Marco Anetzberger

Entwicklung eines Netzwerk Gateways

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Informations- und Elektrotechnik
Studienrichtung Informationstechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Professor Dr. Franz Schubert
Zweitgutachter: Professor Dr. Jürgen Reichardt

Abgegeben am 06.12.2007

Marco Anetzberger

Thema der Diplomarbeit

Entwicklung eines Netzwerk Gateways

Stichworte

Netzwerk Gateway, Coldfire, KUHSE TeleControl System, serielle Schnittstelle, EIA-232, Ethernet, EIA-485, MPI

Kurzzusammenfassung

Entwicklung eines Netzwerk-Gateways für die KUHSE TeleControl Unterstation und weitere KUHSE Geräte sowie zur Einbindung von Fremdsteuerungen in ein TeleControl System. Die Datenkopplung zu den Steuerungen erfolgt seriell über EIA-232 per Kabel oder Lichtwellenleiter bzw. über EIA-485/MPI per Kabel.

Marco Anetzberger

Title of the paper

Development of a Network Gateway

Keywords

Network Gateway, Coldfire, KUHSE TeleControl System, serial Port, EIA-232, Ethernet, EIA-485, MPI

Abstract

Development of a Network Gateway for the KUHSE TeleControl substation and other KUHSE devices as well as for integration of external process control systems into a TeleControl system. The data connection to these systems is made via serial EIA-232 connection using cable or optical fiber or EIA-485/MPI connection via cable.

Inhalt

1 Einleitung [1, 2, 3]

Grundlage der Entwicklung des Netzwerk Gateways ist das Fernüberwachungssystem TeleControl der Firma Alfred KUHSE GmbH. Über dieses System werden Anlagen zur Energieerzeugung, z.B. Blockheizkraftwerke (BHKW), von der Ferne aus überwacht und auch gesteuert.
Die Datenverbindungen von den einzelnen Anlagen zum zentralen TeleControl Server werden dabei bisher über eine Einwahlverbindung per Analog-, ISDN- oder GSM-Modem hergestellt. Diese Verbindungsarten sind für eine permanente Verbindung zur Anlage schon aus Kostengründen nicht optimal. Durch die aktuellen Herausforderungen auf dem Energiemarkt wird allerdings eine permanente Verbindung benötigt.

Der immer stärker dezentralisierte Energiemarkt, besonders in Deutschland und Europa, erfordert eine umfassende und ständige Übersicht und Kontrolle über die Energiebilanz. Jeder Energie-Erzeuger muss über den aktuellen Status seiner Werke informiert sein und muss bei Veränderungen in der Energiebilanz schnell reagieren und steuernd eingreifen können. Besonders für den Handel an der Leipziger Strombörse und die Bereitstellung von Minutenreserven (Sekundärregelung) ist dies unverzichtbar.
Ebenso wird eine permanente Verbindung an eine zentrale Leitstelle für die Realisierung von sog. virtuellen Kraftwerken benötigt. Hier werden mehrere Kleinst-Erzeuger (ab 1 KW Erzeugung) nach außen zu einem Kraftwerk mit entsprechend hoher Leistung zusammengefasst.

Für die Datenverbindung bedeuten diese Herausforderungen, dass eine zuverlässige und gleichzeitig günstige Realisierung für eine Dauerverbindung angeboten werden muss. Hier bietet sich die Nutzung von Ethernet Netzwerken und damit die Nutzung einer DSL Internet-Verbindung an. DSL Anschlüsse sind inzwischen nahezu überall verfügbar, die Verbindungen sind stabiler als Modem-Einwahlverbindungen und die entsprechenden Daten-Flatrates sind günstig erhältlich.
Um eine DSL Verbindung zum Datenaustausch in einem KUHSE TeleControl nutzen zu können, wird eine Schnittstelle zum Ethernet Netzwerk in Form eines Gateway Moduls benötigt.

2 Allgemeine Grundlage – KUHSE TeleControl [4]

Das KUHSE TeleControl (TC) System besteht aus folgenden drei Hauptkomponenten:

- TC Unterstation
- TC Master
- TC Client

Die Unterstation ist ein KUHSE 68000 SPS System, das Betriebsdaten der angeschlossenen Anlage sammelt und auf Anfrage weiterleitet. Hierfür ist eine anlagenspezifische Datenstruktur hinterlegt. Außerdem werden bestimmte Daten (Trends) und Ereignisse (Protokolle) in Echtzeit aufgezeichnet und gespeichert.

Der TeleControl Master ist ein Server-PC, auf dem die TeleControl Master Software des KUHSE Data Publisher Softwarepakets läuft. Der Master stellt zyklisch oder auf Anfrage eine direkte Einwahlverbindung zur Unterstation per Analog-, ISDN- oder GSM-Modem her. Über diese Verbindung werden Anlagendaten, Trends und Protokolle übermittelt.
Ausnahme bildet hier die UDP Unterstation, die lediglich per Anruf zur Verbindung aufgefordert wird und sich daraufhin über das Modem mit dem Internet verbindet. So wird eine Datenverbindung mit dem Master über das Internet ermöglicht.

Der Master speichert die Trends und Protokolle auf Festplatte ab, da diese nur für einen begrenzten Zeitraum in der Unterstation zwischengespeichert werden können.
Der TeleControl Master ist eine von KUHSE angebotene Dienstleistung und ist für den Benutzer (Anwender) über das Internet erreichbar. Diese Verbindung ist durch eine Firewall und eine Zugriffskontrolle gesichert. (siehe Bild 1)

Der Benutzer kann über eine Internetverbindung und der Data Publisher TeleControl Client Software aktuelle Anlagendaten einsehen und Änderungen an Parametern durchführen. Außerdem können Trend- und Protokolldaten vom Master abgerufen und lokal gespeichert werden. Diese Daten können mit entsprechenden Auswertungs-Programmen offline betrachtet bzw. analysiert werden.

