt hat, wird dies dem Benutzer durch das Fenster in Abbildung C.5 mitgeteilt. Das eigentliche Kopieren der Dateien beginnt nach einem Klick auf «Weiter». Im folgenden Schritt wird *EAGLE* auf dem Rechner installiert. Der Installationsfortschritt wird wie in Abbildung C.6 dargestellt.

Nach dem erfolgreichen Kopieren aller Dateien erscheint das Fenster aus Abbildung C.7 und meldet dem Benutzer den Erfolg der Installation. Durch einen Klick auf die Schaltfläche «Weiter» erscheint das Fenster aus Abbildung C.8, welches das Ende der Installation anzeigt.

Durch einen Klick auf «Beenden» wird die Installationsroutine abgeschlossen.

ANHANG C. ENTWICKLUNGSUMGEBUNG

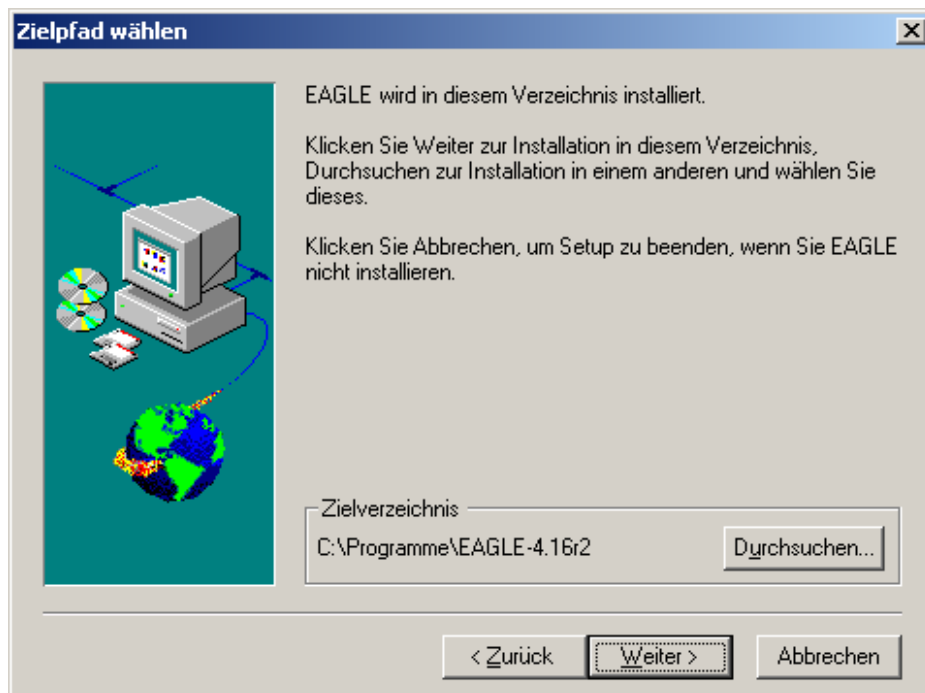


Abbildung C.4.: Auswahl des Installationspfads

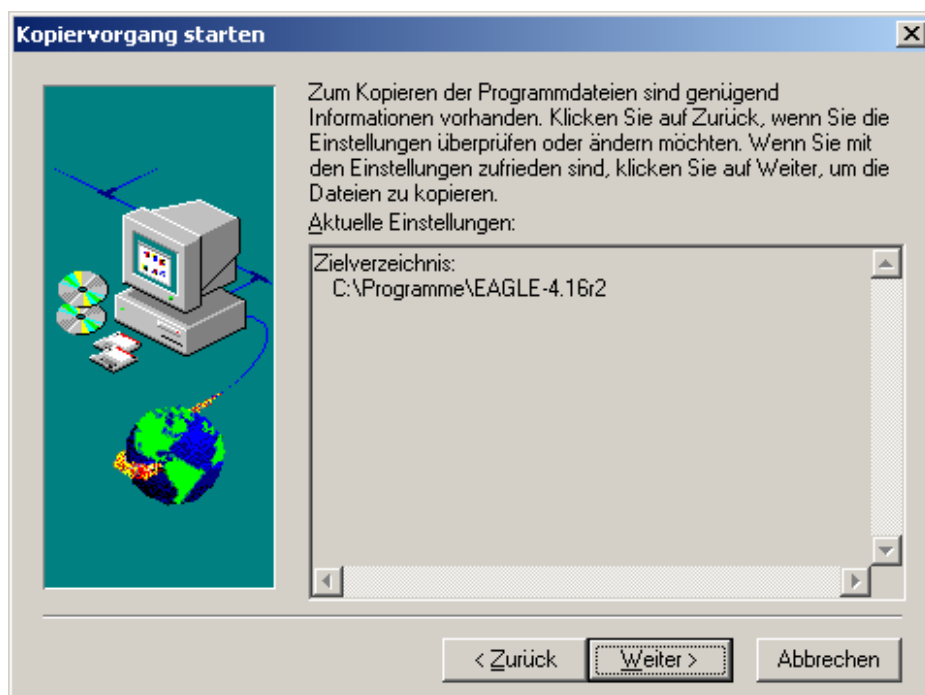


Abbildung C.5.: Beginn der eigentlichen Installation

ANHANG C. ENTWICKLUNGSUMGEBUNG

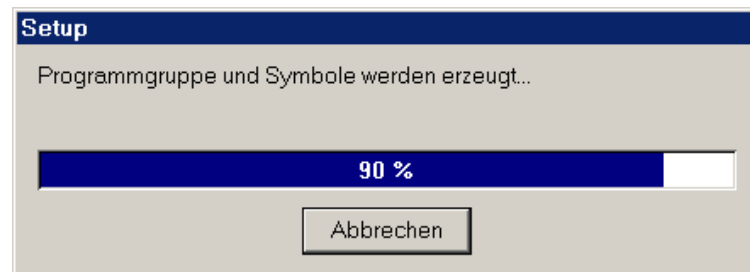


Abbildung C.6.: Anzeige des Installationsfortschritts

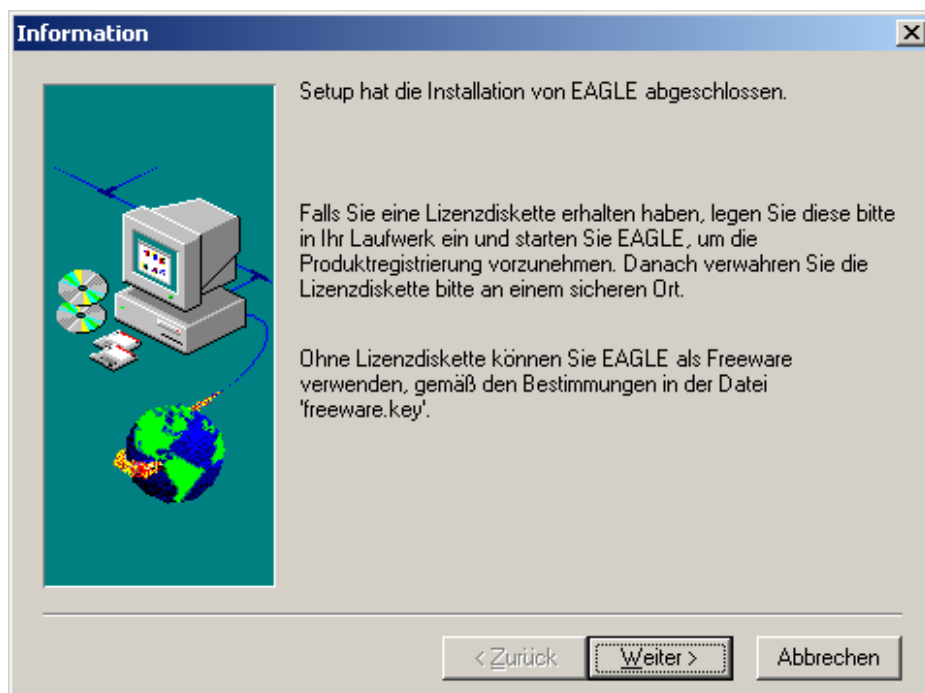


Abbildung C.7.: Erfolgreiche Installation

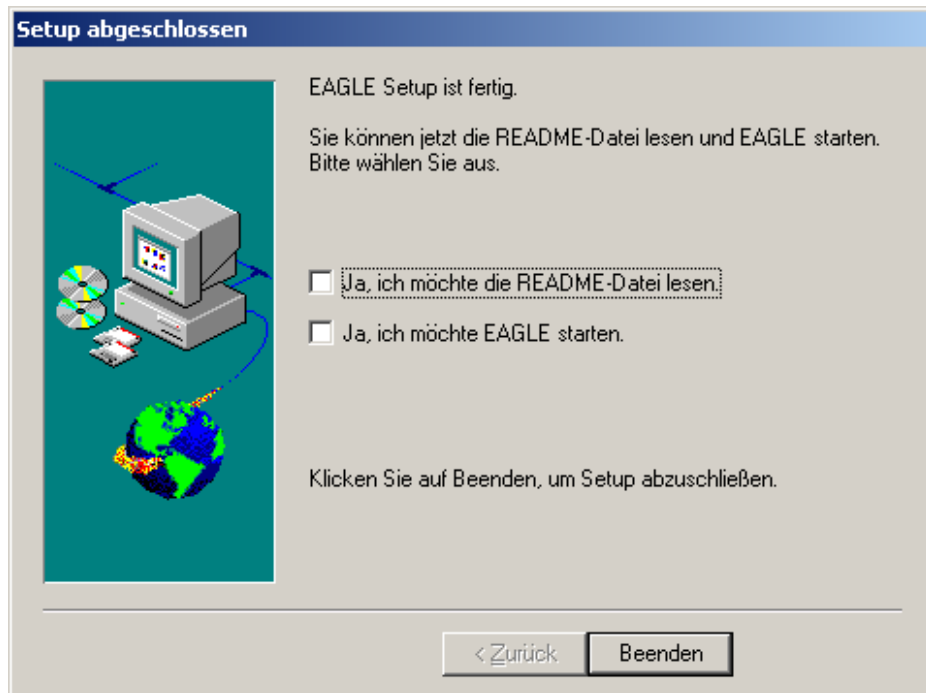


Abbildung C.8.: Abschließen der Installation

C.1.1.2. Programmoberfläche

Bevor nun die grundsätzliche Bedienung des Programms erläutert wird, sollte das *EAGLE-Projekt* «STK_LAN» in den *EAGLE-Projektordner* kopiert werden. Hierzu muss der komplette Ordner «STK_LAN» aus dem Verzeichnis «EAGLE-Projekt» der beiliegenden Diplomarbeits-CD-ROM in den Ordner «C:\Programme\EAGLE-4.16r2\projects» kopiert werden.

Anschließend wird das Programm über «Start - Programme - EAGLE Layout Editor 4.16r2 - EAGLE 4.16r2» aufgerufen. Beim ersten Programmstart erscheint die Meldung aus Abbildung C.9, welche nach der gewünschten Lizenz fragt. Mit einem Klick auf «Als Freeware lizenzieren» kann das Programm in der kostenlosen *Light Edition* genutzt werden. Nachdem *EAGLE* vollständig gestartet ist, wird das sogenannte *Control Panel* (Abbildung C.10) angezeigt.

An dieser Stelle wird ausdrücklich darauf hingewiesen, dass es bei *EAGLE* sehr wichtig ist, dass *Schaltplan* und *Board* immer gemeinsam geöffnet sind. Wird darauf nicht geachtet und nur eine der Dateien geöffnet und bearbeitet, so resultiert dies in einer Inkonsistenz zwischen *Schaltplan* und *Board*. Es ist dann nicht mehr gewährleistet, dass die beiden Dateien zueinander passen.

ANHANG C. ENTWICKLUNGSUMGEBUNG

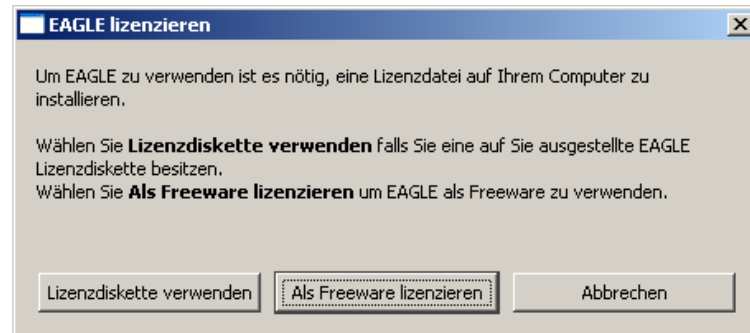


Abbildung C.9.: Auswahl der gewünschten Lizenz

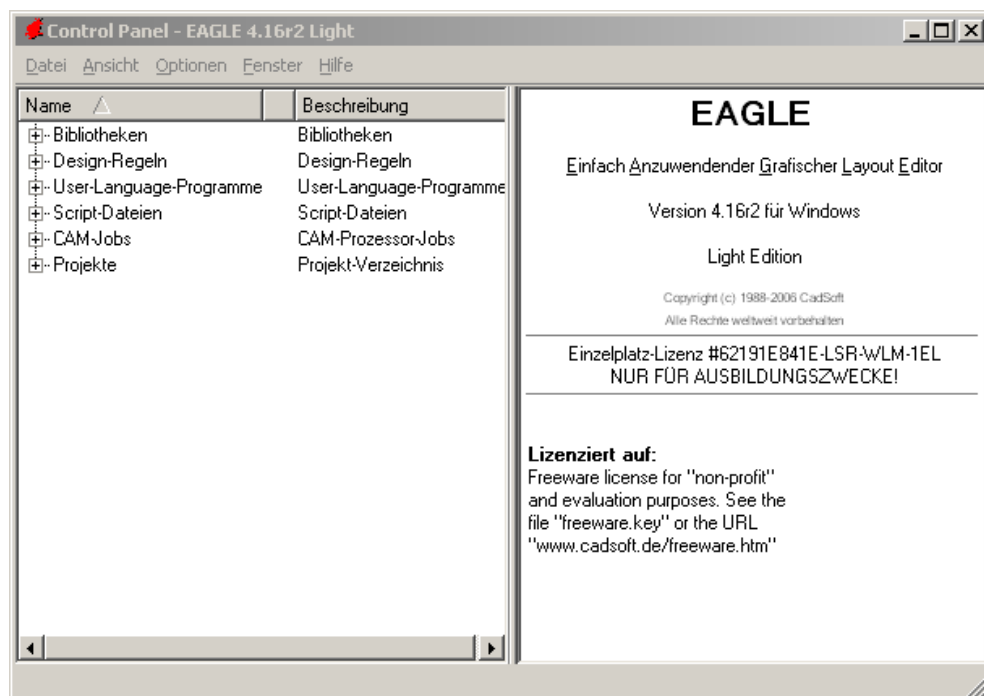


Abbildung C.10.: Oberfläche des Control Panels

ANHANG C. ENTWICKLUNGSUMGEBUNG

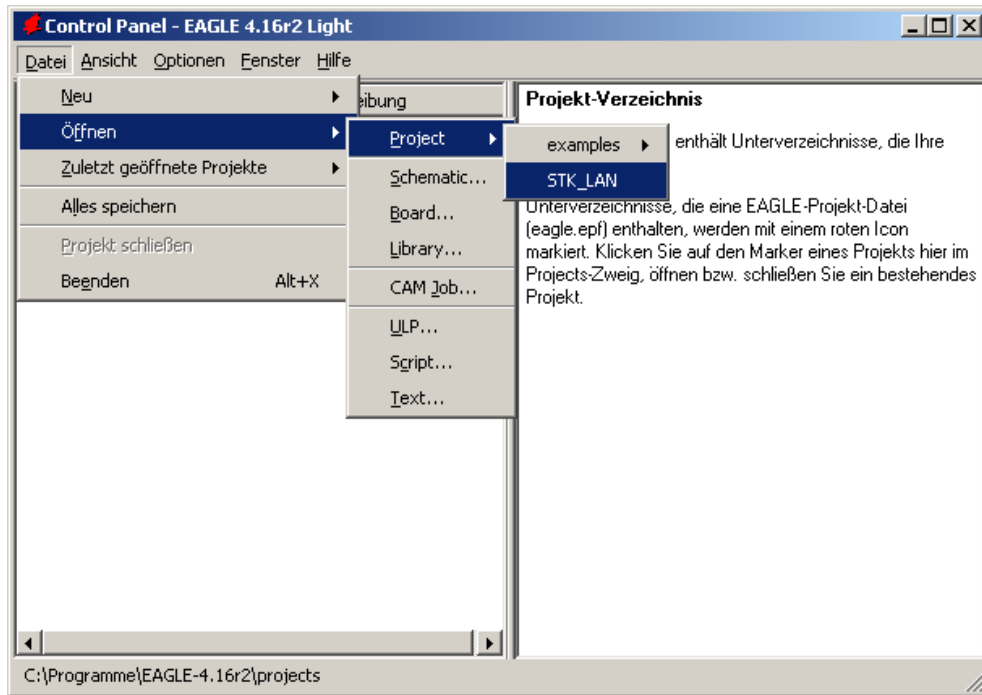


Abbildung C.11.: Öffnen eines bestehenden Projekts

Um die Konsistenz jederzeit gewährleisten zu können, wird empfohlen die *Schaltplan*- oder *Board-Datei* nicht einzeln über den Menüeintrag «Datei - Öffnen - Schematic», beziehungsweise «Datei - Öffnen - Board», im *Control Panel* zu öffnen. Stattdessen sollte über den Menüeintrag «Datei - Öffnen - Project - STK_LAN» (Abbildung C.11) immer das komplette Projekt geöffnet werden, bei welchem der *Schaltplan* und das *Board* immer gemeinsam geladen werden.

Nach dem Öffnen des Projekts erscheinen die zwei neuen Fenster mit den Titeln «Schaltplan» und «Board». In diesen beiden Fenstern können nun der Schaltplan und das Layout bearbeitet werden. Die Beschreibung der kompletten *EAGLE-Oberfläche* würde an dieser Stelle den Rahmen sprengen und daher wird nur auf das *EAGLE-Handbuch* [17] verwiesen.

Nachdem die Arbeit mit *EAGLE* beendet ist, werden im *Control Panel* über das Menü «Datei» und den Eintrag «Projekt schließen» der *Schaltplan* und das *Board* wieder geschlossen.

Eine Anleitung zur Erstellung der Gerberdaten für die Platinenfertigung ist in Anhang D zu finden.

C.1.2. STK500

Das *STK500* (Abbildung C.12) ist das von *Atmel* angebotene Starterkit für die Mikrocontroller der *AVR-Familie*. Das Board bietet unter anderem zwei *RS232-Schnittstellen*. Eine davon dient zur Steuerung des *STK500*, während die andere für eigene Anwendungen verfügbar ist. Zusätzlich bietet die Experimentierplatine Programmieranschlüsse für *ISP*, *JTAG*¹ sowie serielle und parallele *High-Voltage-Programmierung*. Zur Interaktion mit dem Anwender sind acht *Taster* und acht *LEDs* vorhanden, welche beliebig mit den Portpins des Mikrocontrollers verbunden werden können. Die Versorgungsspannung sowie das Taktsignal des Zielsystems werden vom *STK500* zur Verfügung gestellt und können über das *Atmel AVR Studio* vom *PC* aus eingestellt werden. Das Experimentierboard beinhaltet außerdem für eine große Anzahl der *AVR-Mikrocontroller* einen passenden IC-Sockel (*Integrated Circuit (IC)*). Für diejenigen Mikrocontroller, welche nicht direkt aufgesteckt werden können, sind Erweiterungsplatinen erhältlich.

Zur Konfiguration sind auf dem *STK500* einige Stiftleisten vorhanden, welche mit Hilfe von *Jumpern* verbunden werden können. Die Jumperkonfiguration, welche während dieser Diplomarbeit zur Verwendung des *STK-LAN* benutzt wurde, ist auf dem *STK-LAN* aufgedruckt beziehungsweise nachfolgend aufgelistet:

- VTARGET: connected
- AREF: connected
- RESET: disconnected
- XTAL1: connected
- OSCEL: 1-2
- BSEL2: disconnected
- PJUMP: disconnected

Für eine detaillierte Ausführung der Jumperkonfigurationen wird an dieser Stelle auf den *STK500 User Guide* [8] der *Atmel Corporation* verwiesen.

¹Für das Programmieren und Debuggen über *JTAG* wird zusätzlich zum *STK500* ein *JTAG-Programmieradapter* benötigt.

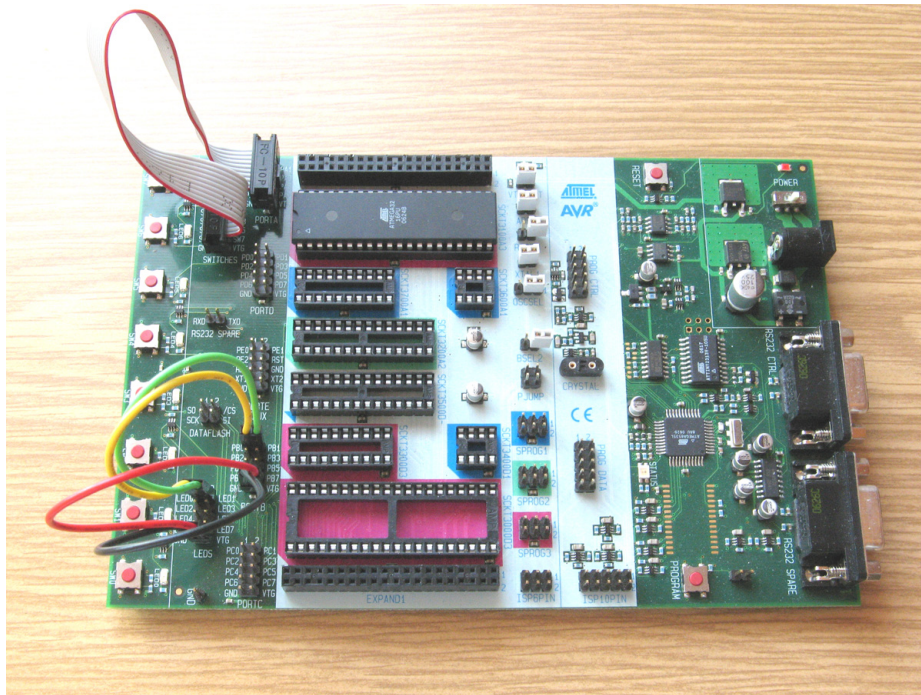


Abbildung C.12.: Das im Mikrocomputerlabor verwendete Entwicklungsboard STK500

C.1.3. AVR-USB-JTAG

Während dieser Diplomarbeit wurde zum Programmieren und Debuggen der JTAG-Programmiersadapter *AVR-USB-JTAG* (Abbildung C.13) der Firma *OLIMEX Ltd.* [80] verwendet. Dieser Programmiersadapter ist für alle *AVR-Mikrocontroller* mit *JTAG-Interface* und maximal 32 Kilobytes *Flash* geeignet. All seine Datenleitungen sind optoelektronisch isoliert. Die Spannungsversorgung erhält das Programmiergerät über den *USB-Anschluss*. Die Versorgungsspannung des Zielsystems darf im Bereich von 3,0 bis 5,0 Volt liegen. Der *AVR-USB-JTAG* ist voll kompatibel zum *Atmel AVR Studio* und wird von diesem als *ATJTAGICE* erkannt. Dadurch kann die Firmware des Programmieradapters direkt über das *AVR Studio* aktualisiert werden.

Der Programmiersadapter kann bei der «Elektronikladen Mikrocomputer Giesler & Danne GmbH & Co.KG» [71] bezogen werden. Das Betriebssystem des *PCs* spricht den Programmiersadapter über eine virtuelle serielle Schnittstelle an. Hierzu wird der Treiber für den Baustein *FT232* der Firma *Future Technology Devices International Ltd (FTDI)* [78] benötigt, welcher direkt im Downloadbereich [79] heruntergeladen werden kann.



Abbildung C.13.: Nachbau des Programmieradapters ATJTAGICE mit der Bezeichnung AVR-JTAG-USB [71]

C.1.4. AVRATJTAGICE mkII

Beim *AVRATJTAGICE mkII* (Abbildung C.14) handelt es sich um den *JTAG-Programmieradapter* der *Atmel Corporation*. Dieser Programmieradapter unterstützt *ISP*, *JTAG*² und *debugWire*. Das *AVRATJTAGICE mkII* wird über eine *USB 1.1-* oder *RS232-Schnittstelle* mit dem PC verbunden und vom *AVR Studio* vollständig unterstützt. Die Versorgungsspannung des Zielsystems darf im Bereich von 1,8 bis 5,5 Volt liegen und das Programmiergerät kann mit einer 9 - 12 Volt Versorgung betrieben werden. Alternativ kann das *AVRATJTAGICE mkII* auch über die *USB-Schnittstelle* mit Spannung versorgt werden.

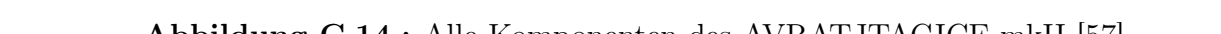
Das *AVRATJTAGICE mkII* ist momentan das einzige auf dem Markt befindliche Gerät, welches das Programmieren und Debuggen von allen³ *AVR-Mikrocontrollern* unterstützt.

Während der Verwendung des *AVRATJTAGICE mkII* kommt es teilweise vor, dass das *AVR Studio* keine Kommunikation mit dem *AVRATJTAGICE mkII* herstellen kann. Dies scheint ein Fehler des Programmieradapters zu sein. Es war während dieser Diplomarbeit leider nicht möglich, diesen Fehler näher einzugrenzen. Es scheint jedoch so, als ob es ein Problem einer neueren *Firmwareversion* ist, da das Problem bei einem Programmieradapter mit nicht aktualisierter *Firmware* nicht auftrat.

Bei der Inbetriebnahme sollte außerdem darauf geachtet werden, dass das *JTAGICE mkII* vor dem Zielsystem eingeschaltet wird. Dies ist darin begründet, dass die Datenleitungen des Programmierge-

²Bei der Verwendung von *JTAG* werden auch mehrere *Devices* in einem *JTAG Scan Chain* unterstützt.

³Also auch den Mikrocontrollern mit mehr als 32 Kilobyte Flash.



Prichard, H. (1991). IX: AND-Distribution and the ϕ -function. *Journal of Philosophy*, 88, 41-64.

ANHANG C. ENTWICKLUNGSUMGEBUNG

Tools. Bei anderen Kombinationen stürzt das *Atmel AVR Studio* beim Start mit einer Fehlermeldung ab.

- Kombination 1

Es wird Version *20060421* oder älter von *WinAVR* in Kombination mit *Atmel AVR Studio 4.12 (build 460)* und optional dem *Update AVR Studio 4.12 Service Pack 4 (build 498)* verwendet.

- Kombination 2

Es wird eine 2007er Version von *WinAVR*, wie beispielsweise die *20070525rc2* vom 15.05.2007, in Kombination mit *Atmel AVR Studio 4.13 b528* oder neuer verwendet.

Im Rahmen dieser Diplomarbeit wird die erstgenannte Kombination verwendet, da zwar die neue *WinAVR-Distribution* fehlerfrei läuft, die neue Version des *Atmel AVR Studios* aber noch einige Fehler enthält. Zwei der gravierendsten Fehler sind, dass bei manchen Installationen gesetzte *Breakpoints* im Quellcodefenster nicht korrekt angezeigt werden und dass der *Cycle Counter* nicht funktioniert.

Nach dem Starten der von der Website «Winavr: Avr-gcc for Windows» [139] heruntergeladenen «WinAVR-20060421-install.exe» erfolgt die Installation in sieben Schritten:

1. Auswahl der Sprache

Der Dialog in Abbildung C.15 erscheint direkt nach dem Starten der Installationsroutine und fordert zur Auswahl der gewünschten Sprache auf. Nach Auswahl der Sprache wird die Installation mit einem Klick auf «OK» fortgesetzt.

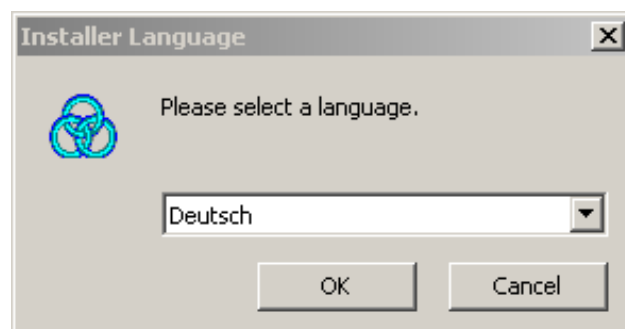


Abbildung C.15.: Auswahl der gewünschten Sprache



Abbildung C.16.: Willkommens-Bildschirm beim Start der Installation

2. Willkommens-Bildschirm

Bei dem Willkommens-Bildschirm (Abbildung C.16) wird die Installation mit einem Klick auf «Weiter» fortgesetzt.

3. Lizenzabkommen

Die Lizenz muss zum Fortsetzen der Installation in Abbildung C.17 mit einem Klick auf «Annehmen» akzeptiert werden.

4. Auswahl des Zielverzeichnisses

In diesem Teil der Installation (Abbildung C.18) kann ausgewählt werden, wohin *WinAVR* installiert werden soll. Der Standardpfad ist «C:\WinAVR» und wurde in der Diplomarbeit auf «C:\Programme\WinAVR» abgeändert. Mit einem Klick auf die Schaltfläche «Weiter» wird die Auswahl übernommen.

5. Auswahl der Komponenten

In diesem Fenster können die zu installierenden Komponenten der *WinAVR-Distribution* bestimmt werden. Die drei Haken wie in Abbildung C.19 sollten gesetzt bleiben, damit das installierte Programm ohne Probleme funktioniert. Mit einem anschließenden Klick auf «Installieren» wird der eigentliche Installationsvorgang (Abbildung C.20) gestartet.

ANHANG C. ENTWICKLUNGSUMGEBUNG

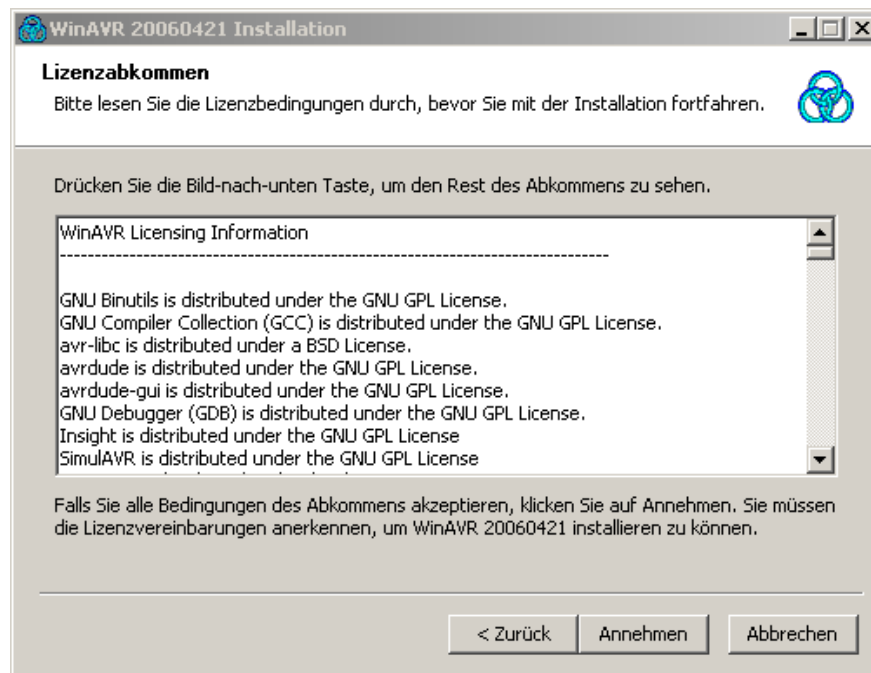


Abbildung C.17.: Annehmen der Lizenzvereinbarung

6. Installation abgeschlossen

Nach einem fehlerfreien Installationsvorgang wird mit Abbildung C.21 das Ende der Installation angezeigt. Nach einem Klick auf die Schaltfläche «Fertig stellen» wird die Installation beendet.

ANHANG C. ENTWICKLUNGSUMGEBUNG

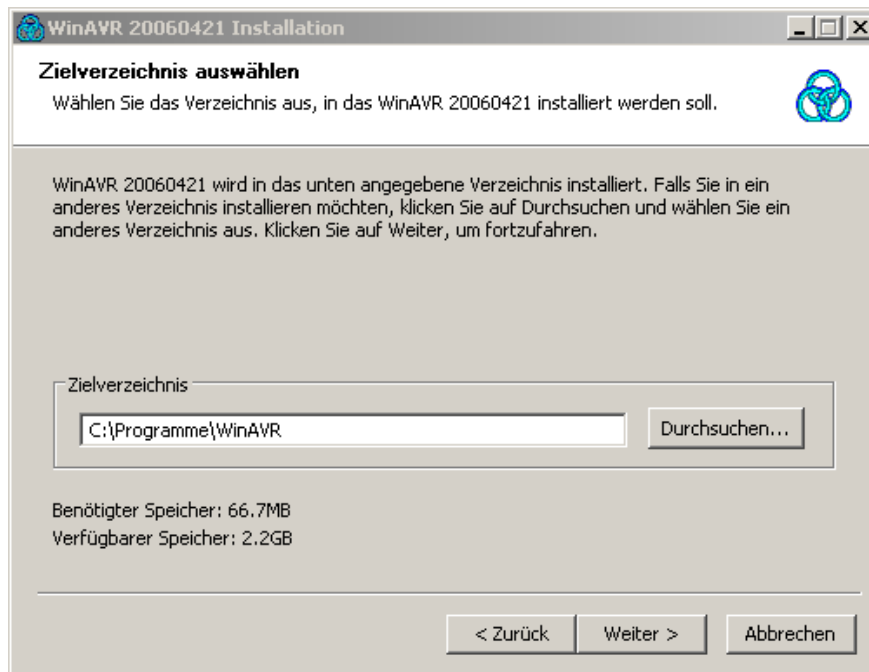


Abbildung C.18.: Auswahl des Zielverzeichnisses

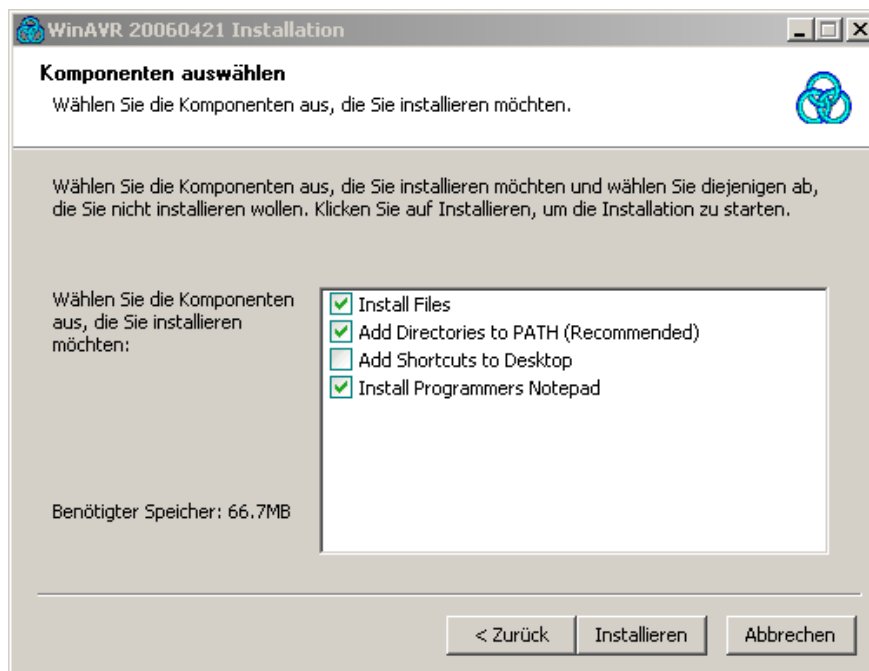


Abbildung C.19.: Selektion der zu installierenden Komponenten

ANHANG C. ENTWICKLUNGSUMGEBUNG

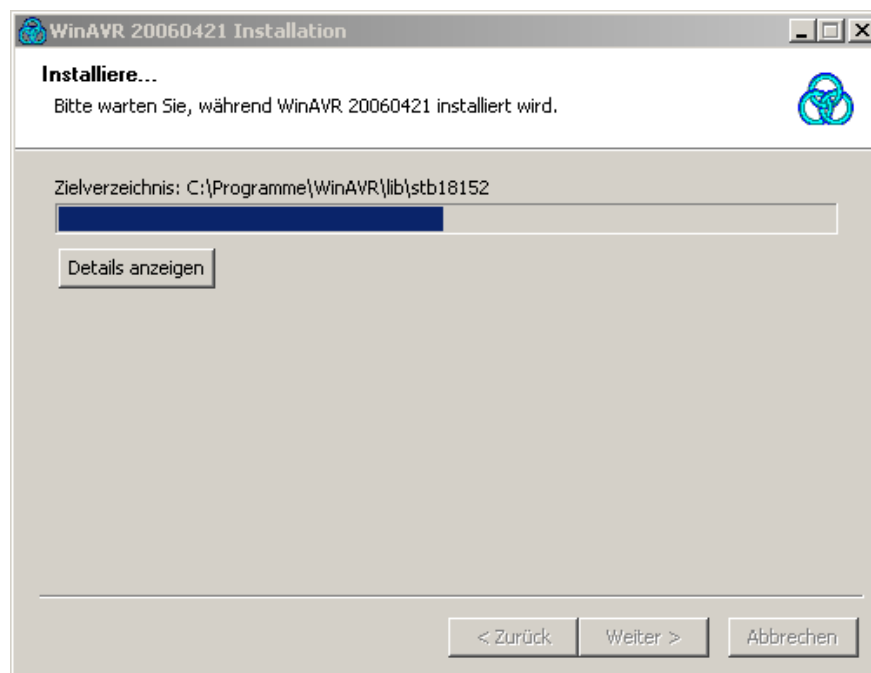


Abbildung C.20.: Kopieren der Installationsdateien

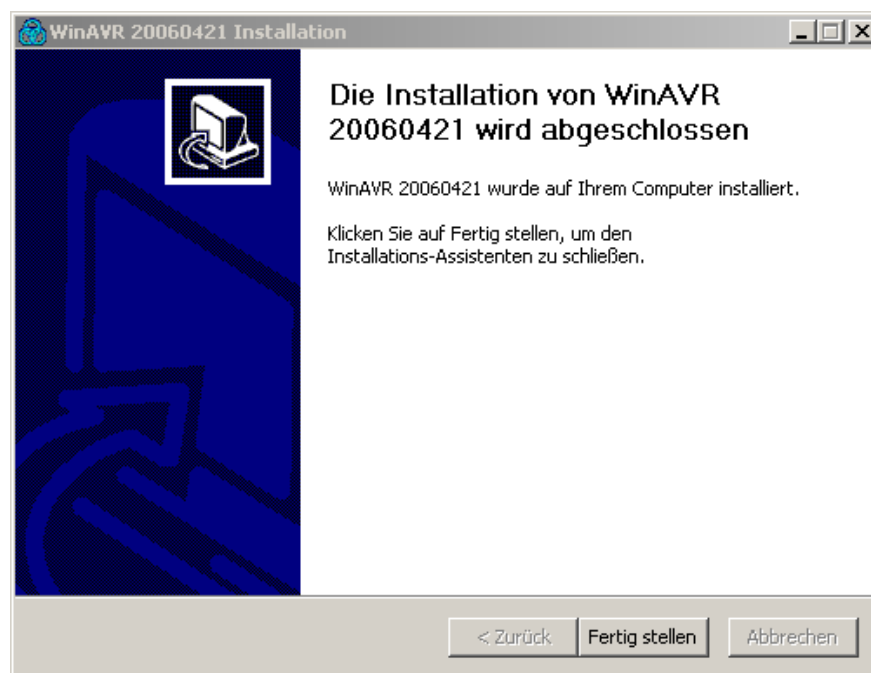


Abbildung C.21.: Beenden der Installationsroutine

C.2.2. AVR Studio

Zum Programmieren und Debuggen des Zielsystems wird das Softwaretool *Atmel AVR Studio* verwendet, welches auf der Webseite der *Atmel Corporation* [23] kostenlos heruntergeladen werden kann. Nach dem Download der beiden Dateien «aStudio4b460.exe» (AVR Studio 4.12 (build 460)) und «aStudio412SP4b498.exe» (AVR Studio 4.12 Service Pack 4 (build 498)) wird die Installation mit einem Doppelklick auf «aStudio4b460.exe» gestartet.

C.2.2.1. Installation

1. Der Begrüßungsbildschirm

Nach dem Starten der Installation erscheint zuerst der Begrüßungsbildschirm (Abbildung C.22). Die Installation wird mit einem Klick auf «Next» fortgesetzt.