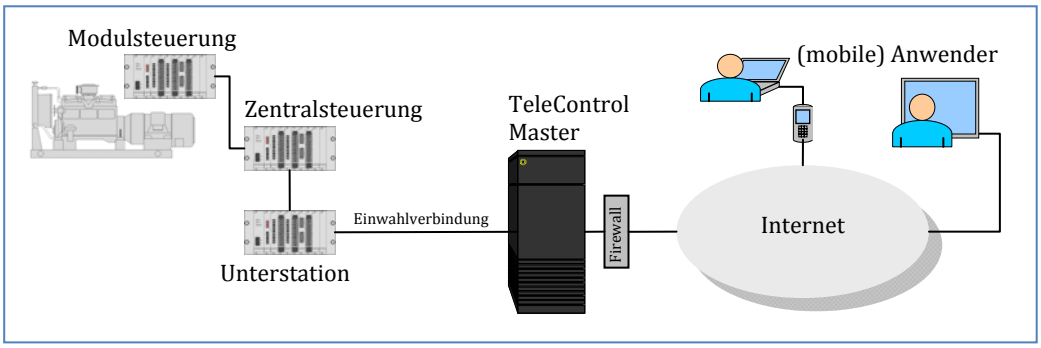


Bild 1: Übersicht KUHSE TeleControl System

Eine weitere Funktion des TeleControl Systems ist die Alarmierung bei bestimmten Ereignissen oder Betriebszuständen. Die Unterstation versendet dazu je nach Ausführung eine SMS oder Email an eine definierte Telefonnummer bzw. Email-Adresse.

3 Spezifikation (Lastenheft) [4, 5]

In einer Entwicklungsbesprechung wurden Anforderungen an ein Netzwerk Gateway für das TeleControl System diskutiert und aus den Ergebnissen eine Spezifikation festgelegt.

Ziel der Entwicklung des KUHSE Netzwerk Gateways (KNG) ist die Möglichkeit, KUHSE Geräte in ein Netzwerk (10/100-BaseT Ethernet) zu integrieren. Die Datenverbindung zu den Geräten wird dabei seriell über den KUHSE Standard-Lichtwellenleiter (LWL) oder ein 9-poliges D-SUB Kabel hergestellt.

Das KNG selber soll ein Kleingerät im Kunststoffgehäuse für Hutschienenmontage sein. Über ein integriertes Netzteil für Eingangsspannungen von 12 bis 36 V DC soll das Gerät mit Spannung versorgt werden.

Für den Benutzer sollen am Gerät betriebsrelevante Zustände durch LEDs angezeigt werden. Außerdem soll ein Reset-Taster vorgesehen werden.
Für detaillierte Informationen zum Betriebszustand und zur Verbindungskontrolle soll ein Web-Interface (HTML-Interface) in das KNG integriert werden.

Die Herstellkosten für das KNG sollen deutlich unter 200 Euro liegen.

3.1 Anwendung

Die erste konkrete Anwendung, für die das KNG benötigt wird, ist der Einsatz im KUHSE TeleControl System. Hier soll das bisher in der KUHSE TeleControl Unterstation eingesetzte Modem (Analog, ISDN oder GSM) ersetzt werden, um so für die Datenverbindung von der Unterstation zum TeleControl Master eine DSL Internet-Verbindung zu nutzen. Die DSL Verbindung selber wird dabei nicht vom KNG hergestellt, sondern muss von einem separaten Router verwaltet und zur Verfügung gestellt werden. (siehe Bild 2)

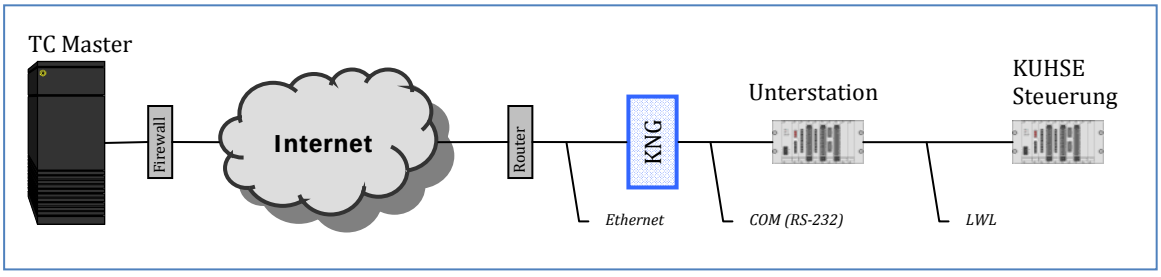


Bild 2: KNG – Einsatz als Gateway

Die Möglichkeit eine DSL-Verbindung zu nutzen ist notwendig, um den Anforderungen des Marktes zu begegnen. Für das TeleControl System soll mit der „Always-on“ Option eine Permanent-Verbindung zum Master über DSL zuverlässig und kostengünstig angeboten werden können.

Die Vorteile der DSL Verbindung sind auch für Anlagen ohne die „Always-on“ Option eine gute Alternative zur Modemverbindung. Meistens sind an den Einsatzorten Netzwerke mit DSL-Verbindung bereits verfügbar. Vor allem für Anlagen in Ländern mit schlechten Telefonnetzen ist die Zuverlässigkeit der DSL-Verbindung ausschlaggebend.

Ein Fernziel für die Entwicklung des KNG ist der vollständige Ersatz der Unterstation. (siehe Bild 3) Hierzu werden zusätzlich nicht-flüchtiger Speicher für Trend- und Protokolldaten und eine Echtzeit-Uhr (RTC) benötigt.

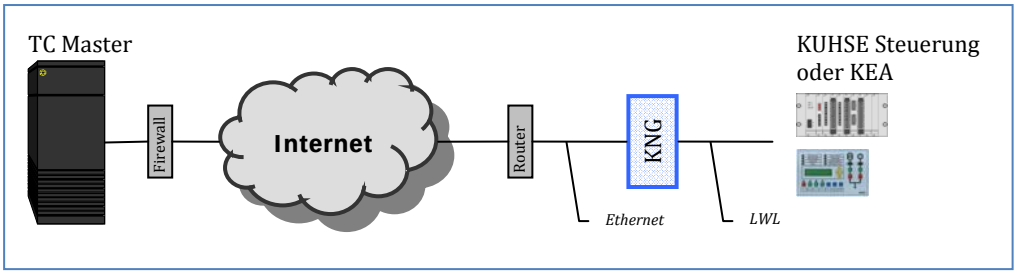


Bild 3: KNG – Einsatz als Unterstation

Darauf aufbauend sollen auch kleinere Energieerzeugungsanlagen, die z.B. nur mit einer KUHSE Kleinsteuerung vom Typ KEA ausgestattet sind, in das TeleControl System eingebunden werden können. Damit ist für diese Anlagen z.B. die Einbindung in ein virtuelles Kraftwerk möglich.