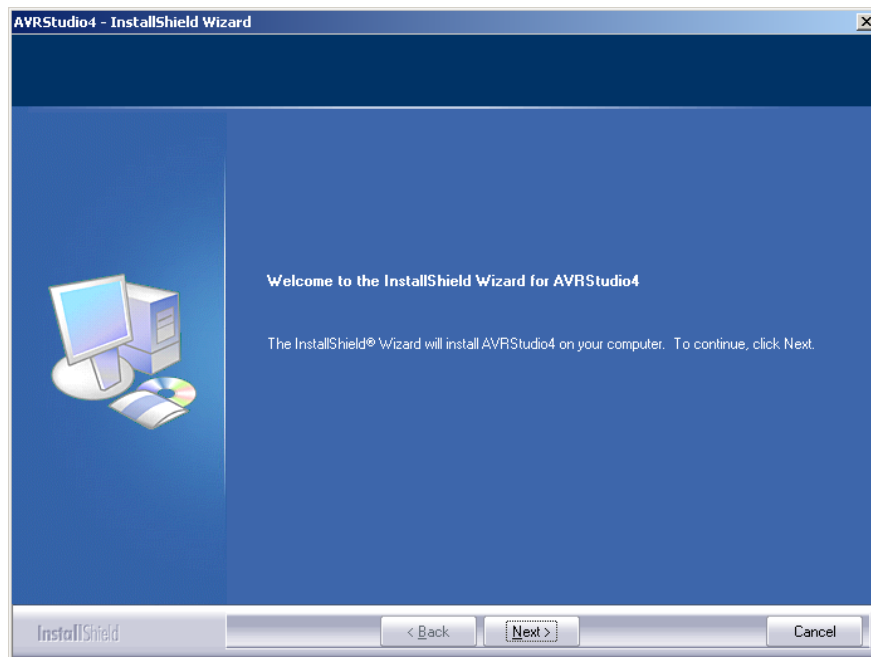


Abbildung C.22.: Anzeige des Begrüßungsbildschirms nach dem Start der Installation

2. Anerkennung der Lizenz

In diesem Schritt der Installation (Abbildung C.23) muss den Lizenzbedingungen durch die Markierung des Feldes «I accept the terms of the license agreement» zugestimmt werden. Mit der Schaltfläche «Next» wird die Installation fortgesetzt.

ANHANG C. ENTWICKLUNGSUMGEBUNG

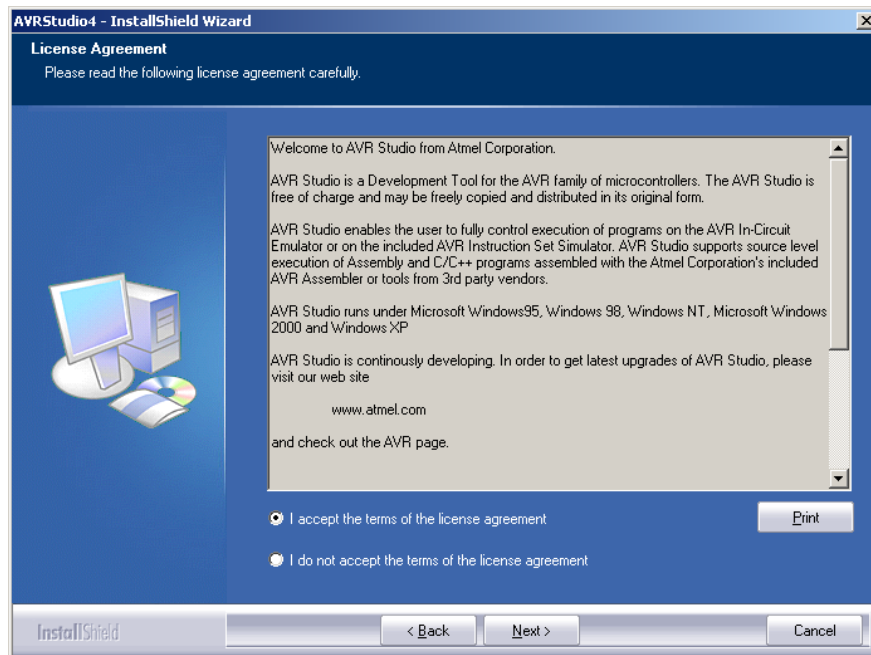


Abbildung C.23.: Anerkennung der Lizenz

3. Auswahl des Installationspfades

Dieses Fenster (Abbildung C.24) bietet die Möglichkeit, den Installationspfad des *Atmel AVR Studios* anzupassen. Normalerweise ist der Pfad bereits korrekt voreingestellt und es kann einfach mit der Schaltfläche «Next» zum nächsten Installationsschritt gewechselt werden.

4. Auswahl der Module

In der nun folgenden Liste (Abbildung C.25) wird der Eintrag «Install/upgrade USB Driver» abgehakt, da sonst später die USB-Programmieradapter nicht verwendet werden können. Mit einem anschließenden Betätigen der Schaltfläche «Next» wird die Auswahl übernommen und das nächste Fenster des Installationsvorgangs angezeigt.

5. Bereit zum Kopieren der Dateien

Das Fenster in Abbildung C.26 weist darauf hin, dass mit dem nächsten Klick auf «Install» der eigentliche Kopiervorgang der Daten gestartet wird. Nach einem Klick auf diese Schaltfläche werden die Daten, wie Abbildung C.27 zeigt, kopiert.

ANHANG C. ENTWICKLUNGSUMGEBUNG

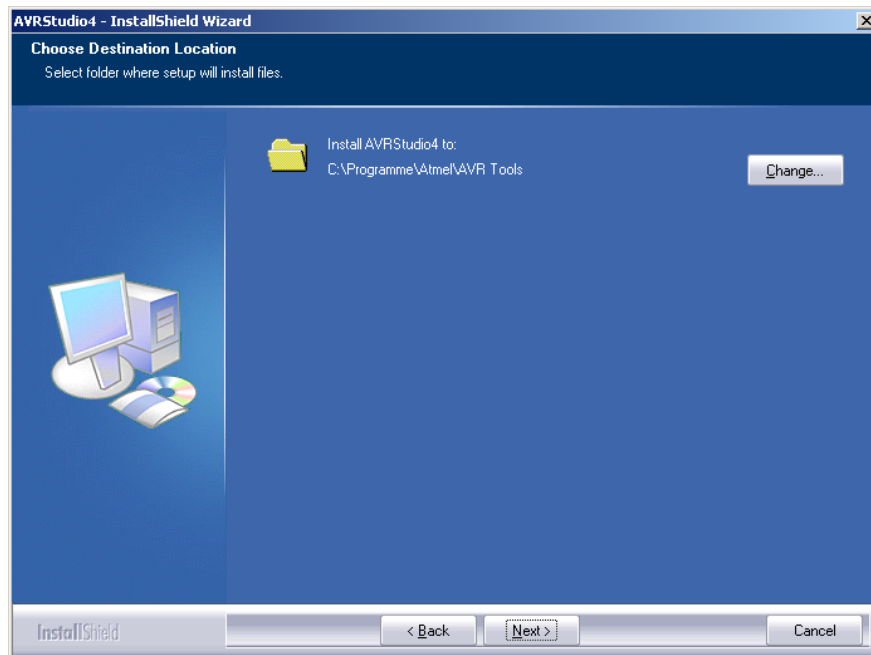


Abbildung C.24.: Auswahl des Installationspfades

6. Installation erfolgreich beendet

Nachdem alle Dateien erfolgreich installiert wurden, wird das Ende der Installation mit dem Fenster in Abbildung C.28 angezeigt. Durch einen letzten Klick auf die Schaltfläche «Finish» wird der Installationsvorgang beendet.

ANHANG C. ENTWICKLUNGSUMGEBUNG

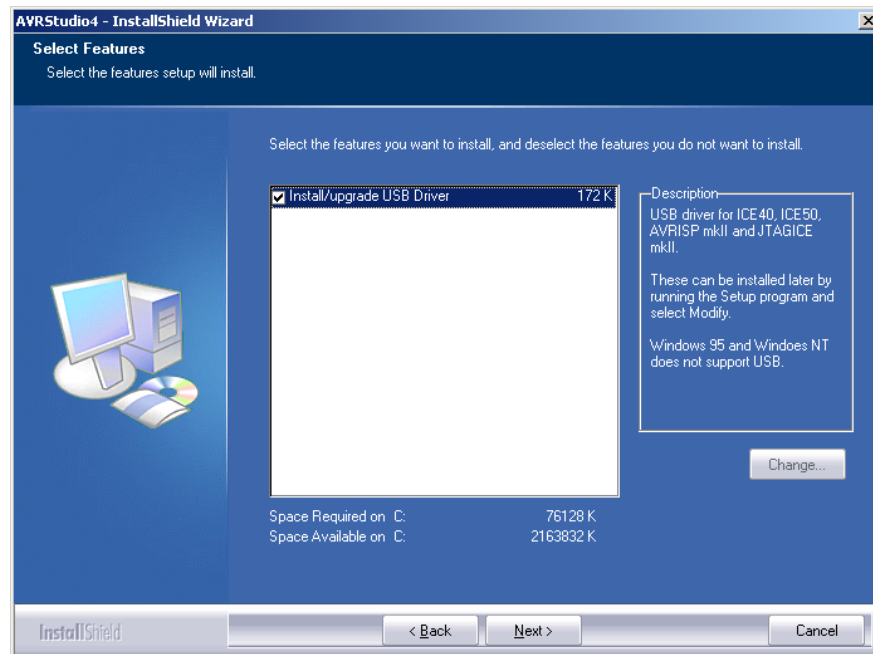


Abbildung C.25.: Auswahl der zu installierenden Module

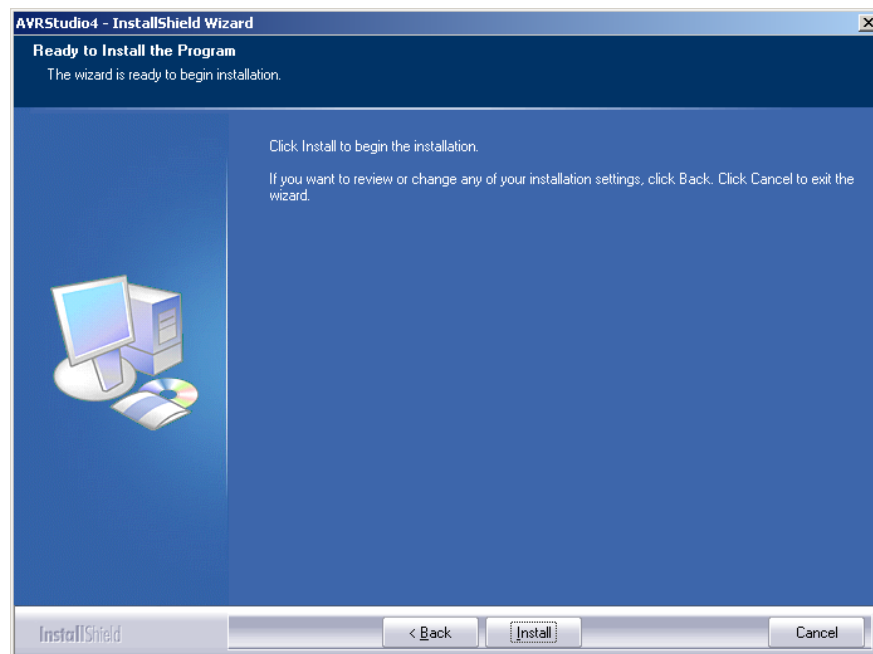


Abbildung C.26.: Bereit zum Kopieren der Dateien

ANHANG C. ENTWICKLUNGSUMGEBUNG

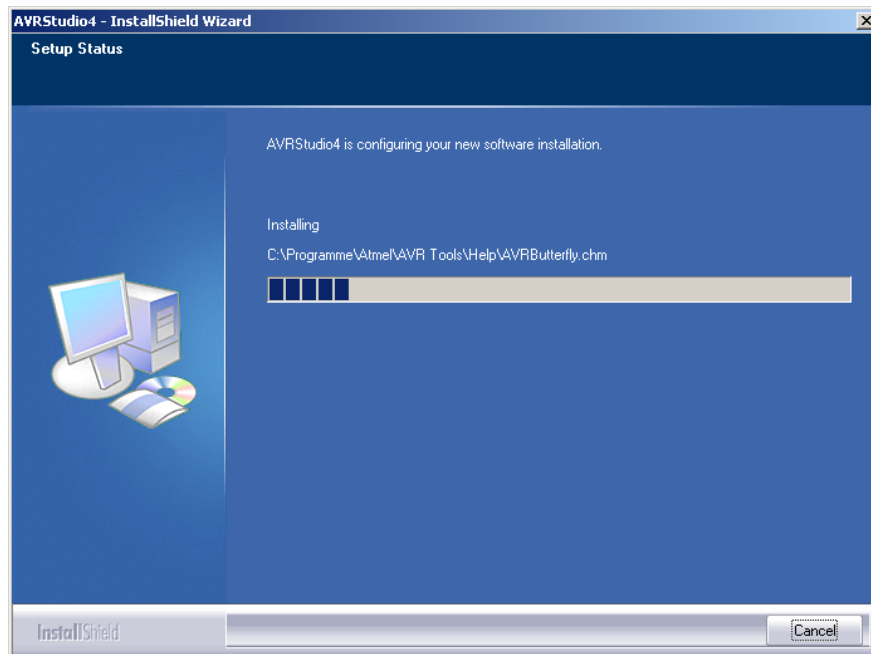


Abbildung C.27.: Kopieren der Installationsdaten

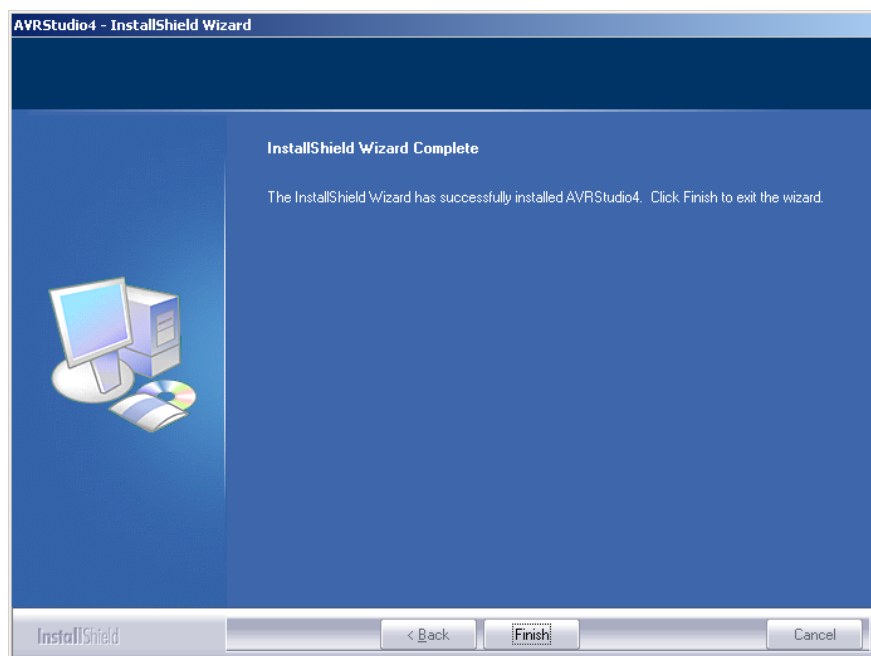


Abbildung C.28.: Meldung über den erfolgreichen Installationsvorgang

C.2.2.2. Installation des Servicepacks

Nachdem das *Atmel AVR Studio* installiert wurde, sollte noch das aktuelle Servicepack 4 installiert werden. Dies behebt einige Probleme der im Vorigen installierten Version des Programms. Vor der Installation dieses Updates sollte der Rechner allerdings sicherheitshalber neu gestartet werden. Nach dem Neustart wird zum Start der Installation die heruntergeladene Datei «aStudio412SP4b498.exe» ausgeführt.

1. Der Begrüßungsbildschirm

Nach dem Start der Installationsroutine wird der Anwender durch das Fenster in Abbildung C.29 begrüßt. Das Betätigen der Schaltfläche «Next» setzt die Installation fort.

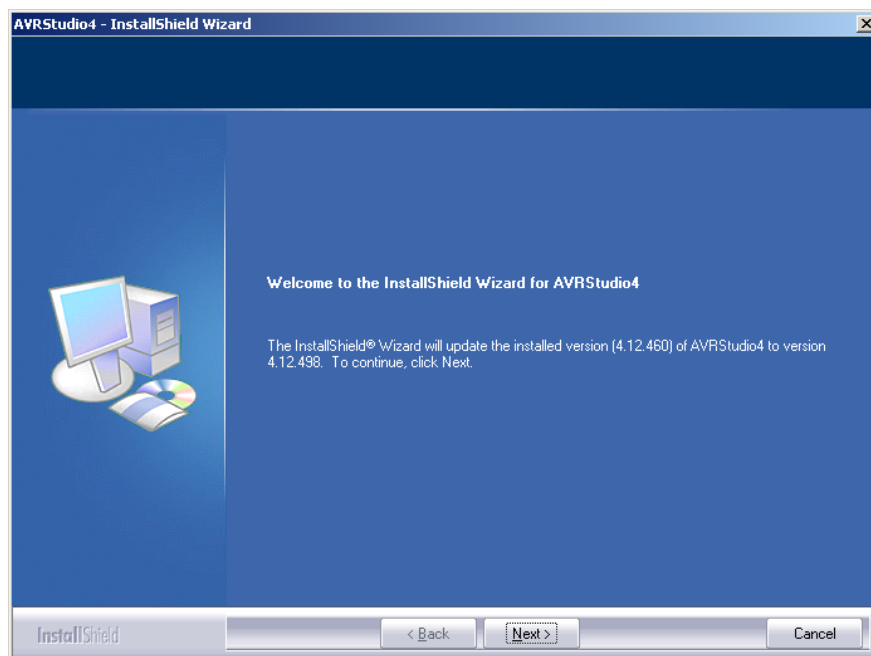


Abbildung C.29.: Begrüßung zu Beginn der Installation

2. Auswahl der Installationsoptionen

Im zweiten Fenster (Abbildung C.30) der Installation werden, wie bei der Installation des *AVR Studios*, die zu installierenden Optionen ausgewählt. Hier muss der Haken bei «Install/upgrade USB Driver» wieder gesetzt sein. Mit «Next» wird die Installation fortgesetzt.

3. Der Kopiervorgang

Abbildung C.31 zeigt den Kopiervorgang der Daten. Sobald dieser erfolgreich abgeschlossen ist, erscheint automatisch das nächste Fenster.

ANHANG C. ENTWICKLUNGSUMGEBUNG

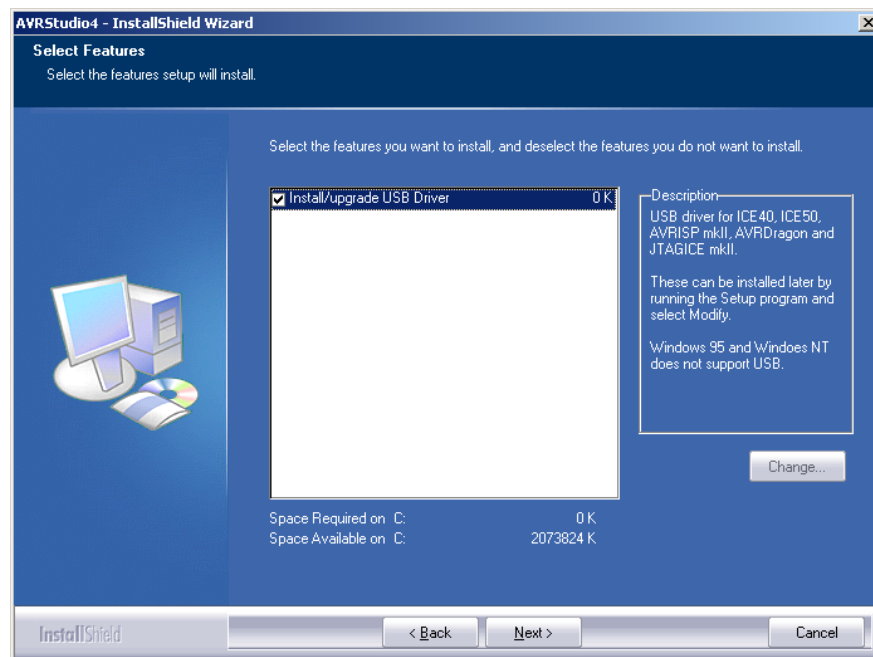


Abbildung C.30.: Auswahl der Installationsoptionen

4. Installation abgeschlossen

Die erfolgreiche Aktualisierung des *Atmel AVR Studios 4.12.460* auf die Version *4.12.498* wird durch Abbildung C.32 signalisiert. Die Installation ist nun beendet und kann durch einen Klick auf «Finish» geschlossen werden.

ANHANG C. ENTWICKLUNGSUMGEBUNG

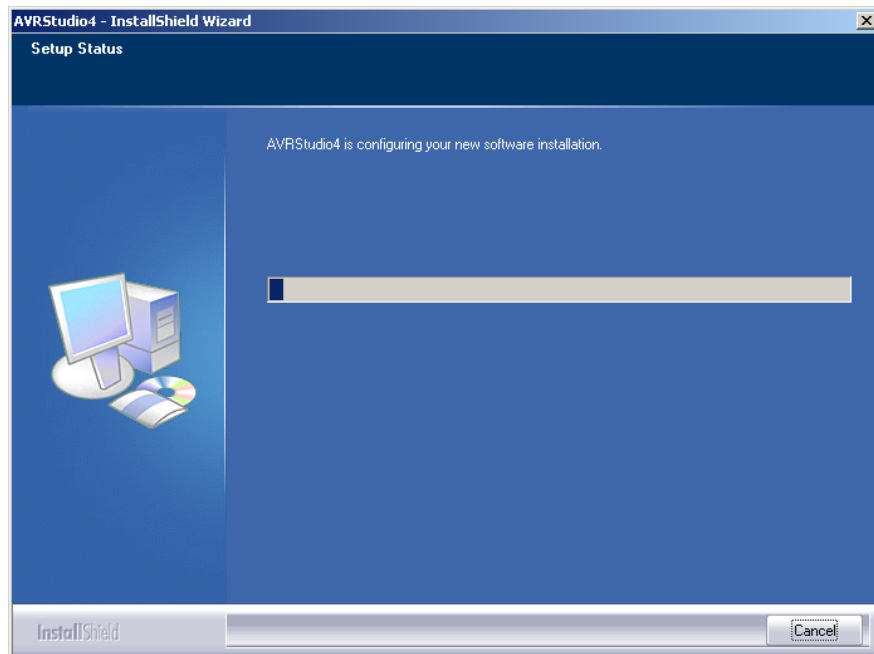


Abbildung C.31.: Kopiervorgang der Daten

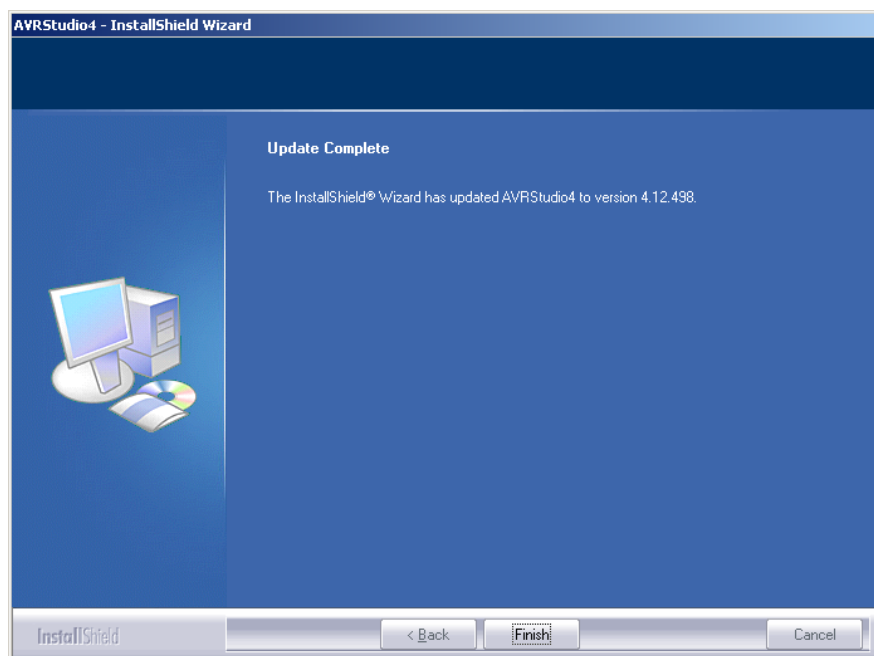


Abbildung C.32.: Installation abgeschlossen

ANHANG C. ENTWICKLUNGSUMGEBUNG

D. Erzeugung der Gerberdaten

Zur Fertigung der Leiterplatte des *STK-LAN* gibt es mehrere Möglichkeiten. Für Laborarbeiten werden die Leiterplatten an der *Hochschule Heilbronn* im Normalfall gefräst. Da für das *STK-LAN* jedoch ein Bestückungsdruck vorgesehen ist, wird die Platine von einer externen Firma namens *Q-print electronic GmbH* [103] professionell gefertigt. Die Fertigungsdaten werden der Firma in Form von Gerber-Dateien zugesandt.

Die Erzeugung dieser Gerber-Dateien mit *EAGLE* ist denkbar einfach. Hierzu wurde im Rahmen dieser Diplomarbeit ein *CAM-Job* mit der Bezeichnung *q-pcb.cam* erstellt, welcher alle nötigen Einstellungen enthält.

Zur Erzeugung der Gerberfiles muss in *EAGLE* zum *Board-Fenster* gewechselt werden. In diesem kann über das Menü «Datei» und den Eintrag «CAM-Prozessor» der *CAM-Prozessor* aufgerufen werden (Abbildung D.1).

Nachdem der *CAM-Prozessor* aufgerufen wurde, wird über das Menü «Datei» des *CAM-Prozessors* und anschließend «Öffnen», «Job...» ein existierender *CAM-Job* geöffnet. Abbildung D.2 verdeutlicht dieses Vorgehen.

In dem daraufhin erscheinenden Öffnen-Dialog wird die Datei «q-pcb.cam»¹ markiert und geöffnet. Nun sind alle Einstellungen in den *CAM-Prozessor* geladen und die Erzeugung der Gerber-Dateien kann durch einen Klick auf die Schaltfläche «Job ausführen» gestartet werden. Während der Erzeugung der Gerber-Dateien können unter Umständen einige Warnmeldungen erscheinen, welche einfach mit einem Klick auf «OK» akzeptiert werden.

Nach der erfolgreichen Erzeugung der Gerber-Daten kann der *CAM-Prozessor* wieder geschlossen werden.

Zu diesem Zeitpunkt sollten nun die folgenden 22 Dateien im *EAGLE-Projektordner* erstellt worden sein:

¹Diese Datei ist auf dem der Diplomarbeit beiliegenden Datenträger vorhanden.

ANHANG D. ERZEUGUNG DER GERBERDATEN

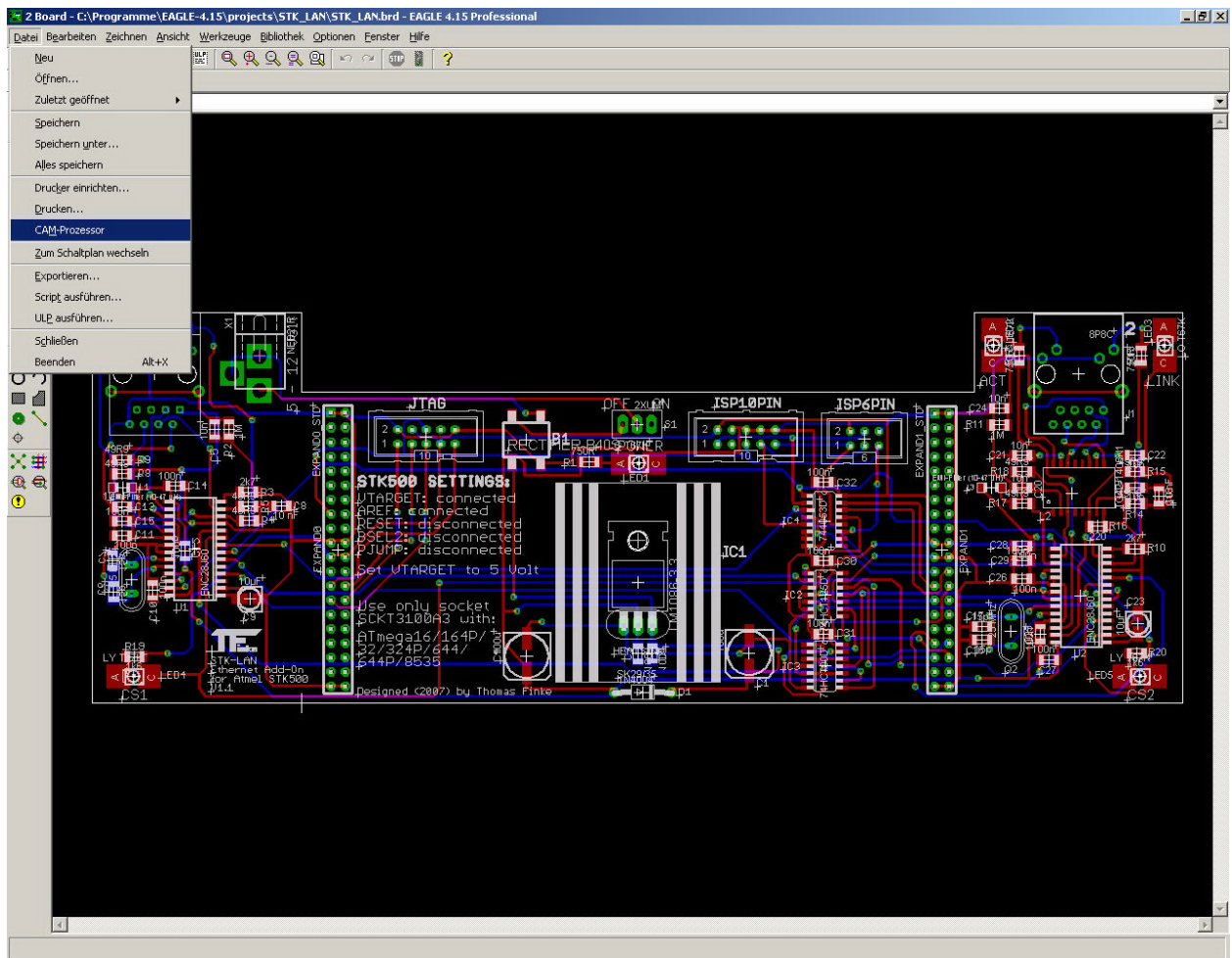


Abbildung D.1.: Starten des CAM-Prozessors

ANHANG D. ERZEUGUNG DER GERBERDATEN

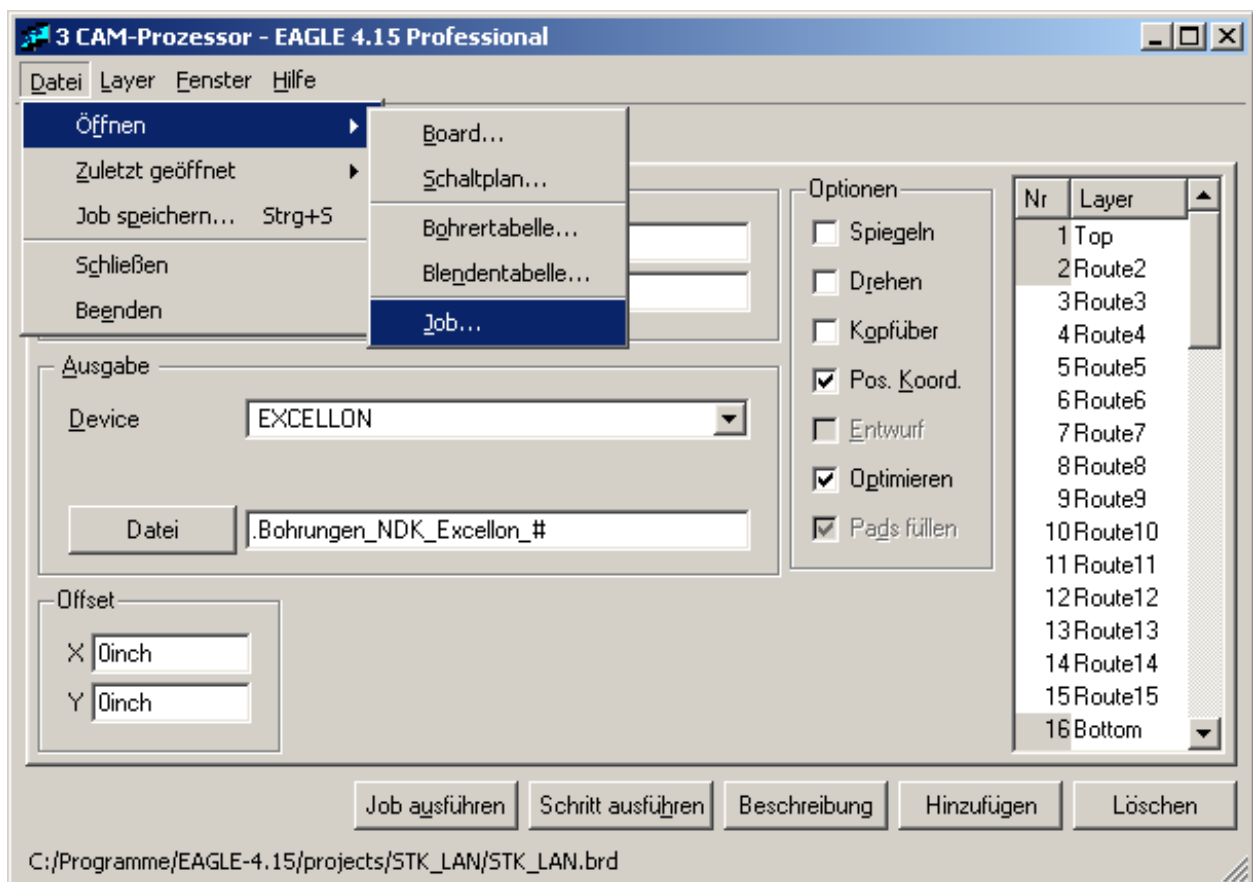


Abbildung D.2.: Öffnen eines CAM-Jobs

ANHANG D. ERZEUGUNG DER GERBERDATEN

- STK_LAN.Bohrungen_DK_Excellon_i
- STK_LAN.Bohrungen_DK_Excellon_x
- STK_LAN.Bohrungen_DK_Gerber_i
- STK_LAN.Bohrungen_DK_Gerber_x
- STK_LAN.Bohrungen_NDK_Excellon_i
- STK_LAN.Bohrungen_NDK_Excellon_x
- STK_LAN.Bohrungen_NDK_Gerber_i
- STK_LAN.Bohrungen_NDK_Gerber_x
- STK_LAN.BOTTOM_Layer_i
- STK_LAN.BOTTOM_Layer_x
- STK_LAN.BOTTOM_Lötstoplack_i
- STK_LAN.BOTTOM_Lötstoplack_x
- STK_LAN.BOTTOM_Positionsdruck_i
- STK_LAN.BOTTOM_Positionsdruck_x
- STK_LAN.Kontur_i
- STK_LAN.Kontur_x
- STK_LAN.TOP_Layer_i
- STK_LAN.TOP_Layer_x
- STK_LAN.TOP_Lötstoplack_i
- STK_LAN.TOP_Lötstoplack_x
- STK_LAN.TOP_Positionsdruck_i
- STK_LAN.TOP_Positionsdruck_x

ANHANG D. ERZEUGUNG DER GERBERDATEN

Die ersten acht Dateien enthalten die Bohrdaten für die Platine einmal im Gerber-Format und einmal im Excellon-Format. Das «DK» und «NDK» stehen für «**D**urch**k**ontak**t**iert» und «**N**icht **D**urch**k**ontak**t**iert». Die Bezeichnungen der restlichen 14 Dateien dürften soweit selbsterklärend sein. Bei allen 22 Dateien ist immer eine «_i»- sowie eine «_x»-Version vorhanden. Die «_i»-Version enthält jeweils Informationen zur verwendeten Maßeinheit oder beispielsweise den verwendeten Bohrern. Die «_x»-Version enthält die eigentlichen Gerber-Daten.

Um die erzeugten Gerber-Daten zu überprüfen, kann beispielsweise der als *Shareware* erhältliche Gerber-Viewer *GC-Prevue* von der Firma *GraphiCode* [46] verwendet werden.

Die 22 Dateien wurden anschließend zusammen mit einer kurzen Info-Datei im *ZIP-Format* gepackt und über das Webformular der *Q-print electronic GmbH* [103] an den Hersteller gesendet.

Die Info-Datei enthielt dabei die folgenden Informationen:

Guten Tag,

hier noch kurz einige Infos zur Platinenfertigung.

Ich habe Ihnen die Daten als Extended Gerber (RS-274X) erstellt.

Die Dateien haben als Dateiendung die Bezeichnung, welche Sie auf Ihrer Internetseite unter der Rubrik "Datenformate" auch für die Lagen verwendet haben. Die zusätzliche Endung "_i" bei einigen Files bedeutet, dass es sich um ein Info-File handelt, welches Zusatzinformationen wie "mm oder inch" beinhaltet.

Die Files mit der Endung "_x" sind die eigentlichen Gerberfiles.

Zusätzlich habe ich Ihnen die Bohrdaten nochmals im Excellon-Format mitgesendet. Die Excellon-Daten habe ich nach dem Erstellen nochmals von Hand überprüft.

Die Platine soll nach den nachfolgenden Punkten gefertigt werden:

- Maße 69mm x 193mm (U-Form)
- 2 Lagen
- Lötstoplack beidseitig
- Positionsdruck nur TOP-Layer

Ich bitte Sie, die Daten vor der Fertigung nochmals zu prüfen, vor

ANHANG D. ERZEUGUNG DER GERBERDATEN

allem die Abstände zwischen den Leiterbahnen. Laut "Design Rule Check" sollten die Abstände 0.2mm nicht unterschreiten, aber teilweise sind die Bahnen recht eng verlegt.

Die Gerberfiles sind so erstellt, dass alle Lagen als Draufsicht von der Bestückungsseite aus dargestellt sind. Die Schrift auf der Lötseite erscheint also gespiegelt in den Gerberfiles !!!

E. Modifikation des Makefile

Eine der wichtigsten Dateien zur Erstellung des Mikrocontrollerprogramms ist das *Makefile*. In diesem sollten Änderungen nur mit äußerster Vorsicht gemacht werden. Zur Anpassung des *Makefiles* an andere Konfigurationen des Zielsystems sind lediglich drei Codezeilen von Bedeutung:

1. MCU

```
# MCU name
MCU = atmega32
```

Mit dieser Angabe wird der verwendete Mikrocontroller gewählt. Im Beispiel handelt es sich um einen *ATmega32*.

2. F_OSC

```
# Main Oscillator Frequency
# This is only used to define F_OSC in all assembler and c-sources.
F_OSC = 8000000 #7372800 #3686400
```

Durch *F_OSC* kann die Frequenz der verwendeten Taktquelle gewählt werden. Die Angabe steht anschließend unter der Variablen *F_OSC* im Programm zur Verfügung. Sie kann beispielsweise dazu verwendet werden, den Wert zu berechnen, mit welchem ein *Timer* geladen werden muss.

3. SRC

Anhand dieser Angabe erhält der *C-Compiler* Informationen über die zu verwendenden *C-Dateien*. Jede *C-Datei*, welche zum Erstellen des Programms benötigt wird, muss an dieser Stelle aufgeführt sein.

ANHANG E. MODIFIKATION DES MAKEFILE

```
# List C source files here.  
# (C dependencies are automatically generated.)  
SRC = $(TARGET).c net/ip.c net/udp.c net/arp.c net/mynic.c  
net/nethelp.c net/icmp.c io/clock.c io/delay.c io/enc28j60.c  
net/tcp.c net/httpd.c util/string.c io/port.c net/base64.c  
io/a2d.c net/snmp_agent.c net/snmp_datastructure.c net/mib.c  
net/mib_sync.c io/uart.c io/terminalserver.c
```

F. Schaltplan und Layout

Der Schaltplan des *STK-LAN* ist auf insgesamt vier Teile aufgeteilt. Der erste Teil in Abbildung F.1 stellt die Spannungsversorgung dar. Die Teile zwei und drei (Abbildungen F.2 und F.3) zeigen die beiden Ethernetports. In Abbildung F.4 werden schließlich noch die Steckverbinder sowie die Logikbausteine der Schaltung aufgezeigt. Alle für das *STK-LAN* benötigten Bauelemente sind in der Stückliste in Anhang G aufgeführt.

Das Layout des *STK-LAN* erstreckt sich über zwei Lagen einer Platine. Der *Top*- und *Bottom-Layer* sind den Abbildungen F.5 und F.6 zu entnehmen. Eine Anleitung zur Erstellung der Gerberdaten für die Platinenfertigung ist in Anhang D vorhanden.

Die Informationen zu den verschiedenen Versionen des *STK-LAN* sind in Anhang H zusammengestellt.