3.2 Optionen

Die in das KNG integrierte serielle Schnittstelle soll neben der Funktion als RS-232/EIA-232 (COM) Schnittstelle durch eine alternative Bestückung als RS-422,-485/EIA-422,-485 Schnittstelle ausgeführt werden können. Über diese Schnittstelle soll eine Ankopplung an den in Industrieanlagen verbreiteten Modbus oder an den Siemens MPI Bus möglich sein.
Die entsprechenden Gewerke sollen damit in ein TeleControl System eingebunden werden.

Zusätzlich soll die spätere Integration einer CAN Schnittstelle berücksichtigt werden, um weitere Kopplungsmöglichkeiten für das Gateway zu erhalten. Der CAN Bus ist ebenfalls in Industrieanlagen stark verbreitet und neue KUHSE Geräte sind bereits mit einer entsprechenden Schnittstelle ausgestattet.

4 Konzeption [6]

Grundlage für die Konzeption des KNG war der Entwurf einer Multi-Schnittstellenkarte, die im Rahmen der Studienarbeit erstellt wurde. Hiervon konnten nach entsprechender Prüfung die meisten Hardware-Komponenten und Schaltungsteile übernommen werden.
Über bestimmte Schaltungselemente wurden weitere Informationen und Empfehlungen von Distributoren eingeholt bzw. im Internet gesammelt und ausgewertet.

4.1 Prozessor [7]

Es wurde geprüft, ob der für die Multi-Schnittstellenkarte vorgesehene Coldfire MCF5270 Prozessor von Freescale (ehem. Motorola) auch für die Anwendung im KNG die optimale Lösung ist. Für die Prüfung wurden Anforderungen an den Prozessor für das KNG formuliert:

- Interface für externes SD-RAM
- Businterface (externer Adress- und Datenbus)
- interner Flash Speicher
- 2 serielle Schnittstellen
- interner Netzwerk-Controller – wenn möglich mit integriertem PHY
- CAN Interface
- QFP Gehäuse – wenn möglich aber auch im BGA Gehäuse erhältlich

Aus dem Freescale Selectors Guide wurden aufgrund der Anforderungen vier Prozessoren in die engere Wahl gezogen. Für diese Prozessoren wurden die Vor- und Nachteile gegenübergestellt.

Prozessor	**Preis*[1]**	**Pro**	**Contra**
MCF5208	$6,50	SD/DDR-RAM Interface QFP und BGA Gehäuse Preis	nur 16 kB internes SRAM kein internes Flash kein CAN
MCF5234	$13	SD-RAM Interface CAN	kein internes Flash nur BGA Gehäuse Preis
MCF5270	$7,50	SD/DDR-RAM Interface QFP und BGA Gehäuse Preis bereits intensiv informiert	kein internes Flash kein CAN
MCF5281 / MCF5282	$16,05 / $17,45	internes Flash (256 kB / 512 kB) SD-RAM Interface CAN	nur BGA Gehäuse Preis

Tabelle 1: Prozessorvergleich

*[1]) Preis pro Stück bei Abnahme von min. 1000 Stück

Alle Prozessoren verfügen – soweit nicht anders aufgeführt – über 64 kB internes SRAM, zwei oder mehr serielle Schnittstellen, einen internen Netzwerk-Controller und ein Businterface.

Da alle in Frage kommenden Prozessoren mit CAN Interface und / oder internem Flash Speicher vergleichsweise teuer sind und sich die entsprechenden Features durch externe Bausteine realisieren lassen, wurde auf diese Lösungen verzichtet.
Bei den verbleibenden zwei Prozessoren überwiegen die Vorteile des MCF5270, so dass dieser Prozessor übernommen werden konnte.

4.2 Peripherie (Speicher) [8, 9, 10]

Alle vorgesehenen Speicherbausteine wurden aus der Konzeption der Multi-Schnittstellenkarte übernommen.
Lediglich das DP-RAM wurde nicht übernommen, da es im KNG nicht benötigt wird.

Folgende Bausteine sind damit im KNG integriert:

Art	Typ	Interface-Breite	Speichergröße
Flash Speicher	Spansion S29AL004D	16 Bit (Wort)	512 kB
SD-RAM	Micron MT48LC2M32B2	32 Bit (Langwort)	8 MB

Tabelle 2: KNG Peripheriebausteine (Speicher)

4.2.1 Flash Speicher

Der Flash Speicherbaustein S29AL004D von Spansion (siehe Bild 4) ist mit folgenden Daten spezifiziert:

- Busbreite: 8 oder 16 Bit
- Speicherkapazität: 4 MBit = 512 KB
- Zugriffszeiten: 40/50/55/70 ns
- Gehäuse: TSOP-48