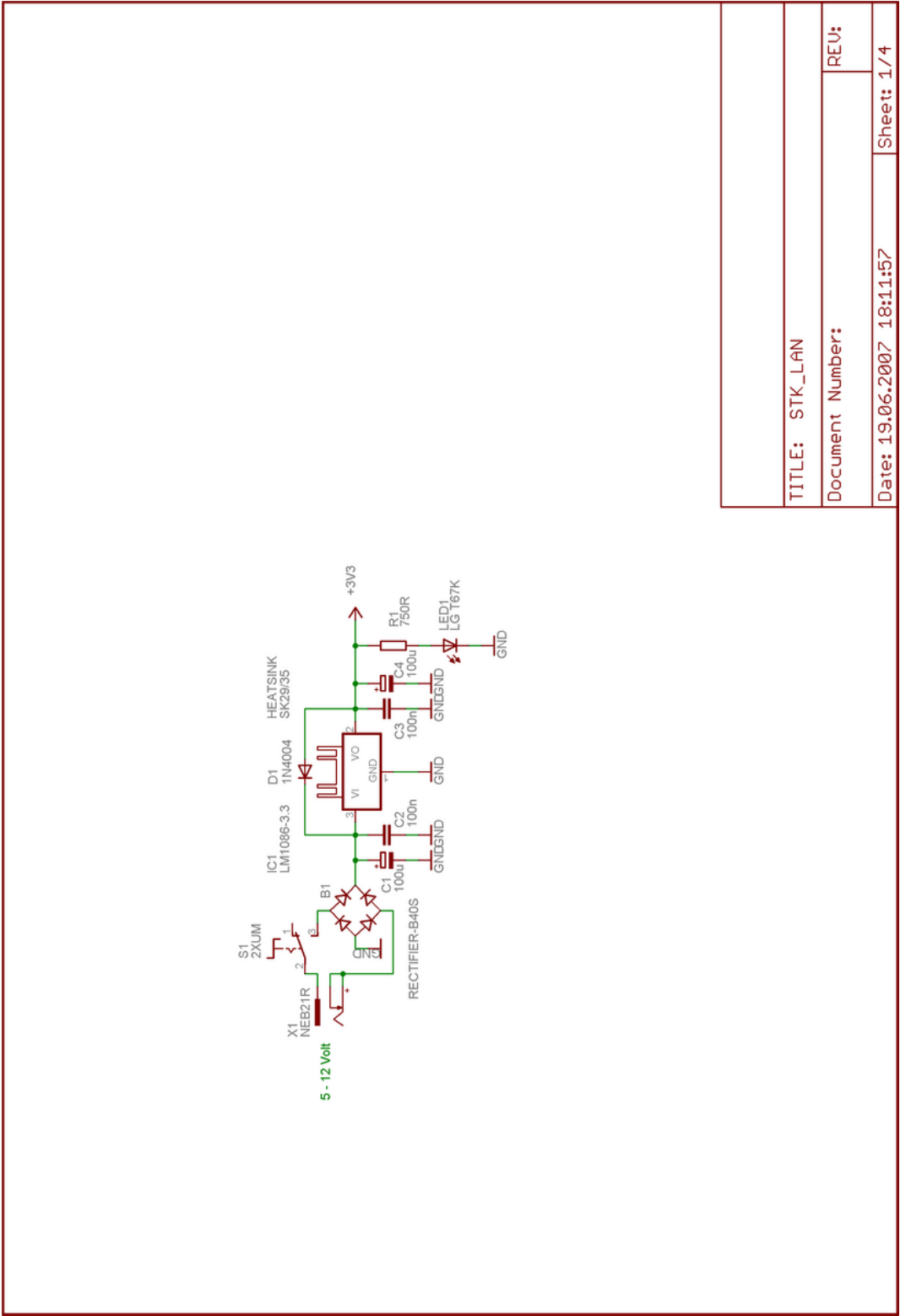


Abbildung F.1.: Schaltplan des STK-LAN (Seite 1/4: Spannungsversorgung)

ANHANG F. SCHALTPLAN UND LAYOUT

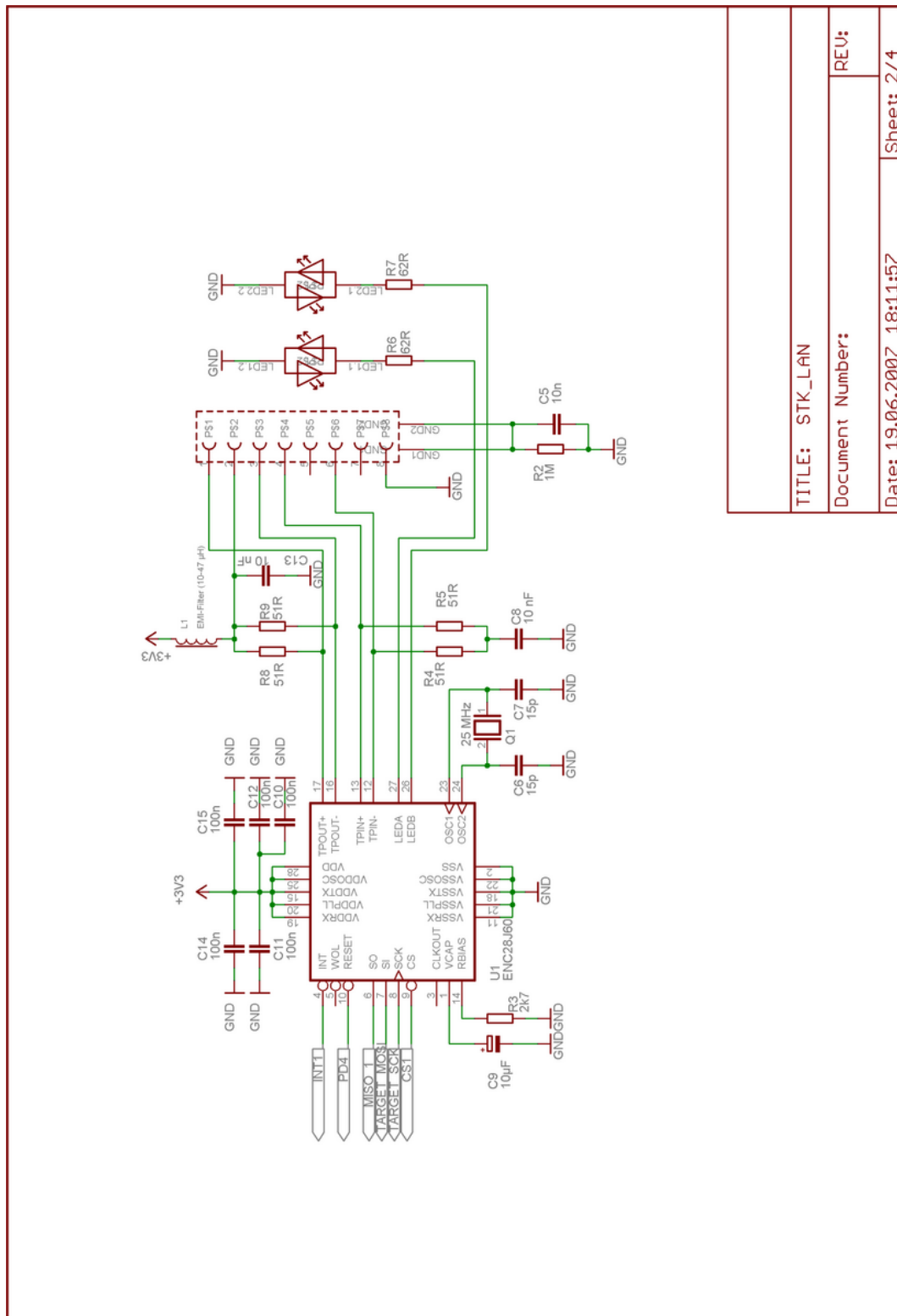


Abbildung F.2.: Schaltplan des STK-LAN (Seite 2/4: Ethernetport 1)

ANHANG F. SCHALTPLAN UND LAYOUT

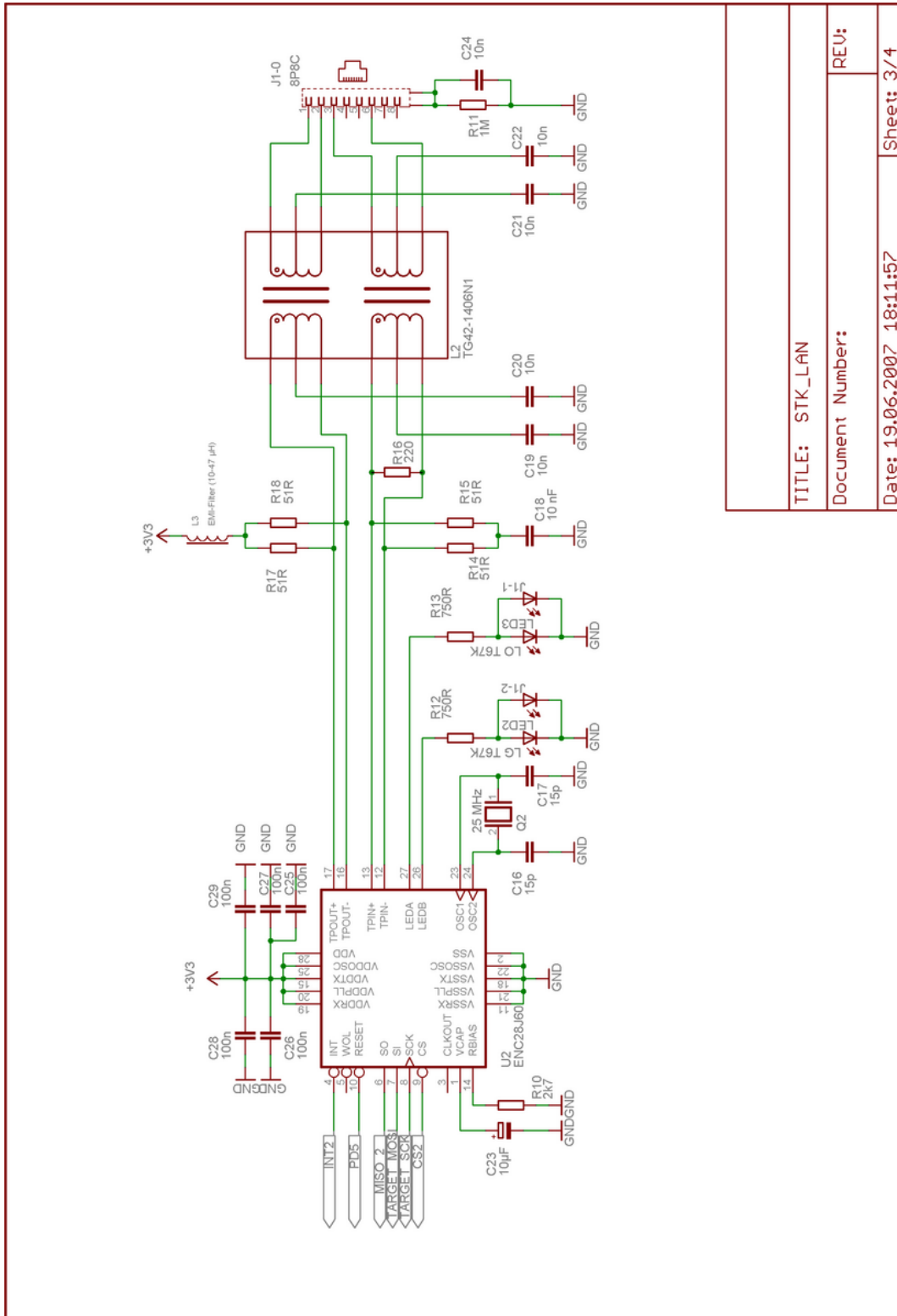


Abbildung F.3.: Schaltplan des STK-LAN (Seite 3/4: Ethernetport 2)

ANHANG F. SCHALTPLAN UND LAYOUT

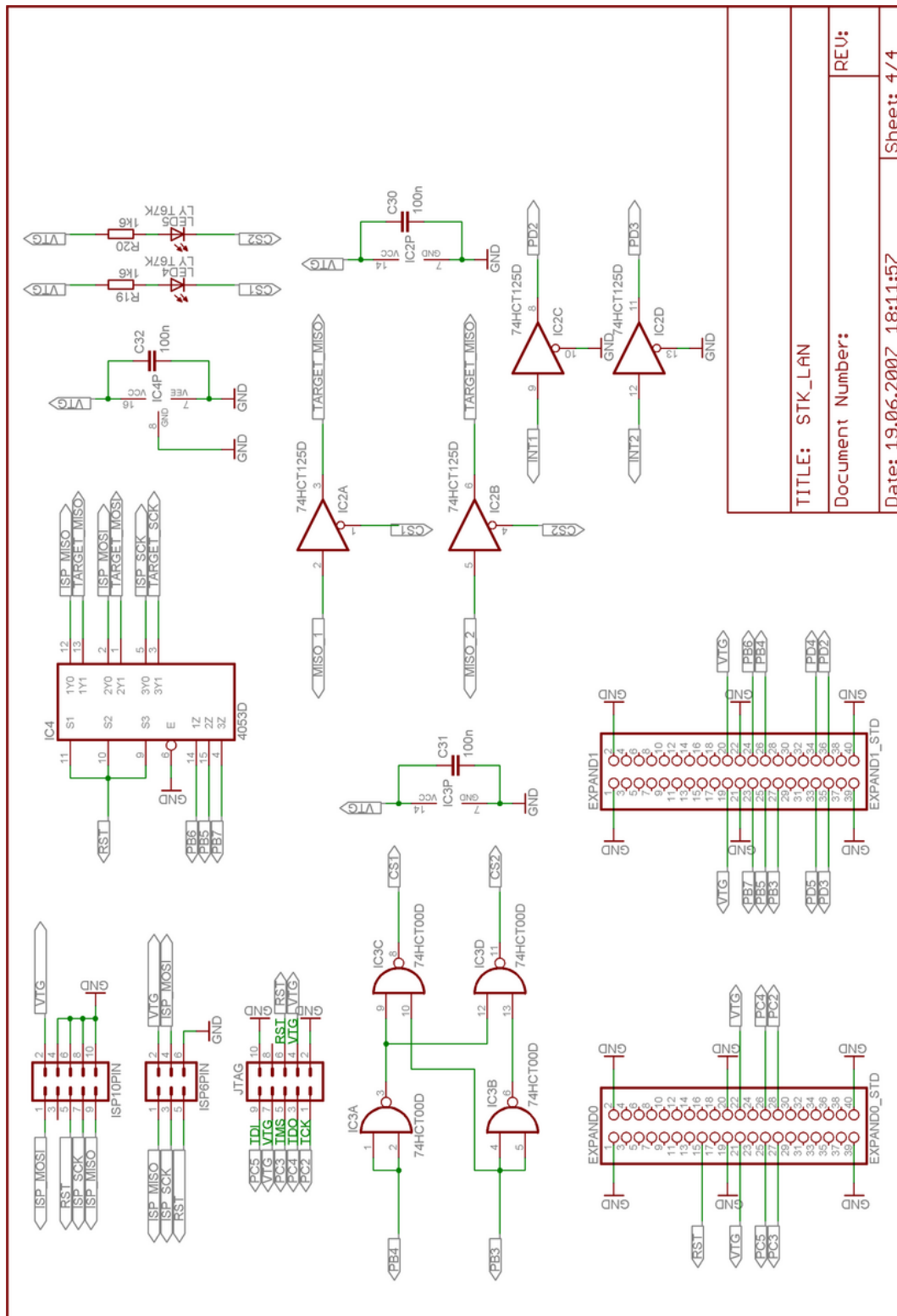


Abbildung F.4.: Schaltplan des STK-LAN (Seite 4/4: Steckverbinder und Logikbausteine)

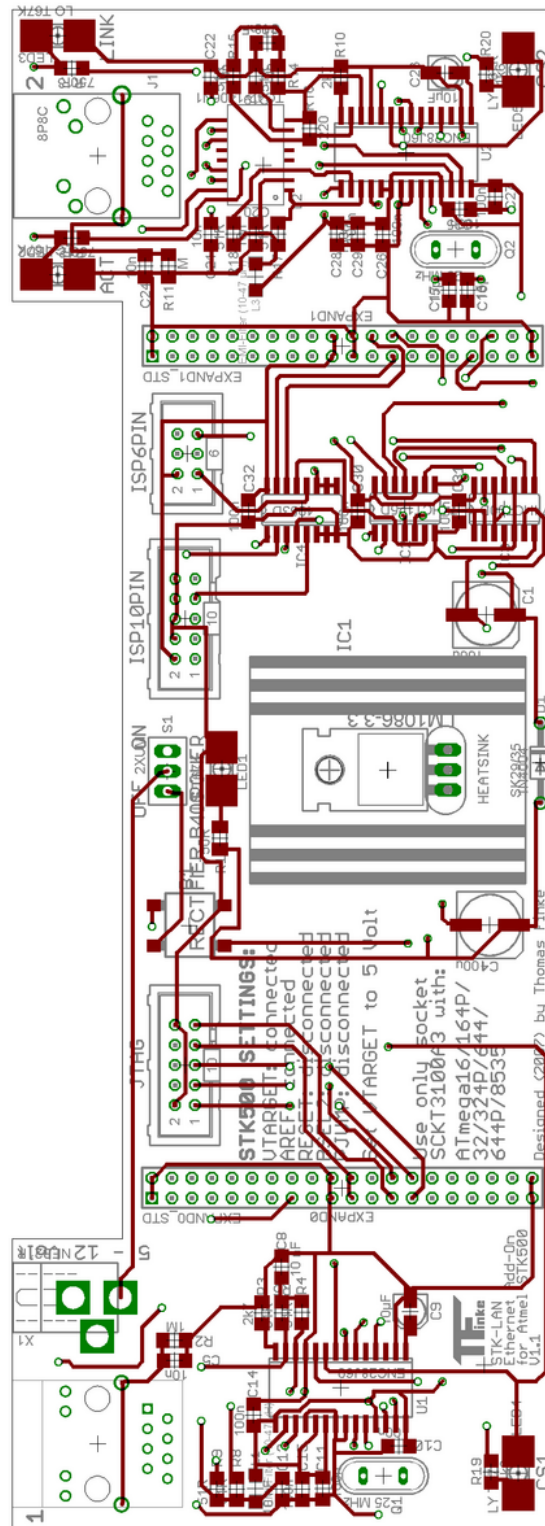


Abbildung F.5.: Layout des *STK-LAN* (Top)

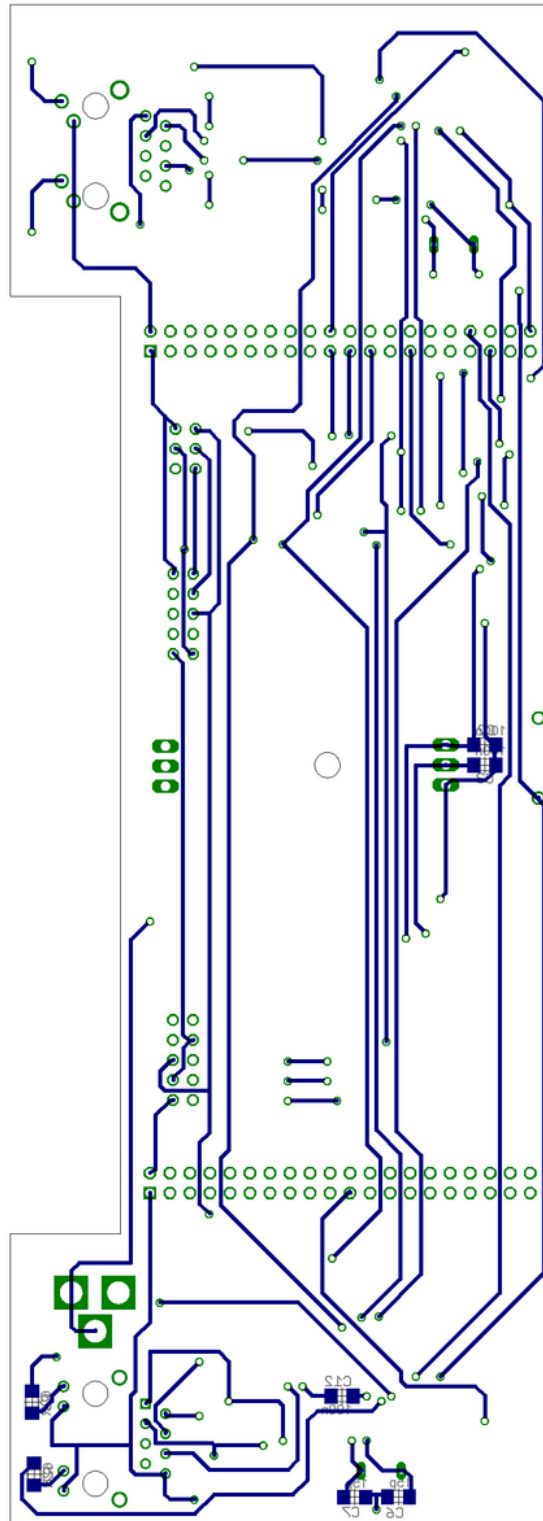


Abbildung F.6.: Layout des *STK-LAN* (Bottom)

ANHANG F. SCHALTPLAN UND LAYOUT

G. Stückliste

Alle für das *STK-LAN* benötigten Bauteile wurden von den drei Versandhäusern *Reichelt Elektronik e. Kfr.* [32], *SEGOR-electronics GmbH* [33] und *Microcontroller-Starterkits* [122] bezogen. Eine Auflistung aller Bauteile mit den dazugehörigen Bestellnummern und Preisen sind Tabelle G.1 zu entnehmen.