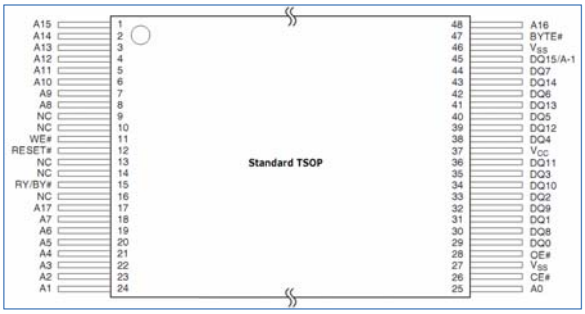


Bild 4: Flash Speicherbaustein S29AL004D

Flash Speicher wird zur nicht-flüchtigen Speicherung der KNG Betriebssoftware (Firmware) benötigt. Außerdem wird im Flash Speicher ein Bootloader untergebracht, der nach einem Reset die Firmware in das SD-RAM kopiert und von dort ausführt.
Die Ausführung eines Programms aus dem SD-RAM ist deutlich schneller als die Ausführung aus dem externen Flash Speicher. Bedingt ist dies durch die unterschiedlichen Zugriffszeiten und die Breite der Anbindung. Der vorgesehene Flash Speicher ist mit Zugriffszeiten zwischen 40 ns und 70 ns erhältlich, das vorgesehene SD-RAM mit Zeiten zwischen 5 ns und 7 ns. Außerdem wird das SD-RAM mit 32 Bit (32 parallelen Datenleitungen) an den Prozessor angebunden und der Flash Speicher lediglich mit den maximal möglichen 16 Bit.
Da die Firmware nicht aus dem Flash Speicher sondern aus dem SD-RAM ausgeführt werden soll, reicht die langsamste Geschwindigkeitsvariante des Bausteins mit 70 ns Zugriffszeit für das KNG aus.

Die Speichergröße des Flash Speichers von 512 kB ist nach Erfahrungen der Firma KUHSE ausreichend, um Bootloader und Firmware unterzubringen. Sollte für spätere Anwendungen mehr Flash

Speicher benötigt werden, sind von Spansion Speicherbausteine mit bis zu 8 MB Speicher verfügbar, die pinkompatibel zu dem 512 kB Baustein sind. Es werden allerdings je nach Größe bis zu vier weitere Adressleitungen benötigt. Diese werden im KNG entsprechend vorgesehen, so dass ggf. keine Überarbeitung der Platine nötig ist.

Der Flash Speicher ist an die Chip Select Leitung CS0 angebunden. Dies ist notwendig, damit das Bootloader-Programm nach einem Reset vom Prozessor ausgeführt wird. Dieser lädt nach einem Reset seine Interrupt Vector Tabelle und damit auch die Adresse des Startprogramms immer zuerst durch einen Speicherzugriff über CS0.

4.2.2 SD-RAM

Der übernommene SD-RAM Baustein MT48LC2M32B2 von Micron (siehe Bild 5) ist wie folgt spezifiziert:

- Busbreite: 32 Bit
- Speicherkapazität: 64 Megabit = 8 MB
- Speicherteilung: 4 Bänke mit je 2^{11} Zeilen (Rows) und 2^{8} Spalten (Columns)
- Geschwindigkeit: 143/166/183/200 MHZ
- Gehäuse: TSOP-86/VFBGA-90

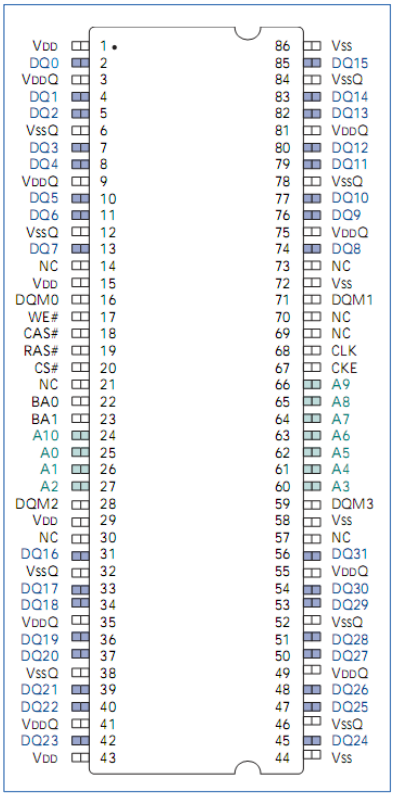


Bild 5: SD-RAM Speicherbaustein MT48LC2M32B2

Aus dem SD-RAM wird die aus dem Flash Speicher kopierte Firmware des KNG ausgeführt. Außerdem werden hier Speicherbereiche für Datenpuffer genutzt. Das gewählte SD-RAM bietet hier mit 8 MB Speicher mehr Speicher, als für die derzeitige Anwendung benötigt wird. Allerdings ist geplant, eine Version des Gerätes mit einem µCLinux als Betriebssystem auszustatten. Dafür ist die vorgesehene Speichergröße empfohlen. Außerdem werden kleinere Speichergrößen generell nicht mehr so häufig eingesetzt, so dass es hier auch keinen preislichen Vorteil gibt. Es kann im Gegenteil sogar zu Beschaffungsschwierigkeiten kommen.

Der SD-RAM Speicherbaustein wird über den integrierten SD-RAM Controller des Prozessors angesteuert. Dieser übernimmt die notwendige Adressierung der Bänke, Zeilen und Spalten. Die Anbindung ergibt sich aus dem Datenblatt. (siehe Bild 6)

MCF5271 Pins	A15	A14	A13	A12	A11	A10	A9	A17	A19	A20	A21	A22	A23
Row	15	14	13	12	11	10	9	17	19	20	21	22	23
Column	2	3	4	5	6	7	8	16	18				
SDRAM Pins	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12

Bild 6: Anbindung von SD-RAM an den MCF5270

Da der SD-RAM Controller den Speicher mit dem Systemtakt des Prozessors (max. 100 MHz) taktet, reicht die langsamste SD-RAM Variante mit 143 MHz (rund 7 ns Zugriffszeit) aus. Es können deshalb je nach Verfügbarkeit und Angebot alle Varianten eingesetzt werden.

4.3 Schnittstellen

4.3.1 Ethernet (Netzwerk) [11, 12, 13]

Der für die Netzwerk-Schnittstelle notwendige Physical Layer Interface-Baustein (PHY) wurde nicht von der Multi-Schnittstellenkarte übernommen, da dieser eher schwierig zu beschaffen ist und eine relativ große Bauform hat.
Hier wurde nach einem Gespräch mit einem Distributor der PHY KSZ8041 von Micrel gewählt. Diesen Baustein gibt es in drei Ausführungen mit folgenden Unterschieden, wobei der KSZ8041TL und der KSZ8041FTL untereinander pinkompatibel sind:

Baustein	Gehäuse	Medien	besondere Features
KSZ8041NL	MLF-32	10/100 MBit/s Kupferkabel	-
KSZ8041TL	TQFP-48	10/100 MBit/s Kupferkabel	Kabeldiagnose
KSZ8041FTL		10/100 MBit/s Kupferkabel 100 MBit/s optisch	Kabeldiagnose

Tabelle 3: Vergleich Ethernet PHYs der Micrel KSZ8041-Serie

Der KSZ8041NL wurde hauptsächlich aus fertigungstechnischen Gründen nicht ausgewählt. Das MLF Gehäuse hat alle Anschlüsse ähnlich dem BGA Gehäuse an der Unterseite. Hier kann es Schwierigkeiten in der Verarbeitung geben.

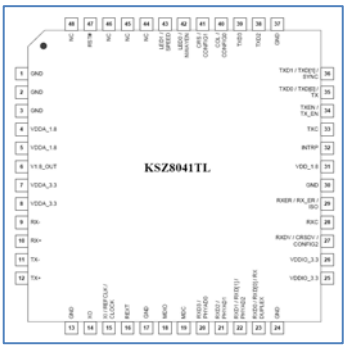


Bild 7: Ethernet PHY KSZ8041TL

Gewählt wurde stattdessen der KSZ8041TL (siehe Bild 7). Dieser Baustein bietet außer dem besser zu verarbeitenden TQFP Gehäuse mit der integrierten Kabeldiagnose ein interessantes Feature.
Mit dem pinkompatiblen KSZ8041FTL können darüber hinaus in späteren Hardware-Versionen auch Netzwerkverbindungen über optische Leiter realisiert werden.

Da für jeden PHY eine bestimmte Übertragerschaltung für die galvanische Trennung zum Netzwerk eingesetzt werden muss, konnte auch die RJ45 Netzwerk-Buchse nicht von der Multi-Schnittstellenkarte übernommen werden.
Passend zum KSZ8041 wurde vom Distributor auch eine RJ45 Buchse angeboten, die ebenfalls über eine interne Übertragerschaltung verfügt. Die Buchse Nr. 6605758 von Tyco (siehe Bild 8) gibt es außerdem in einer Ausführung mit zwei integrierten LEDs. Diese Ausführung ist für den Einsatz im KNG ausgewählt worden. Die gelbe LED (LED1) wird dabei zur Datenverkehr-Anzeige (ACT) verwendet. Die grüne LED (LED2) zur Anzeige des Verbindungsstatus (Link).

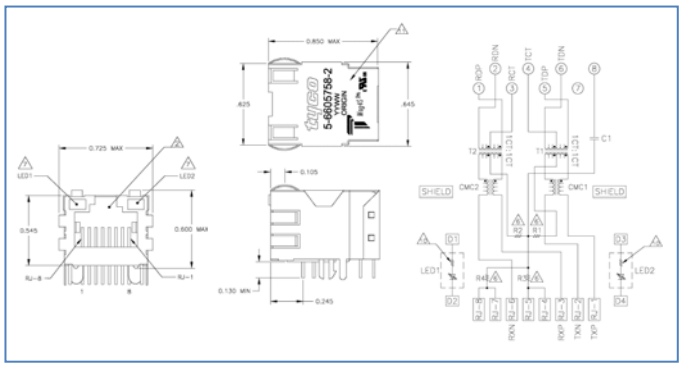


Bild 8: RJ-45 Buchse 5-6605758-2 mit integrierter Übertragerschaltung und LEDs

4.3.2 LWL Schnittstelle [14, 15]

Bild 9: LWL Schnittstelle

Für die LWL Schnittstelle wurden die KUHSE Standardkomponenten übernommen. (siehe Bild 9)
Als Sender wird der HFBR-1523 und als Empfänger der HFBR-2523 von der Firma Avago (ehemals HP) eingesetzt. Diese Bauteile sind für Datenübertragungsraten von maximal 40 kBit/s auf einer Strecke von maximal 120 m (bei 25 °C) spezifiziert.

4.3.3 Serielle Schnittstelle (D-SUB)

Die serielle Schnittstelle des KNG ist als 9-polige D-SUB Buchse ausgeführt. Sie kann durch entsprechende Bestückung entweder als EIA-232 oder als EIA-485 Schnittstelle betrieben werden, wobei die Standard-Schnittstelle die EIA-232 Schnittstelle ist. Diese ist außerdem über eine interne Stiftleiste verfügbar, um sie z.B. zur Unterstützung bei der Software-Entwicklung einsetzen zu können.

4.3.3.1 EIA-232 (COM) Schnittstelle [16, 17]

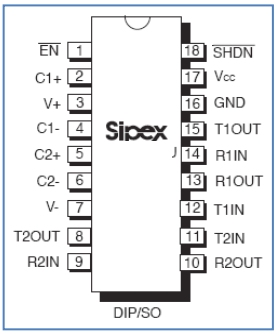


Bild 10: EIA-232 Treiber SP3232EU

Für die Empfangs- und Sendeleitungen (RxD, TxD) der COM Schnittstelle wird ein separater Treiber benötigt. Der gewählte Treiber SP3232EU wurde von der Muli-Schnittstellenkarte übernommen. (siehe Bild 10)

Aufgabe des Treibers ist die Erzeugung der notwendigen Schnittstellen-Pegel. Für EIA-232 ist eine Spannung zwischen -15 V und -3 V als logische Eins für Datenleitungen bzw. als logische Null für Signalleitungen sowie analog dazu eine Spannung zwischen +3 V und +15 V als logische Null für Datenleitungen bzw. logische Eins für Signalleitungen definiert.
Der gewählte Treiber liefert ±5 V an den Treiberausgängen. Außerdem übernimmt er die Invertierung der Datenleitungen.