Tabelle G.1.: Stückliste des STK-LAN

Anzahl	Bestellnummer	Beschreibung	Bezugsquelle	Stückpreis
8	SMD 1/4W 51	SMD-Chip-Widerstand, Bauform 1206, 51 Ω	Reichelt	0,08 €
2	SMD 1/4W 62	SMD-Chip-Widerstand, Bauform 1206, 62 Ω	Reichelt	0,10 €
3	SMD 1/4W 750	SMD-Chip-Widerstand, Bauform 1206, 750 Ω	Reichelt	0,10 €
2	SMD 1/4W 1,6K	SMD-Chip-Widerstand, Bauform 1206, 1.6 K Ω	Reichelt	0,10 €
2	SMD 1/4W 2,7K	SMD-Chip-Widerstand, Bauform 1206, 2.7 K Ω	Reichelt	0,10 €
2	SMD 1/4W 1,0M	SMD-Chip-Widerstand, Bauform 1206, 1 M Ω	Reichelt	0,10 €
4	15p-1206-NPO	15pF COG/NPO 63V 1206	Segor	0,05 €
9	X7R-G1206 10N	SMD-VIELSCHICHT 10nF	Reichelt	0,06 €
15	X7R-G1206 100N	SMD-VIELSCHICHT 100nF	Reichelt	0,09 €
2	10u-16V SMD	SMD-Elko 4x5,5mm 10 μ F 16V	Segor	0,15 €

ANHANG G. STÜCKLISTE

Anzahl	Bestellnummer	Beschreibung	Bezugsquelle	Stückpreis
2	100u-50V SM-D/105'lowESR	SMD-Elko 8x10,5mm 100 μ F 50V	Segor	0,60 €
1	AWHW 06	Steckerwanne 6pol 180'	Segor	1,00 €
2	AWHW 10	Steckerwanne 10pol 180'	Segor	0,80 €
2	SL 2X25G 2,54	2x25pol.-Stiftleiste, gerade, RM 2,54	Reichelt	0,24 €
1	HEBW 21	Hohlstecker-Einbaubuchse, gewinkelt	Reichelt	0,27 €
1	HS 21-9	Hohlstecker, Øi=2,1mm Øa=5,5mm	Reichelt	0,09 €
1	MS-611 A	Sub-min Kippschalter	Segor	1,70 €
1	SMD DF 04	CHIP-GLEICHR. = DF 06	Reichelt	0,18 €
1	1N 4001	DIODE	Reichelt	0,02 €
1	LM 1086 IT3,3	Festspannungsregler +3,3V, 1,5A, TO-2	Reichelt	1,05 €
2	4019	ENC28J60_SO	Microcontroller-Starterkits.de	6.50 €
1	74HCT 00-SMD	Quad 2-In NAND SO14	Segor	0,25 €
1	74HCT 125-SMD	Quad Buffer/Driver 3-S	Segor	0,25 €
1	4053-SMD	Triple 2-Ch.Mux/Demux SO16	Segor	0,30 €
2	LG T67K	OSRAM TOPLED, Low Current, 9 mcd, grün	Reichelt	0,12 €
1	LO T67K	OSRAM TOPLED, Low Current, 28 mcd, orange	Reichelt	0,13 €
2	LY T67K	OSRAM TOPLED, Low Current, 22 mcd, gelb	Reichelt	0,09 €
1	V 4330K	Rippen-Kühlkörper, 35x29x12mm, 12K/W	Reichelt	0,77 €
2	4182	Quarz 25 MHz	Microcontroller-Starterkits.de	0.60 €

ANHANG G. STÜCKLISTE

Anzahl	Bestellnummer	Beschreibung	Bezugsquelle	Stückpreis
2	BLM31A601SPT	EMI-Filter 1206 SMD	Segor	0,20 €
1	PE-65745	Übertrager SMD (CS 8900)	Segor	4,90 €
1	4276	MagJack	Microcontroller- Starterkits.de	5,50 €
1	2149	RJ-45-Buchse	Microcontroller- Starterkits.de	0.95 €

Zusätzlich zu diesen Bauteilen wird noch die Platine für das *STK-LAN* benötigt, welche mit den in Anhang D erzeugten Gerber-Daten beispielsweise von der Firma *Q-print electronic GmbH* [103], als Prototyp für 56,86 € bezogen werden kann.

ANHANG G. STÜCKLISTE

H. Versionen des STK-LAN

Das erste aufgebaute *STK-LAN* besitzt die Version 1.0 und wurde während dem Zeitraum dieser Diplomarbeit dazu verwendet, den entwickelten Code zu testen. Die Version 1.0 enthält allerdings einige kleinere Bugs. Der gravierendste Fehler liegt in der Logik zur Generierung der beiden Chipselect-Signale. Dieser wurde direkt nach dem Aufbau der Platine durch eine kleine Drahtbrücke unter *IC3* behoben.

Die mit dieser Diplomarbeit abgegebenen Unterlagen enthalten den Schaltplan und das Layout der Version 1.1, welche die folgenden Änderungen enthält:

- Beschriftung «BSEL: disconnected» wurde in «BSEL2: disconnected» geändert.
- Die Angabe «5 - 18 Volt» der Versorgungsspannung wurde auf «5 - 12 Volt» geändert, da der Spannungsregler bei 18 Volt sehr warm wird.
- Der oben genannte Fehler in der Logik zur Generierung der Chipselect-Signale wurde im Schaltplan und Layout behoben.
- Die beiden Interruptsignale der Ethernetcontroller werden in Version 1.1 über *IC2* auf 5-Volt-Pegel gebracht, bevor diese zum Mikrocontroller weitergereicht werden.

Die Versionsnummer des *STK-LAN* ist auf der Platine aufgedruckt.

ANHANG H. VERSIONEN DES STK-LAN

I. Erweiterungen des SNMP-Agent

I.1. Einbindung weiterer *Managed Objects*

Auf eine einfache Einbindung weiterer *Managed Objects* in den *MIB-Tree* wurde bei der Realisierung des *SNMP-Agent* besonderer Wert gelegt. Hierzu müssen lediglich die folgenden Schritte abgearbeitet werden:

1. Einfügen des Objekts in den *MIB-Tree*

Hierzu muss in der Datei *mib.c*, wie in Kapitel 3.3.5.3 beschrieben, ein neuer Array-Eintrag erstellt werden, welcher die Daten für das neue *Managed Object* enthält. Die Indizes der Array-Einträge müssen hierbei fortlaufend gewählt werden.

2. Anlegen des internen Namens des Objekts

In der Datei *mib.h* muss ein neuer interner Name für das *Managed Object* erstellt werden.

3. Erhöhen der maximalen Größe des MIB-Array

In der Datei *snmp_datastructure.h* muss über die Variable *SNMP_MAXIMUM_NUMBER_OF_OID* die maximale Größe des MIB-Array angepasst werden. Diese Variable muss die Anzahl der verwendeten *Managed Objects* enthalten.

4. Erstellen einer Sync-Routine

Für den Fall, dass der Variablenwert des neuen *Managed Object* mit einer anderen Größe im Mikrocontrollerprogramm synchronisiert werden soll, muss in der Datei *mib_sync.c* eine Synchronisationsroutine in der Funktion *void MIB_Sync(void)* hinzugefügt werden.

I.2. Realisierung eines MIB-Compilers

Da die *Managed Objects*, wie in Kapitel 3.3.5.3 beschrieben, in einem Array organisiert sind, müssen diese momentan noch von Hand eingetragen werden. Es ist jedoch denkbar, hierfür ein kleines Kommandozeilentool zu schreiben, welches die *MIB-Datei* «STK-LAN.mib» einliest, analysiert und daraus die Datei *mib.c* erstellt. Das Kommandozeilentool kann bei Bedarf in das *Makefile* eingebunden werden, so dass der *MIB-Tree* durch den Kompiliervorgang automatisch erzeugt wird.

Wie in Kapitel I.1 gezeigt, müssen bei einer Änderung im *MIB-Tree* einige Variablen in anderen C-Dateien angepasst werden. Diese Variablen können bei einer automatischen Generierung der *mib.c* in eine eigene Datei ausgelagert werden, welche dann auch vom Kommandozeilentool erstellt wird.

J. Netzwerkonfiguration des STK-LAN

Zur Konfiguration des Netzwerkstacks wird die Datei *config.h* verwendet. In ihr werden mit *NIC_IP_ADDRESS* die *IP-Adresse* des Systems, mit *NIC_IP_NETMASK* die *Subnetmask* und mit *NIC_GATEWAY_IP_ADDRESS* der zu verwendende *Gateway* angegeben.

Von Herrn *Jörg Storch* [124] des Rechenzentrums der *Hochschule Heilbronn* wurden die in Tabelle J.1 aufgeführten *IP-Adressen* mit den zugehörigen Namen für die Verwendung mit dem *STK-LAN* reserviert. Im Mikrocomputerlabor der *Hochschule Heilbronn* steht an der Netzwerkdose mit der Bezeichnung «C8 - D12» der rechte Anschluss für die Verwendung mit diesen *IP-Adressen* zur Verfügung.

Die zu verwendende *Subnetmask* ist die 255.255.255.0. Der *Gateway* hat die Adresse 141.7.24.254. Für die Auflösung von Netzwerknamen können die *Domain Name Server* 141.7.1.18 und 141.7.1.20 verwendet werden.

Für den *SNMP-Agent* wird mit *SNMP_MANAGER_IP_ADDRESS* die *IP-Adresse* des *SNMP-Managers* eingestellt.

Während der Erstellung dieser Diplomarbeit wurde für das *STK-LAN* die *IP-Adresse* 192.168.10.123 verwendet. Daher erscheint diese *IP-Adresse* in einigen der Abbildungen und an manchen Stellen im Text.

Die *MAC-Adresse* des Systems wird mit den Variablen *NIC_MAC0* bis *NIC_MAC5* gewählt. Bei der Auswahl der *MAC-Adresse* muss darauf geachtet werden, dass die Adresse kein zweites Mal im Netzwerk vergeben ist. Die in dieser Diplomarbeit verwendeten *MAC-Adressen* haben die Form «AC-DE-48-xx-xx-xx». Dieser Bereich ist laut «The public OUI listing» [9] als *PRIVATE* definiert.

ANHANG J. NETZWERKONFIGURATION DES STK-LAN

IP-Adresse	Name
141.7.24.140	f222-0.te1.hs-heilbronn.de
141.7.24.141	f222-1.te1.hs-heilbronn.de
141.7.24.142	f222-2.te1.hs-heilbronn.de
141.7.24.143	f222-3.te1.hs-heilbronn.de
141.7.24.144	f222-4.te1.hs-heilbronn.de
141.7.24.145	f222-5.te1.hs-heilbronn.de
141.7.24.146	f222-6.te1.hs-heilbronn.de
141.7.24.147	f222-7.te1.hs-heilbronn.de
141.7.24.148	f222-8.te1.hs-heilbronn.de
141.7.24.149	f222-9.te1.hs-heilbronn.de

Tabelle J.1.: Für das STK-LAN reservierte IP-Adressen an der Hochschule Heilbronn

K. Übersicht der implementierten Managed Objects

Der implementierte *SNMP-Agent* stellt die nachfolgend genannten *Managed Objects* zur Verfügung:

- sysDescr

Dieses *Managed Object* enthält eine kurze Systembeschreibung.

- Wert: «ATmega32 with embedded ethernet»
- OID: .1.3.6.1.2.1.1.1.0
- Zugriffsart: read only

- sysContact

In dieser Variablen ist die Kontaktadresse des Systemadministrators vorhanden.

- Wert: «tfinke@stud.hs-heilbronn.de»
- OID: .1.3.6.1.2.1.1.4.0
- Zugriffsart: read only

- sysName

Durch dieses Objekt wird der Name des Systems angegeben.

- Wert: «STK-LAN»
- OID: .1.3.6.1.2.1.1.5.0
- Zugriffsart: read only

ANHANG K. ÜBERSICHT DER IMPLEMENTIERTEN MANAGED OBJECTS

- HID LEDs

Mit Hilfe dieses *Managed Objects* werden die *LEDs* auf dem *STK500* angesprochen.

- Wert: 0 - 255
- OID: .1.3.6.1.4.1.4766.3.2.2.0
- Zugriffsart: read / write

L. Die CD-ROM

Die dieser Diplomarbeit beiliegende CD-ROM enthält alle verwendeten Programme, Sourcecodes, Datenblätter und sonstigen Unterlagen. Zum einfacheren Auffinden bestimmter Dateien dient die Verzeichnisstruktur aus Abbildung L.1.

ANHANG L. DIE CD-ROM

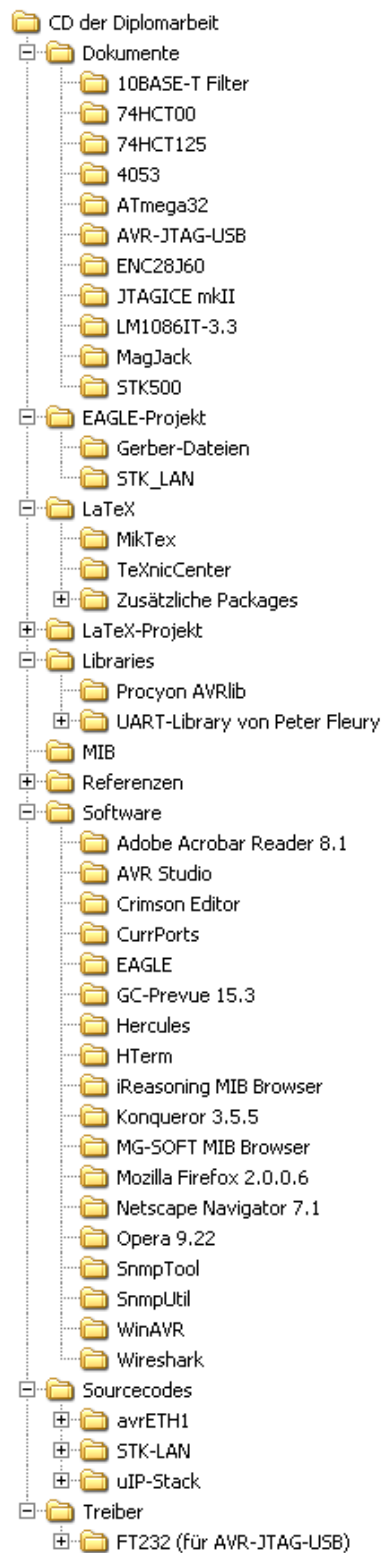


Abbildung L.1.: Inhalt der beiliegenden CD-ROM zu dieser Diplomarbeit

Quellenverzeichnis

- [1] *STK500-Schaltplan*, Februar 2001. http://www.avrfreaks.net/Tools/ToolFiles/115/STK500_Schematics.pdf.
- [2] Base64. Website, Juli 2007. <http://de.wikipedia.org/wiki/Base64>.
- [3] M. Allman, V. Paxson, and W. Stevens. *Congestion Control in IP/TCP Internetworks*, Januar 1984. <ftp://ftp.rfc-editor.org/in-notes/rfc896.txt>.
- [4] M. Allman, V. Paxson, and W. Stevens. *TCP Congestion Control*, April 1999. <ftp://ftp.rfc-editor.org/in-notes/rfc2581.txt>.
- [5] P. Almquist. *Type of Service in the Internet Protocol Suite*, Juli 1992. <ftp://ftp.rfc-editor.org/in-notes/rfc1349.txt>.
- [6] H. Alvestrand. *Content Language Headers*, Mai 2002. <ftp://ftp.rfc-editor.org/in-notes/rfc3282.txt>.
- [7] Juri Andruschenko. *Entwurf und Realisierung eines SNMP-Agenten in einer 16-Bit uController-Umgebung*. Fachhochschule Gelsenkirchen, 2005.
- [8] Atmel Corporation. *STK500 User Guide*, März 2003. http://www.atmel.com/dyn/resources/prod_documents/doc1925.pdf.
- [9] IEEE Registration Authority. *The public OUI listing*, August 2007. <http://standards.ieee.org/regauth/oui/oui.txt>.
- [10] Bel Fuse Inc. *Magjack Connector Modules - 10/100Base-TX*, 2007. <http://www.belfuse.com/Data/Datasheets/SI-40141.pdf>.
- [11] T. Berners-Lee, R. Fielding, and H. Frystyk. *Hypertext Transfer Protocol – HTTP/1.0*, Mai 1996. <ftp://ftp.rfc-editor.org/in-notes/rfc1945.txt>.

Quellenverzeichnis

- [12] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright. *Transport Layer Security (TLS) Extensions*, April 2006. <ftp://ftp.rfc-editor.org/in-notes/rfc4366.txt>.
- [13] U. Blumenthal and B. Wijnen. *User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)*, Dezember 2002. <ftp://ftp.rfc-editor.org/in-notes/rfc3414.txt>.
- [14] R. Braden. *Requirements for Internet Hosts – Application and Support*, Oktober 1989. <ftp://ftp.rfc-editor.org/in-notes/rfc1123.txt>.
- [15] R. Braden. *Requirements for Internet Hosts – Communication Layers*, Oktober 1989. <ftp://ftp.rfc-editor.org/in-notes/rfc1122.txt>.
- [16] R. Braden. *Extending TCP for Transactions – Concepts*, November 1992. <ftp://ftp.rfc-editor.org/in-notes/rfc1379.txt>.
- [17] CadSoft. *EAGLE Handbuch Version 4.1*, 2004. <ftp://ftp.cadsoft.de/eagle/program/4.16r2/manual-ger.pdf>.
- [18] J. Case, M. Fedor, M. Schoffstall, and J. Davin. *A Simple Network Management Protocol*, August 1988. <ftp://ftp.rfc-editor.org/in-notes/rfc1067.txt>.
- [19] J. Case, M. Fedor, M. Schoffstall, and J. Davin. *A Simple Network Management Protocol (SNMP)*, Mai 1990. <ftp://ftp.rfc-editor.org/in-notes/rfc1157.txt>.
- [20] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. *Introduction to Community-based SNMPv2*, Januar 1996. <ftp://ftp.rfc-editor.org/in-notes/rfc1901.txt>.
- [21] P. Chown. *Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)*, Juni 2002. <ftp://ftp.rfc-editor.org/in-notes/rfc3268.txt>.
- [22] Debian Community. Debian. Website, Mai 2007. <http://www.debian.org>.
- [23] Atmel Corporation. Atmel corporation. Website. <http://www.atmel.com>.
- [24] MG-SOFT Corporation. Mg-soft corporation. Website. <http://www.mg-soft.com>.
- [25] Netscape Communications Corporation. Netscape communications corporation. Website. <http://www.netscape.com>.
- [26] J. Davin, J. Case, M. Fedor, and M. Schoffstall. *A Simple Gateway Monitoring Protocol*, November 1987. <ftp://ftp.rfc-editor.org/in-notes/rfc1028.txt>.

Quellenverzeichnis

- [27] S. Deering and R. Hinden. *Internet Protocol, Version 6 (IPv6)*, Dezember 1998. <ftp://ftp.rfc-editor.org/in-notes/rfc2460.txt>.
- [28] T. Dierks and C. Allen. *The TLS Protocol Version 1.0*, Januar 1999. <ftp://ftp.rfc-editor.org/in-notes/rfc2246.txt>.
- [29] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.1*, April 2006. <ftp://ftp.rfc-editor.org/in-notes/rfc4346.txt>.
- [30] Adam Dunkels. The uip tcp/ip stack for embedded microcontrollers. Website, Juni 2007. <http://www.sics.se/~adam/uip/>.
- [31] Jens Dutine. *Entwurf und Implementierung eines Netzwerk-Management-Systems auf Basis von SNMP*. Universität Siegen, 2006. http://www.bs.informatik.uni-siegen.de/web/wismueller/arbeiten/2006_dutine.pdf.
- [32] Reichelt Elektronik e. Kfr. Reichelt elektronik e. kfr. Website. <http://www.reichelt.de>.
- [33] SEGOR electronics GmbH. Segor-electronics gmbh. Website. <http://www.segor.de>.
- [34] Kolja Engelmann. *SNMP Simple Network Management Protocol*. Universität Potsdam, Februar 2005. <http://fara.cs.uni-potsdam.de/~engelman/9.%20Semester/Internetprotokolle/Vortrag/pdf/SNMP%20Simple%20Network%20Management%20Protocol.pdf>.
- [35] Chaos Computer Club Cologne e.V. Ip-header. Website, Mai 2007. <http://koeln.ccc.de/archiv/drt/ip-header.html>.
- [36] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*, Juni 1999. <ftp://ftp.rfc-editor.org/in-notes/rfc2616.txt>.
- [37] Ross Finlayson, Timothy Mann, Jeffrey Mogul, and Marvin Theimer. *A Reverse Address Resolution Protocol*, Juni 1984. <ftp://ftp.rfc-editor.org/in-notes/rfc903.txt>.
- [38] Peter Fleury. Peter fleury’s home page. Website, Juli 2007. <http://homepage.hispeed.ch/peterfleury/>.
- [39] Free Software Foundation. Gnu general public license. Website. <http://www.fsf.org/licenses/licenses/gpl.html>.