4.3.3.2 EIA-485 (MPI) Schnittstelle [18, 19, 20, 21]

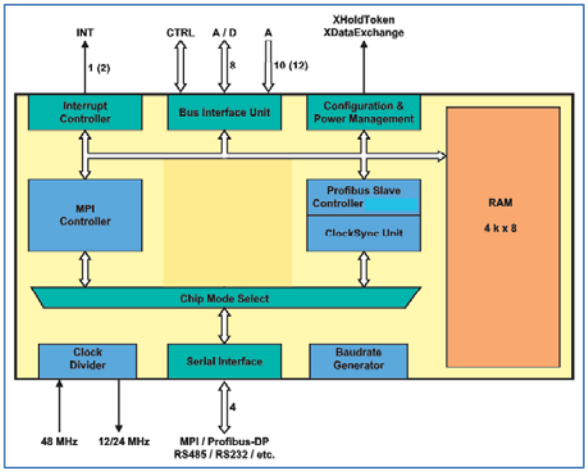


Bild 11: Blockdiagramm des MPI12x

Wird die D-SUB Schnittstelle verwendet, um an den MPI Bus anzukoppeln, muss der Interface-Baustein MPI12x von Profichip (siehe Bild 11) bestückt werden.
Mit diesem Baustein ist auch eine Ankopplung an einen Profibus als DP-Slave Gerät möglich.

Der MPI12x ist ein eigener Controller, der über den Adress- und Datenbus des Prozessors angebunden wird. Daten werden über ein in den Baustein integriertes Dual Port RAM (DP-RAM) ausgetauscht.

Um an den differentiellen Bus (Profibus/MPI) anzukoppeln, wird ein tristate-fähiger Bustreiber benötigt, der das differentielle Datensignal erzeugt bzw. die Bussignale in CMOS Pegel zurückwandelt.
Im KNG wird dafür der DS75176B von National Semiconductor eingesetzt. (siehe Bild 12) Dieser Baustein wird auch im Referenzdesign des MPI12x verwendet.

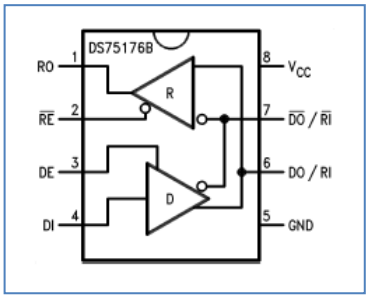


Bild 12: EIA-485 Bustreiber DS75176B

Die Datenleitungen zwischen Bustreiber und MPI12x werden zur galvanischen Trennung über Optokoppler geführt.
Für die schnellen Übertragungsraten von bis zu 12 MBit/s sind besondere Typen notwendig, die diese Übertragungsraten unterstützen. Hier wurde der ebenfalls im Referenzdesign des MPI12x verwendete HCPL-0710 von Avago übernommen. (siehe Bild 13) Dieser Baustein ist für Übertragungsraten von maximal 12 MBit/s spezifiziert und damit ausreichend.
Für das Steuersignal RTS (ready to send) wird der bei KUHSE bereits eingesetzte langsamere Optokoppler HCPL-0601 eingesetzt.

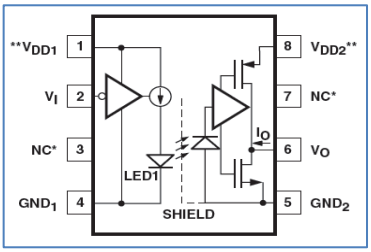


Bild 13: Optokoppler HCPL-0710

4.3.4 CAN Schnittstelle [22, 23]

Da der Prozessor nicht über ein internes CAN Interface verfügt, wird ein externer CAN Controller benötigt. Hier wurde aufgrund der hohen Verfügbarkeit bei mehreren Distributoren der MCP2515 von Microchip gewählt. (siehe Bild 14)

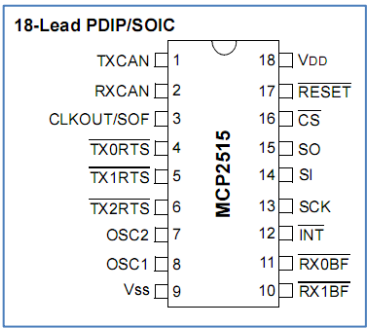


Bild 14: SPI CAN Controller MCP2515

Der Interface-Baustein wird über den SPI (QSPI) Bus des Prozessors angebunden. Er unterstützt den CAN Standard V2.0B und kann sowohl Standard-Telegramme als auch erweiterte Telegramme behandeln. Um den Kommunikationsaufwand mit dem Hauptprozessor möglichst gering zu halten, sind zwei Masken und sechs Filter einstellbar, die auf ankommende Telegramme angewendet werden.

Zusätzlich zum Interface-Baustein wird auch für den CAN Bus ein Bustreiber benötigt, der die Signalumsetzung zwischen dem CAN Controller und dem differentiellen CAN Bus durchführt.
Hier wurde auf den bereits bei KUHSE verwendeten PCA82C250 von NXP (ehemals Philips) zurückgegriffen. (siehe Bild 15)
Dieser Treiber ist für die maximale Busgeschwindigkeit von 1 MBit/s spezifiziert.

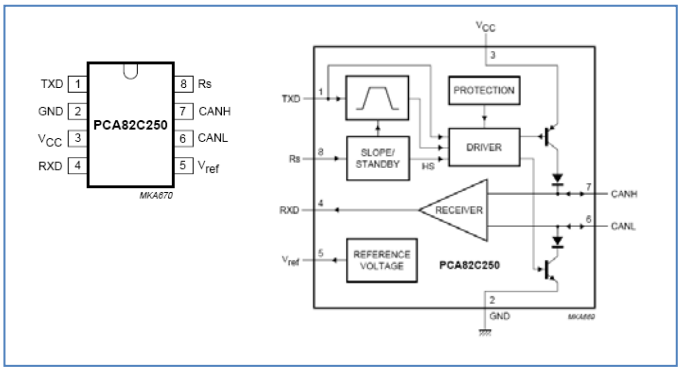


Bild 15: CAN Bustreiber PCA82C250

Die CAN Schnittstelle ist aktuell nur intern über eine Stiftleiste verfügbar. Die Verwendbarkeit soll zunächst getestet und der Bedarf genau geprüft werden.

4.4 Datenspeicher [24, 25, 26]

Für den Einsatz des KNG als vollwertige TeleControl Unterstation, wird als Datenspeicher für Trend- und Protokolldaten eine MiniSD Speicherkarte eingesetzt. (siehe Bild 16) SD-Karten sind weit verbreitet und können über ein Kartenlesegerät auch direkt mit einem PC ausgelesen werden – vorausgesetzt ein entsprechendes Dateisystem wird verwendet.

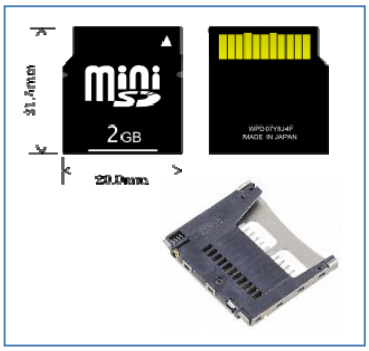


Bild 16: MiniSD Speicherkarte mit Kartenhalter

Beim Zugriff auf MiniSD Karten werden Datenübertragungsraten von durchschnittlich ca. 2 MB/s, bei sog. High-speed Karten von bis zu 20 MB/s erreicht.
Aktuelle MiniSD Karten sind mit Speichergrößen von 32 MB bis hin zu 4 GB erhältlich.

Um die Speicherkarte in das KNG zu integrieren wird ein entsprechender Kartenhalter benötigt. (siehe Bild 16) Aufgrund der guten Verfügbarkeit ist der Kartenhalter Nr. 500525-1100 der Firma Molex ausgewählt worden. Über diesen Halter wird die Speicherkarte mit dem SPI (QSPI) Bus des Prozessors verbunden.

Die Speicherkarte wird so im KNG untergebracht, dass sie nicht direkt von außen zugänglich ist. Sie soll nicht vom Anwender und nicht während des Betriebes entnommen werden können.

4.5 SPI [10, 27]

Das Serial Peripheral Interface (SPI) ist im MCF5270 als Queued Serial Peripheral Interface (QSPI) integriert und bietet damit zusätzliche Features, wie z.B. Nachrichtenwarteschlangen (Queues) und programmierbare Chip Select Leitungen. In der eingesetzten Gehäuseform bietet der Prozessor allerdings nur eine QSPI Chip Select Leitung.

Da im KNG das CAN Interface, die SD Speicherkarte und später evtl. die Echtzeituhr per QSPI an den Prozessor angebunden werden, muss eine externe Chip Select Erweiterung vorgesehen werden. Hierfür wurde der Multiplexer ADG704 von Analog Devices ausgewählt (siehe Bild 17), über den die verfügbare Chip Select Leitung mit Hilfe von zwei Standard-Ausgängen in vier Leitungen aufgeteilt wird.

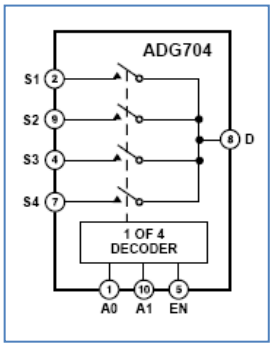


Bild 17: Multiplexer ADG704

4.6 Geräte-Aufbau [28, 29]

Für das KNG wurde ein Gehäuse aus der ME Gehäuseserie der Firma Phoenix Contact ausgewählt. (siehe Bild 18) In das Kunststoffgehäuse mit einer Breite von 45 mm können zwei Platinen eingeschoben werden. Diese werden für das KNG auch benötigt.
An der unteren Gehäuseseite befindet sich eine Aufnahme zur Hutschienenbefestigung.

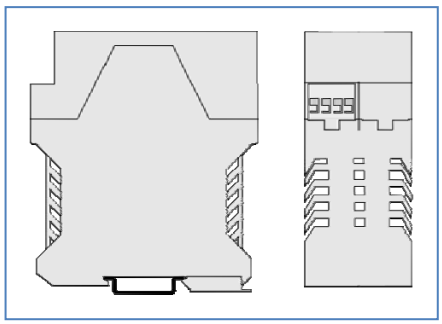


Bild 18: KNG Gehäuse

Das dazu passend gewählte Gehäuse-Oberteil bietet eine große Oberfläche für die Schnittstellen (LWL, D-SUB, RJ-45) und das User-Interface (Reset-Taster und LEDs).
Außerdem ist pro Platine ein vierpoliger Stecker mit Schraubklemmen vorgesehen. Einer davon wird für die Spannungsversorgung verwendet. Der andere wird aktuell nicht benötigt und die entsprechende Öffnung wird mit einer Blindabdeckung versehen.

Folgende Phoenix Komponenten werden eingesetzt:

Teil	Phoenix-Artikel	Basisartikelnummer
Gehäuseunterteil	ME 45 UT/FE KMGY	2909358
Gehäuseoberteil	ME 45 OT-1MSTBO KMGY	2709192

Tabelle 4: KNG Gehäusekomponenten (Phoenix)

Für das User-Interface sind vier LEDs vorgesehen, die folgenden Zustände anzeigen:

- Betrieb (grün)
- Reset (rot)
- Status der Netzwerkschnittstelle (gelb)
- Status der seriellen Schnittstelle (gelb)

5 Realisierung der Hardware [6, 30]

Die Hardware wurde mit Hilfe der Software EAGLE (Einfach anzuwendender grafischer Layout Editor), Version 4.14, der Firma CadSoft Computer GmbH realisiert. Das Programm beinhaltet einen Schaltplan und PCB Layout Editor.

Diverse Bauteile mussten in einer neuen Bibliothek angelegt werden, da diese nicht Teil einer Standard-Bibliothek sind. Dazu musste ein Funktionsblock mit entsprechenden Anschlüssen für den Schaltplan sowie das Gehäuse (Package) für das Layout erstellt werden. Da die meisten Gehäuse verbreitete Standard-Gehäuse sind, konnten diese kopiert werden und mussten nicht neu gezeichnet werden.