Quellenverzeichnis

- [40] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. *HTTP Authentication: Basic and Digest Access Authentication*, Juni 1999. <ftp://ftp.rfc-editor.org/in-notes/rfc2617.txt>.
- [41] Peter Gagnon. The hypertext transfer protocol. Website, Mai 2007. http://www.doc.ic.ac.uk/~nd/surprise_97/journal/vol2/pcg1/.
- [42] TO-Consult GbR. To-consult gbr - was ist snmp? Website, Juli 2007. http://www.to-consult.de/html/nsm_snmp.htm.
- [43] Beck IPC GmbH. Beck ipc gmbh. Website. <http://www.beck-ipc.com>.
- [44] CadSoft Computer GmbH. Cadsoft online: Eagle layout editor. Website. <http://www.cadsoft.de>.
- [45] Wilke Technology GmbH. Wilke technology gmbh. Website. <http://www.wilke.de>.
- [46] GraphiCode. Graphiccode - software innovations for electronics manufacturing. Website. <http://www.graphicode.com>.
- [47] HW group s.r.o. Hw group. Website. <http://www.hw-group.com>.
- [48] HW group s.r.o. Hercules setup utility. Website, Juli 2007. <http://www.hw-group.com/download/sw/HerculesSetup.zip>.
- [49] HALO Electronics Inc. *10BASE-T SMD Isolation Module Selection Guide*, 2006. <http://www.haloelectronics.com/pdf/smd10baset.pdf>.
- [50] IANA. *CHARACTER SETS*. <http://www.iana.org/assignments/character-sets>.
- [51] IANA. Internet assigned numbers authority. Website, Mai 2007. <http://www.iana.org>.
- [52] IANA. *INTERNET CONTROL MESSAGE PROTOCOL*, Mai 2007. <http://www.iana.org/assignments/icmp-parameters>.
- [53] IANA. *Private Enterprise Number (PEN) Request Template*, Mai 2007. <http://www.iana.org/cgi-bin/enterprise.pl>.
- [54] IANA. *PRIVATE ENTERPRISE NUMBERS*, Mai 2007. <http://www.iana.org/assignments/enterprise-numbers>.
- [55] Internet Assigned Numbers Authority (IANA). *Special-Use IPv4 Addresses*, September 2002. <ftp://ftp.rfc-editor.org/in-notes/rfc3330.txt>.

Quellenverzeichnis

- [56] Bel Fuse Inc. Bel fuse inc. Website. <http://www.belfuse.com>.
- [57] Compass Laboratory Inc. Avr jtag ice2. Website, Juli 2007. http://www.compass-lab.com/STK_CAN/AVR_JTAG_ICE.htm.
- [58] Ethereal Inc. Ethereal: A network protocol analyzer. Website. <http://www.ethereal.com>.
- [59] HALO Electronics Inc. Halo electronics inc. Website. <http://www.haloelectronics.com>.
- [60] Microchip Technology Inc. Microchip technology inc. Website. <http://www.microchip.com>.
- [61] iReasoning Inc. ireasoning inc. Website. <http://www.ireasoning.com>.
- [62] Iso - international organization for standardization. Website. <http://www.iso.org>.
- [63] V. Jacobson, R. Braden, and D. Borman. *TCP Extensions for High Performance*, Mai 1992. <ftp://ftp.rfc-editor.org/in-notes/rfc1323.txt>.
- [64] Rainer Janssen and Wolfgang Schott. *SNMP - Konzepte, Verfahren, Plattformen*. DATACOM-Verlag, 1993.
- [65] S. Josefsson. *The Base16, Base32, and Base64 Data Encodings*, Juli 2003. <ftp://ftp.rfc-editor.org/in-notes/rfc3548.txt>.
- [66] S. Josefsson. *The Base16, Base32, and Base64 Data Encodings*, Oktober 2006. <ftp://ftp.rfc-editor.org/in-notes/rfc4648.txt>.
- [67] Ingyu Kang. Homepage of crimson editor. Website. <http://www.crimsoneditor.com>.
- [68] Christian Kauhaus. *TCP/IP Illustrated*, Dezember 2006. <http://www2.informatik.uni-jena.de/~ckauhaus/2006/tcpip/tcpip.pdf>.
- [69] Heinz Keller. Heinz keller. Website. <http://www.keller-elektronik.de>.
- [70] Carl Hanser Verlag GmbH & Co. KG. *Transport-Protokolle TCP und UDP*. http://files.hanser.de/hanser/docs/20040401_244515439-7998_3-446-21501-8.pdf.
- [71] Elektronikladen Mikrocomputer Giesler & Danne GmbH & Co. KG. Avr-jtag programmier und emulator-interface. Website. <http://elmicro.com/de/avrjtag.html>.

Quellenverzeichnis

- [72] R. Khare and S. Lawrence. *Upgrading to TLS Within HTTP/1.1*, Mai 2000. <ftp://ftp.rfc-editor.org/in-notes/rfc2817.txt>.
- [73] E. Kohler, M. Handley, and S. Floyd. *Datagram Congestion Control Protocol (DCCP)*, März 2006. <ftp://ftp.rfc-editor.org/in-notes/rfc4340.txt>.
- [74] D. Kristol and L. Montulli. *HTTP State Management Mechanism*, Februar 1997. <ftp://ftp.rfc-editor.org/in-notes/rfc2109.txt>.
- [75] D. Kristol and L. Montulli. *HTTP State Management Mechanism*, Oktober 2000. <ftp://ftp.rfc-editor.org/in-notes/rfc2965.txt>.
- [76] L-A. Larzon, M. Degermark, S. Pink, L-E. Jonsson, and G. Fairhurst. *The Lightweight User Datagram Protocol (UDP-Lite)*, Juli 2004. <ftp://ftp.rfc-editor.org/in-notes/rfc3828.txt>.
- [77] David Law. Ieee 802.3 csma/cd (ethernet). Website, Mai 2007. <http://www.ieee802.org/3/>.
- [78] Future Technology Devices International Ltd. Ftdi chip home page. Website. <http://www.ftdichip.com>.
- [79] Future Technology Devices International Ltd. Virtual com port drivers. Website. <http://www.ftdichip.com/Drivers/VCP.htm>.
- [80] OLIMEX Ltd. Olimex ltd. Website. <http://www.olimex.com>.
- [81] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. *TCP Selective Acknowledgment Options*, Oktober 1996. <ftp://ftp.rfc-editor.org/in-notes/rfc2018.txt>.
- [82] K. McCloghrie and J. Galvin. *Party MIB for version 2 of the Simple Network Management Protocol (SNMPv2)*, April 1993. <ftp://ftp.rfc-editor.org/in-notes/rfc1447.txt>.
- [83] K. McCloghrie, D. Perkins, and J. Schoenwaelder. *Conformance Statements for SMIPv2*, April 1999. <ftp://ftp.rfc-editor.org/in-notes/rfc2580.txt>.
- [84] K. McCloghrie, D. Perkins, and J. Schoenwaelder. *Structure of Management Information Version 2 (SMIPv2)*, April 1999. <ftp://ftp.rfc-editor.org/in-notes/rfc2578.txt>.
- [85] K. McCloghrie, D. Perkins, and J. Schoenwaelder. *Textual Conventions for SMIPv2*, April 1999. <ftp://ftp.rfc-editor.org/in-notes/rfc2579.txt>.

Quellenverzeichnis

- [86] K. McCloghrie and M. Rose. *Management Information Base for Network Management of TCP/IP-based internets*, Mai 1990. <ftp://ftp.rfc-editor.org/in-notes/rfc1156.txt>.
- [87] K. McCloghrie and M. Rose. *Management Information Base for Network Management of TCP/IP-based internets: MIB-II*, März 1991. <ftp://ftp.rfc-editor.org/in-notes/rfc1213.txt>.
- [88] A. Medvinsky and M. Hur. *Addition of Kerberos Cipher Suites to Transport Layer Security (TLS)*, Oktober 1999. <ftp://ftp.rfc-editor.org/in-notes/rfc2712.txt>.
- [89] Marcus Meyerhöfer. *Design und Implementierung eines Ethernet-Treibers und TCP/IP-Protokolls für das Java-Betriebssystem JX*. Friedrich-Alexander-Universität Erlangen-Nürnberg, Oktober 2000. <http://www4.informatik.uni-erlangen.de/Projects/JX/publications/SA-I4-2000-16-Meyerhoefer.ps.gz>.
- [90] Microchip Technology Inc. *ENC28J60 Data Sheet - Stand-Alone Ethernet Controller with SPI Interface*, 2006. <http://ww1.microchip.com/downloads/en/DeviceDoc/39662b.pdf>.
- [91] James D. Murray. Snmptool - simple network management protocol tool for win32. Website, August 1997. http://www.jklein.de/techniker_arbeit/tech_data/snmptool.exe.
- [92] J. Myers and M. Rose. *The Content-MD5 Header Field*, Oktober 1995. <ftp://ftp.rfc-editor.org/in-notes/rfc1864.txt>.
- [93] National Semiconductor Corporation. *LM1086*, 2005. <http://www.national.com/ds.cgi/LM/LM1086.pdf>.
- [94] NXP Semiconductors. *74HC/HCT125*, 1990. http://www.nxp.com/acrobat_download/datasheets/74HC_HCT125_CNV_2.pdf.
- [95] University of California. Berkeley software distribution. Website, Mai 2007. <http://www.bsd.org>.
- [96] Information Sciences Institute University of Southern California. *INTERNET PROTOCOL*, September 1981. <ftp://ftp.rfc-editor.org/in-notes/rfc791.txt>.
- [97] Information Sciences Institute University of Southern California. *TRANSMISSION CONTROL PROTOCOL*, September 1981. <ftp://ftp.rfc-editor.org/in-notes/rfc793.txt>.

Quellenverzeichnis

- [98] C. Partridge and R. Hinden. *Version 2 of the Reliable Data Protocol (RDP)*, April 1990. <ftp://ftp.rfc-editor.org/in-notes/rfc1151.txt>.
- [99] David C. Plummer. *An Ethernet Address Resolution Protocol*, November 1982. <ftp://ftp.rfc-editor.org/in-notes/rfc826.txt>.
- [100] J. Postel. *User Datagram Protocol*, August 1980. <ftp://ftp.rfc-editor.org/in-notes/rfc768.txt>.
- [101] J. Postel. *INTERNET CONTROL MESSAGE PROTOCOL*, September 1981. <ftp://ftp.rfc-editor.org/in-notes/rfc792.txt>.
- [102] J. Postel. *The TCP Maximum Segment Size and Related Topics*, November 1983. <ftp://ftp.rfc-editor.org/in-notes/rfc879.txt>.
- [103] Q print electronic GmbH. Q-print electronic gmbh. Website. <http://www.q-pcb.de>.
- [104] Ulrich Radig. Elektronik, mikroelektronik, computer - ulrichradig.de. Website, Juni 2007. <http://www.ulrichradig.de>.
- [105] E. Rescorla. *HTTP Over TLS*, Mai 2000. <ftp://ftp.rfc-editor.org/in-notes/rfc2818.txt>.
- [106] J. Reynolds and J. Postel. *ASSIGNED NUMBERS*, Oktober 1994. <ftp://ftp.rfc-editor.org/in-notes/rfc1700.txt>.
- [107] R. Rivest. *The MD5 Message-Digest Algorithm*, April 1992. <ftp://ftp.rfc-editor.org/in-notes/rfc1321.txt>.
- [108] Ben Rockwood. *The Net-SNMP Programming Guide*, November 2004. http://www.cuddletech.com/snmp_guide.pdf.
- [109] M. Rose. *A Convention for Defining Traps for use with the SNMP*, März 1991. <ftp://ftp.rfc-editor.org/in-notes/rfc1215.txt>.
- [110] M. Rose and K. McCloghrie. *Management Information Base for Network Management of TCP/IP-based internets*, August 1988. <ftp://ftp.rfc-editor.org/in-notes/rfc1066.txt>.
- [111] M. Rose and K. McCloghrie. *Structure and Identification of Management Information for TCP/IP-based internets*, August 1988. <ftp://ftp.rfc-editor.org/in-notes/rfc1065.txt>.

Quellenverzeichnis

- [112] M. Rose and K. McCloghrie. *Structure and Identification of Management Information for TCP/IP-based Internets*, Mai 1990. <ftp://ftp.rfc-editor.org/in-notes/rfc1155.txt>.
- [113] Marshall T. Rose. *The Simple Book*. Prentice Hall, 1991.
- [114] Heidrun Schmidt. Homepage von heidrun schmidt. Website, Juli 2007. <http://mitarbeiter.hs-heilbronn.de/~schdt/>.
- [115] Simon Schulz. avr und diverse andere projekte. Website, Juni 2007. <http://avr.auctionant.de>.
- [116] Pierrick Simie. Snmp version 3. Website, Mai 2007. <http://www.snmplink.org/SNMPv3.html>.
- [117] The simpleweb - tutorial slides in pdf format. Website, Juni 2007. <http://www.simpleweb.org/tutorials/slides.html>.
- [118] IEEE Communications Society. Im 2001 distinguished experts panel. Website, Juli 2007. <http://www.comsoc.org/confs/im/2001/Distinguished.html>.
- [119] Nir Sofer. Currports: View opened tcp/ip ports / connections in windows. Website. <http://www.nirsoft.net/utils/cports.html>.
- [120] Nir Sofer. Nirsoft. Website. <http://www.nirsoft.net>.
- [121] Pascal Stang. Procyon avrlib - c-language function library for atmel avr processors. Website, September 2005. <http://hubbard.engr.scu.edu/avr/avrlib/>.
- [122] Microcontroller Starterkits. Microcontroller starterkits. Website. <http://www.microcontroller-starterkits.de>.
- [123] Johannes Stoll. Kommunikation im osi-7-schichten-modell. Website, Oktober 2006. http://www.godofbytes.de/images/osi_iso_7layer.png.
- [124] Jörg Storch. Hshn - dipl.-ing. (fh) jörg storch. Website, Juli 2007. <http://www.hs-heilbronn.de/Members/joergstorch>.
- [125] CACE Technologies. Cace technologies. Website. <http://www.cacetech.com>.
- [126] Unbekannt. Arp und routing. Website, September 2003. http://upload.wikimedia.org/wikipedia/de/f/fd/ARP_und_Routing.png.

Quellenverzeichnis

- [127] TEIA AG Internet Akademie und Lehrbuch Verlag. Konzepte der internet-technik. Website, Mai 2007. <http://www.teialehrbuch.de/KIT/16216-Transmission-Control-Protocol.html>.
- [128] Forschungs und Transferzentrum Leipzig e.V. easytoweb. Website. <http://www.easytoweb.net>.
- [129] International Telecommunication Union. The itu telecommunication standardization sector (itu-t). Website. <http://www.itu.int/ITU-T>.
- [130] INTERNATIONAL TELECOMMUNICATION UNION. *OSI networking and system aspects - Abstract Syntax Notation One (ASN.1)*, Juli 2002. <http://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf>.
- [131] UNISONIC TECHNOLOGIES CO.,LTD. *4053 - ANALOG MULTIPLEXERS/DEMUL-TIPLEXERS*, 2007. http://www.tranzistoare.ro/datasheets2/14/149194_2.pdf.
- [132] David Velten, Robert Hinden, and Jack Sax. *Reliable Data Protocol*, Juli 1984. <ftp://ftp.rfc-editor.org/in-notes/rfc908.txt>.
- [133] W3C. *The Original HTTP as defined in 1991*. <http://www.w3.org/Protocols/HTTP/AsImplemented.html>.
- [134] W3C. *Progress on HTTP-NG*. <http://www.w3.org/Protocols/HTTP-NG/http-ng-status.html>.
- [135] G. Waters. *User-based Security Model for SNMPv2*, Februar 1996. <ftp://ftp.rfc-editor.org/in-notes/rfc1910.txt>.
- [136] Ethernet-ii-frameformat aus ieee 802.3, inklusive vlan tag aus ieee 802.1q. Website, Mai 2007. <http://de.wikipedia.org/wiki/Bild:Etherframe.png>.
- [137] Url. Website, Mai 2007. <http://upload.wikimedia.org/wikipedia/commons/f/f7/URL.gif>.
- [138] Koordinierte weltzeit. Website, Mai 2007. http://de.wikipedia.org/wiki/Koordinierte_Weltzeit.
- [139] Winavr : Avr-gcc for windows. Website. <http://winavr.sourceforge.net>.
- [140] Jan Winkler. Html world. Website, Mai 2007. http://www.html-world.de/program/http_3.php.

Quellenverzeichnis

- [141] Wireshark-Community. Wireshark: the world's most popular network protocol analyzer. Website. <http://www.wireshark.org>.
- [142] Defining traps. Website, Juni 2007. <http://www.zvon.org/tmRFC/RFC1215/Output/chapter2.html>.

Quellenverzeichnis

Abkürzungsverzeichnis

ACK Acknowledgement

ADC Analog-Digital-Converter

ARM Advanced RISC Machines

ARP Address Resolution Protocol

ASN.1 Abstract Syntax Notation One

BER Basic Encoding Rules

BSD Berkeley Software Distribution

CAN Controller Area Network

CCITT Comité Consultatif International Télégraphique et Téléphonique

CPU Central Processing Unit

CR Carriage Return

CRC Cyclic Redundancy Check

CSMA/CD Carrier Sense Multiple Access with Collision Detection

DCCP Datagram Congestion Control Protocol

DIL Dual In-Line

EAGLE Einfach Anzuwendender Grafischer Layout Editor

ETag Entity Tag

Abkürzungsverzeichnis

FIN Finish

FTDI Future Technology Devices International Ltd

FTP File Transfer Protocol

GCC GNU Compiler Collection

GND Ground

GPL GNU General Public License

HID Human Interface Device

HTTP Hypertext Transfer Protocol

HTTDP Hypertext Transfer Protocol Daemon

HTTP-NG Hypertext Transfer Protocol Next Generation

HTTPS Hypertext Transfer Protocol Secure

IANA Internet Assigned Numbers Authority

IC Integrated Circuit

ICMP Internet Control Message Protocol

ID Identifikationsnummer

IEEE Institute of Electrical and Electronics Engineers

I2C Inter-Integrated Circuit

IP Internet Protocol

ISN Initial Sequence Number

ISO International Organization for Standardization

ISP In-System-Programming

ITU International Telecommunication Union

ITU-T ITU Telecommunication Standardization Sector

Abkürzungsverzeichnis

JTAG Joint Test Action Group

KB Kilobyte

kHz Kilohertz

LAN Local Area Network

LDAP Lightweight Directory Access Protocol

LED Light Emitting Diode

LSB Least Significant Bit

MAC Medium Access Control

MAC Medium Access Controller

MHz Megahertz

MIB-II Managed Information Base 2

MNE Managed Network Entity

MO Managed Object

MSB Most Significant Bit

MSL Maximum Segment Lifetime

MSS Maximum Segment Size

MTU Maximum Transfer Unit

NAND Not AND

NMS Network-Management-Station

OID Object Identifier

OSI Open System Interconnection

PC Personal Computer

Abkürzungsverzeichnis

PDU Personal Data Unit

RAM Random Access Memory

RARP Reverse Address Resolution Protocol

RCF Router Congestion Feedback

RDP Reliable Datagram Protocol

RFC Request for Comments

RISC Reduced Instruction Set Computing

RST Reset

RTT Round Trip Time

RXD Received Data

SFD Start Frame Delimiter

SMI Structure of Management Information

SMTP Simple Mail Transfer Protocol

SNMP Simple Network Management Protocol

SPI Serial Peripheral Interface

SRAM Static Random Access Memory

STK Starterkit

SYN Synchronisation

TCP Transmission Control Protocol

TXD Transmitted Data

UDP User Datagram Protocol

URL Uniform Resource Identifier

USART Universal Synchronous and Asynchronous Receiver and Transmitter

Abkürzungsverzeichnis

UTC Universal Time Coordinated

VLAN Virtual Local Area Network

VoIP Voice over IP

VTarget Target-Spannung

WWW World Wide Web

Abkürzungsverzeichnis

Glossar

1:1-Verbindung Eine Verbindung zwischen zwei Steckerverbindern, bei welcher die einzelnen Adern des Kabels an beiden Seiten in der selben Reihenfolge angeschlossen sind

Abstract Syntax Notation One Beschreibungssprache zur Definition von strukturierten Datentypen

Address Resolution Protocol Hilfsprotokoll zum Finden der zu einer *IP-Adresse* gehörenden *MAC-Adresse*

Advanced RISC Machines Leistungsfähige 32-Bit-Mikroprozessoren, welche mit dem *Reduced Instruction Set Computing* arbeiten

American Standard Code for Information Interchange Zeichenkodierung

Analog-Digital-Converter Bauteil zur Wandlung analoger in digitale Signale

Array Datentyp in der Informatik, welcher aus Daten des gleichen Typs zusammengesetzt ist

Assembler Maschinennahe Programmiersprache

Asynchrones Protokoll Es können gleichzeitig mehrere Anfragen an die Gegenstelle gesendet werden. Es ist der Gegenstelle überlassen in welcher Reihenfolge die Anfragen bearbeitet werden. Der Sender hat selbst Sorge dafür zu tragen, dass die Antwortpakete wieder korrekt zugeordnet werden können

AVR Familie von *Reduced Instruction Set Computing (RISC)-Mikrocontrollern*

Ball Grid Array Gehäusebauform für *ICs*

Basic Encoding Rules Kodierungsschema von *ASN.1-Datentypen*

Baudrate Einheit der Symbolrate einer Kommunikationsverbindung

Glossar

Berkeley Software Distribution Version des Betriebssystems *Unix*

Bidirektional Daten können in beide Richtungen übertragen werden

Big Endian Höchstwertigstes *Byte* wird zuerst übertragen

Binärdatei Alle möglichen Dateiformate

Bit Kleinste Speichereinheit in digitaler Datenverarbeitung, welche die Werte null und eins annehmen kann

Body Körper einer Nachricht

Bootvorgang Starten eines Systems

Breakpoint Punkt in einem Programm, bei welchem die Programmausführung während des *Debuggens* angehalten wird

Broadcast Wie *Multicast*, jedoch werden dabei alle sich im Netzwerk befindenden Geräte angesprochen

Buffer Zwischenspeicher

Bus Elektrische Verbindungen zwischen mehr als zwei Systemen zum Austausch von Daten

Byte Paket aus acht *Bits*

Cache Schneller Zwischenspeicher

Carriage Return Wagenrücklauf in der Textverarbeitung

Carrier Sense Multiple Access with Collision Detection Verfahren zum Steuern des Buszugriffs

Central Processing Unit Zentraler Prozessor eines Systems

Checksumme Zusätzliche Angabe im Datenpaket zur Überprüfung der korrekten Übertragung der Nutzdaten

Chipselect Elektrisches Signal zur Auswahl eines Bausteins

Comité Consultatif International Télégraphique et Téléphonique Technisches Komitee, welches sich mit Normen der Telekommunikation befasst

Community String Bezeichnung des Gruppennamens zu welchem ein *SNMP-Teilnehmer* gehört

Glossar

- Compiler** Programm, welches einen *Quellcode* in eine andere Form umwandelt
- Computer Aided Manufacturing** Herstellung mit Hilfe eines Computers
- Congestion Window** Überlastfenster, welches angibt, wie viele unbestätigte Pakete zu einem Zeitpunkt unterwegs sein dürfen
- Controller** Elektronisches Bauteil zum Steuern verschiedener Vorgänge
- Controller Area Network** Asynchrones, serielles *Bussystem*, welches speziell zur Vernetzung von Steuergeräten in Kraftfahrzeugen entwickelt wurde
- Cookie** Kleine Textdatei auf *Clients*, in welcher ein *Server* Informationen ablegen kann
- Crossover-Kabel** Kabel mit gekreuzten Signalleitungen, womit die *TX-Pins* des Senders mit den *RX-Pins* des Empfängers verbunden werden
- Cycle Counter** Anzeige, wie viele Maschinenzyklen das Zielsystem abgearbeitet hat
- Cyclic Redundancy Check** Verfahren zur Bestimmung einer Prüfsumme
- Daemon** Im Hintergrund laufender Dienst, welcher spezielle Funktionen zur Verfügung stellt
- Datagramm** Datenpaket in einem Netzwerk
- Debuggen** Auffinden, Diagnostizieren und Beheben von Fehlern in der Hard- und Software
- debugWire** Serielle Schnittstelle zum *Debuggen* von Mikrocontrollern
- Department of Defense** Verteidigungsministerium der Vereinigten Staaten
- Direct Memory Access** Hardwaredesign, bei welchem Systemkomponenten ohne Umweg über die *CPU* direkt auf den Speicher zugreifen können
- Domain** Eindeutige Adressangabe in einem Netzwerk
- Download** Herunterladen von Dateien von einem *Server*
- Drei-Wege-Handshake** Verfahren zum Aufbau einer Verbindung bei *TCP*
- Dual In-Line** Gehäusebauform für *ICs*
- E-Mail** Elektronische Post, die in Computernetzwerken ausgetauscht wird
- Einerkomplement-Arithmetik** Arithmetische Operation, bei welcher alle *Bits* invertiert werden

Glossar

Einfach Anzuwendender Grafischer Layout Editor Weit verbreiteter *Schaltplan-* und *Lay-outeditor*

Electrically Erasable Programmable Read Only Memory Speicher, welcher elektrisch beschrieben und gelöscht werden kann und seinen Inhalt auch nach Abschalten der Versorgungsspannung behält

Embedded Webserver Kleiner, in einem System integrierter, *Webserver*

Ethernet Weit verbreitete Netzwerktechnologie, welche die *Netzzugangsschicht* im *TCP/IP-Referenzmodell* repräsentiert

Exterior Gateway Protocol Protokoll, welches dazu dient, Erreichbarkeitsinformationen zwischen autonomen Systemen auszutauschen

File Transfer Protocol Spezielles Netzwerkprotokoll zur Datenübertragung

Firmware Festes Programm in einem Gerät, welches nur selten bei Updates aktualisiert wird

Flag Konfigurationsbit

Flash Genaue Bezeichnung: *Flash-EEPROMs*. Digitale Speicher, welche sich im Vergleich zu *EEPROMs* nur blockweise löschen lassen

Flusssteuerung Verfahren zur Steuerung der Netzwerkauslastung

Frame Sich auf Schicht zwei des *OSI-Modells* befindendes *Datagramm*

Freeware Software, welche vom Urheber zur kostenlosen Nutzung bereitgestellt wird

Fuse Einzelnes *Bit*, welches zur Konfiguration des Mikrocontrollers verwendet werden kann

Gateway Netzwerkgerät zum Verbinden verschiedener Netzwerktechnologien

Halbduplex Verbindung, bei der die Daten nicht gleichzeitig in beide Richtungen übertragen werden können

Handshake Verfahren zur Datenflusssteuerung

Header Kopf eines Datenpakets mit Informationen zur weiteren Verarbeitung

Hexadezimal Zahlensystem mit der Basis 16, welches die Zahlen 0 bis 9 und die Buchstaben A bis F verwendet

Glossar

High-Voltage-Programming Spezieller Programmiermodus für *AVR-Mikrocontroller*

Hop Weiterleitung von einem Netzknoten zum nächsten

Hop-Count Anzahl der Netzknoten, welche ein Datenpaket passiert hat

Host Netzwerkgerät, welches Dienste zur Verfügung stellt

Hub Einfacher Verteiler in einem Netzwerk

Human Interface Device Schnittstelle zur Kommunikation mit dem Benutzer

Hyperlink Verknüpfung von einer Seite auf eine andere in einem *Hypertextsystem*

HyperText Markup Language Textbasierte Sprache zur Darstellung von Texten, Grafiken und Hyperlinks

Hypertext Transfer Protocol Protokoll zur Übertragung von *Webseiten* und anderen Daten

Hypertext Transfer Protocol Daemon Dienst, der im Hintergrund läuft und Daten per *HTTP* zur Verfügung stellt

Hypertext Transfer Protocol Next Generation Neuste Version von *HTTP*, welche sich noch in der Entwicklung befindet

Hypertext Transfer Protocol Secure Erweitertes *HTTP* mit Möglichkeiten zur Datenverschlüsselung und Authentifizierung

Hypertextsystem Vernetzung von Objekten durch *Hyperlinks*

In-System-Programming Programmieren einer Komponente im Zielsystem

Initial Sequence Number Startsequenznummer einer *TCP-Verbindung*

Institute of Electrical and Electronics Engineers Weltweiter Zusammenschluss von Elektrotechnik- und Informatikingenieuren mit Gremien für Normung

Integer Ganzzahliger Datentyp in der Informatik

Integrated Circuit Integrierter Schaltkreis

Inter-Integrated Circuit Serieller *Datenbus*, welcher speziell zur Verwendung in Geräten der Unterhaltungselektronik entwickelt wurde

Glossar

International Electrotechnical Commission Internationales Gremium für Normung in den Bereichen Elektrotechnik und Elektronik

International Organization for Standardization Internationale Normungsorganisation für alle Bereiche außer Elektrik, Elektronik und Telekommunikation

International Telecommunication Union Unterorganisation der Vereinten Nationen, die sich offiziell und weltweit mit technischen Aspekten der Telekommunikation beschäftigt

Internet Weltweiter Verbund von Netzwerken

Internet Assigned Numbers Authority Organisation, welche die Vergabe von *IP-Adressen*, *Top-Level-Domains* und *IP-Protokollnummern* sowie die Zuordnung der *Haupt-Ports* 0-1023 regelt

Internet Control Message Protocol Protokoll zur Übertragung von Statusmeldungen über das *Internet Protocol*

Internet Protocol Netzwerkprotokoll zur Realisierung der *Internetschicht* des *TCP/IP-Referenzmodells*

Interrupt Von der Hardware angeforderte Programmunterbrechung zum Ausführen einer Service-routine

Intranet Kleine Form des *Internets*, welches innerhalb einer Organisation verwendet wird

ITU Telecommunication Standardization Sector Abteilung der *ITU*, die sich mit technischen Normen, Standards und Empfehlungen für alle Gebiete der Telekommunikation auseinandersetzt

Joint Test Action Group Schnittstelle zum Programmieren und *Debuggen* von Systemen

Jumper Kleine Steckbrücke, mit deren Hilfe zwei *Pins* verbunden werden können

Kilobyte 1000 *Bytes* (10^3 *Bytes*), häufig jedoch auch für 1024 *Bytes* verwendet, obwohl hier korrekterweise die Bezeichnung *Kibibyte* (2^{10} *Bytes*) verwendet werden müsste

Konfiguration Einstellen von Parametern

Layouteditor Programm zum Erstellen von Platinenlayouts

Least Significant Bit Niederwertigstes *Bit* in einem *Byte*

Glossar

Lexikografie Das Erstellen von Wörterbüchern, also alphabetisch geordneten Nachschlagewerken

Light Emitting Diode Optoelektronisches Bauteil, welches Strahlung aussenden kann

Lightweight Directory Access Protocol Protokoll, welches Funktionen zur Abfrage und Modifikation von Informationen eines Verzeichnisdienstes über ein Netzwerk zur Verfügung stellt

Line Feed Zeilenumbruch in der Textverarbeitung

Link siehe: *Hyperlink*

Liquid Crystal Display Flüssigkristallanzeige, welche durch die Polarisierung von Licht alphanumerische Zeichen und/oder Grafiken darstellen kann

Little Endian Niederwertigstes *Byte* wird zuerst übertragen

Local Area Network Lokales Netzwerk

MagJack Eingetragenes Warenzeichen der Bel Fuse Inc. (vergleichbare Produkte sind unter den Bezeichnungen *FastJack* und *PulseJack* zu finden)

Makro Teil eines *Quellcodes*, welcher beim *Kompilieren* automatisch durch einen anderen Code ersetzt wird

Man-In-The-Middle Zwischenschalten eines Computers zwischen zwei andere Netzwerkteilnehmer, um den Datenverkehr abzuhören und zu manipulieren

Managed Information Base 2 Zweite Version der Baumstruktur zur Verwaltung der *Managed Objects*

Managed Network Entity Über *SNMP* verwaltetes System

Managed Node Durch *SNMP* verwalteter Netzwerkteilnehmer

Managed Object Variable, welche über *SNMP* verwaltet wird

Master In Slave Out Datenleitung des *SPI-Bus*, welche Daten vom Slave zum Master überträgt

Master Out Slave In Datenleitung des *SPI-Bus*, welche Daten vom Master zum Slave überträgt

Master-Slave-Prinzip Verfahren zur Steuerung der Datenübertragung

Maximum Segment Lifetime Maximale Lebensdauer eines Datensegments in einem Netzwerk

Glossar

Maximum Segment Size Maximale Anzahl von Bytes, die in einem *TCP-Frame* übertragen werden können

Maximum Transfer Unit Angabe der maximalen Größe des Nutzdatenpakets einer bestimmten Netzwerktechnologie

Medium Access Control Eindeutige Hardwareadresse zur Identifikation eines Geräts in einem Netzwerk

Medium Access Controller Bildet Schicht zwei des *OSI-Referenzmodells* in Hardware nach

Mikrocontrollerprogramm Programm für den Mikrocontroller in dieser Diplomarbeit das bei fast jeder Anwendung für Test-/Lehrzwecke modifiziert wird.

Modularbuchse Bauform für einen Steckverbinder

Modularstecker Bauform für einen Steckverbinder

Most Significant Bit Höchstwertigstes *Bit* in einem *Byte*

Multi-Homed Server Betreiben mehrerer *Server* unter einer einzigen *IP-Adresse*

Multicast Punkt-zu-Mehrpunkt-Verbindung

Multiplexer Schaltung, mit der aus einer gewissen Anzahl von Eingangssignalen eines ausgewählt werden kann

Network Element Netzwerkteilnehmer

Network-Management-Station System, auf welchem der *SNMP-Manager* läuft

Nullmodemkabel Kabel zum Simulieren einer Modemverbindung

Object Identifier Adresse, welche ein Objekt eindeutig identifiziert

Oktett siehe: *Byte*

Open Source Software, welche zusammen mit dem Quelltext vom Urheber zur kostenlosen Nutzung bereitgestellt wird

Open System Interconnection Genormtes Schichtenmodell, nach welchem Protokolle implementiert werden sollen

Glossar

Paddingbyte Füllbytes zum Erreichen einer vorgeschriebenen Mindestlänge eines Datagramms

Parsen Analysieren und Verarbeiten von Daten

Patchkabel Kabel mit einfacher *1:1-Verbindung* der beiden *8P8C-Modularstecker*

Personal Data Unit Datenbereich in einem *SNMP-Paket*

PHP: Hypertext Preprocessor Skriptsprache, welche vorwiegend zur Erstellung von dynamischen *Webseiten* verwendet wird. Ursprünglich stand die Abkürzung *PHP* für *Personal Home Page Tools*

PHY Schicht eins im *OSI-Referenzmodell*. Der physikalische *Transceiver* in elektronischen Systemen

Ping Netzwerkfunktion zur Feststellung, ob ein gewünschter Teilnehmer im Netzwerk vorhanden ist

Pointer Zeiger auf einen Speicherbereich

Polling Zyklisches Abrufen von Informationen

Proxy Zwischen *Client* und *Server* geschalteter Netzwerkteilnehmer, welcher Daten zwischenspeichern kann

Random Access Memory Speicher, welcher seinen Inhalt nach Abschalten der Versorgungsspannung verliert

RC-Oszillator Elektronische Schaltung, welche mit Hilfe von Widerstand und Kondensator eine Schwingung erzeugt

Registry Konfigurationsdatenbank von *Microsoft Windows*

Request for Comments Ansammlung von technischen Dokumenten zu Internetprotokollen

Reverse Address Resolution Protocol Protokoll zur Auflösung von zugehörigen *IP-Adressen* zu *MAC-Adressen*

Ringbuffer In Software aufgebaute Warteschlange

Root Wurzel einer Hierarchie

Round Trip Time Zeit zwischen dem Absenden eines Datenpakets und dem Empfangen einer Bestätigung

Glossar

- Router** Netzwerkgerät, welches sich mit der Wegevermittlung von Paketen beschäftigt
- Router Congestion Feedback** Sammlung von Informationen durch die *Router* zur Bestimmung der Netzlast
- Routing** Festlegung von Wegen für die Datenpakete in einem Netzwerk
- RS232** Standard für eine serielle Schnittstelle
- Serial Clock** Datenleitung des *SPI-Bus*, die das Taktsignal überträgt
- Serial Peripheral Interface** Sehr weit verbreitete, synchrone, serielle Schnittstelle, welche nach dem *Master-Slave-Prinzip* arbeitet
- Servicepack** Softwarepaket mit verschiedenen Aktualisierungen für ein Produkt
- Shareware** Zum Test frei verfügbare Software, welche nach dem Testzeitraum gekauft werden soll
- Simple Mail Transfer Protocol** Netzwerkprotokoll zum Austausch von E-Mails in Computernetzwerken
- Simple Network Management Protocol** Netzwerkprotokoll zur Verwaltung von Geräten von einer zentralen Stelle aus
- Sliding Window** Verfahren zur Flusssteuerung
- Small Outline** Gehäusebauform für *ICs*
- Socket** Virtuelle Schnittstelle auf einem System, welche aus der *IP-Adresse* und einem *Port* besteht
- Spoofing** Versuch, in einem Netzwerk eine andere Identität vorzutäuschen
- Start Frame Delimiter** Zeichen nach der Präambel eines *Ethernetframes* zum Anzeigen der Stelle, an welcher der eigentliche *Ethernetframe* beginnt
- Static Random Access Memory** Flüchtiger Speicherbaustein, bei welchem die Informationen durch bistabile Kippstufen gespeichert werden
- Stream** Verbindung, bei welcher die Daten auf der einen Seite gesendet und auf der anderen Seite sofort wiedergegeben werden
- String** Zeichenkette, bestehend aus alphanumerischen Zeichen und einigen Sonderzeichen
- Structure of Management Information** Auf *ASN.1* basierende Beschreibungssprache

Glossar

Symmetrisches Protokoll Anfrage- und Antwort-Pakete sind nach dem gleichen Schema aufgebaut

Target-Spannung Versorgungsspannung des Mikrocontrollers im Zielsystem

TeletypeNetwork Netzwerkprotokoll für Terminalanwendungen

Telnet Stellt eine einfache Terminalverbindung zur Verfügung

Template Vorlage, in welche noch Daten eingefügt werden können

Time-To-Live Maximale Lebensdauer eines *TCP-Pakets*

Timestamp Zeitstempel, um einem Ereignis eine bestimmte Zeit zuzuweisen

Top-Level-Domain Bezeichnung der obersten Ebene bei der hierarchischen Namensauflösung von Netzwerknamen

Traceroute Hilfe zur Ermittlung des Weges, den die Pakete in einem Netzwerk nehmen

Traffic In einem bestimmten Zeitraum übertragenes Datenvolumen

Transceiver Setzt sich aus den Begriffen *Transmitter* und *Receiver* zusammen und bezeichnet eine Kombination aus Sender und Empfänger

Transmission Control Protocol Verbindungsorientiertes Netzwerkprotokoll, welches eine Verbindung zwischen zwei Endpunkten herstellt

Transport Layer Security Verschlüsselungssystem für Datenübertragungen

Triggern Auslösen eines Ereignisses

Tristate Digitale Ausgänge, welche neben den Pegeln *High* und *Low* einen dritten hochohmigen Zustand annehmen können

Unicast Punkt-zu-Punkt-Verbindung

Uniform Resource Identifier Heutige Bezeichnung für *URL*, nachdem es ursprünglich als *Universal Resource Identifier* definiert war

Universal Asynchronous Receiver and Transmitter Asynchrone serielle Schnittstelle

Universal Serial Bus Universelle serielle Schnittstelle zur Verbindung eines Computers mit der Peripherie

Glossar

- Universal Synchronous and Asynchronous Receiver and Transmitter** Synchrone/asynchrone serielle Schnittstelle
- Universal Time Coordinated** Weit verbreitete Angabe der Weltzeit
- User Datagram Protocol** Einfaches, verbindungsloses Netzwerkprotokoll
- Verbindungsorientierte Übertragung** Eine Ende-zu-Ende Verbindung, bei welcher vor der Übertragung zuerst eine Übertragungsstrecke aufgebaut werden muss
- Virtual Local Area Network** Virtuelles Netz innerhalb eines physikalischen Netzes
- Voice over IP** Telefonieren über Computernetzwerke mit Hilfe spezieller Protokolle
- Vollduplex** Verbindung, bei der die Daten gleichzeitig in beide Richtungen übertragen werden können
- Webbrowser** Spezielles Programm zum Betrachten von *Websites*
- Webseite** siehe: *Website*
- Website** Seite in einem *Hypertextsystem*
- WinPcap** *Open Source* Programmbibliothek, welche unter Windows hardwarenahen Zugriff auf Netzwerkkarten ermöglicht
- World Wide Web** Über das *Internet* abrufbares *Hypertextsystem*
- World Wide Web Consortium** Gremium, welches sich mit den Normen des *World Wide Web* befasst
- Wrapper** Entwurfsmuster zur Anpassung von Schnittstellen
- Zigzag Inline Package** Dateiformat zum komprimierten Archivieren von Dateien