5.1 Schaltplan

Bedingt durch die Aufteilung der Schaltung auf zwei Platinen mussten zwei separate Schaltpläne erstellt werden. Dazu war es notwendig, schon im Voraus festzulegen, welche Teile auf welcher Platine untergebracht werden sollen. Es wurde eine Liste der Schaltungselemente – sortiert nach vermutetem Platzbedarf – erstellt und diese den Platinen zugeordnet:

Schaltungselement	**Platine**	
	Vorauswahl	**Zuordnung**
CPU und Peripherie	1	1
Netzteil	2	2
QSPI (→ CAN, SD Speicherkarte)	?	2
Ethernet Interface (PHY)	1	1
EIA-485 Interface (MPI12x)	1	1
BDM Programmier-Interface	1	1
User Interface (LEDs, Reset-Taster)	1	1
EIA-485 Schnittstelle	?	2
EIA-232 Schnittstelle	?	2
LWL Schnittstelle	?	2
Ethernet Schnittstelle (RJ45)	?	2

Tabelle 5: Aufteilung der Schaltungselemente

Die Platinen wurden nach ihrem Kernbestandteil als CPU-Platine bzw. Netzteilplatine bezeichnet.

Einige Schaltungselemente wurden in einer Vorauswahl noch nicht zugeordnet, da diese nicht zwingend auf einer bestimmten Platine untergebracht werden mussten. Generell ist darauf geachtet worden, die Anzahl der Verbindungsleitungen zwischen den Platinen möglichst gering zu halten. Deshalb wurden z.B. die Interface-Schaltungen und das User-Interface der CPU-Platine zugewiesen.
Die verbleibenden Schaltungselemente (QSPI und Schnittstellen) wurden im zweiten Schritt aufgrund der Platzverhältnisse der Netzteilplatine zugewiesen.

5.1.1 CPU-Platine

Der Schaltplan der CPU-Platine konnte nahezu komplett aus der Konzeption der Multi-Schnittstellenkarte übernommen werden, da derselbe Prozessor mit weitgehend denselben Peripheriebausteinen eingesetzt wird.

5.1.1.1 Ethernet Interface (PHY) [10, 12]

Der bisher geplante Intel PHY musste durch den neuen PHY ersetzt werden. Durch das definierte Media Independent Interface (MII) zwischen PHY und Prozessor war dies allerdings einfach möglich.
Die externe Konfigurationsbeschaltung für den alten PHY entfiel, da der KSZ8041TL durch interne Pull-up bzw. Pull-down Widerstände eine geeignete Grundkonfiguration lädt. Lediglich für die LEDs wurden Pull-up Widerstände vorgesehen. (siehe Bild 19)

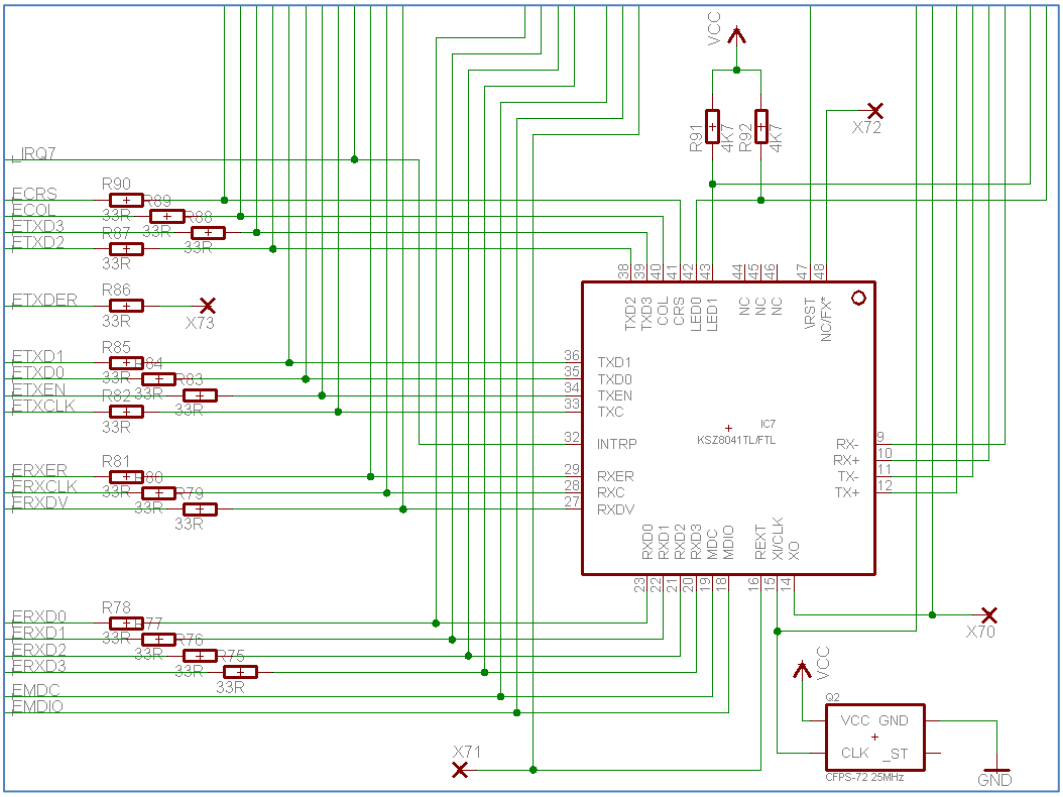


Bild 19: Anschluss Ethernet PHY

Die Serienwiderstände in den Leitungen des MII sind aus dem Referenzdesign für den PHY übernommen und dienen der Signal-Qualität.

Es wurde im Schaltplan auch die kleinere Variante des neuen PHY (KSZ8041NL) eingeplant, um ggf. einen einfachen Austausch zu ermöglichen.

5.1.1.2 EIA-485 Interface (MPI12x) [31]

Die Beschaltung des MPI12x wurde komplett übernommen. Lediglich der Jumper, der die Adressleitung µA10 an den MPI12x anschließt, wurde durch einen SMD DIP Schalter (S1) ersetzt. (siehe Bild 20)
Die Adressleitung µA10 wird für den Profibus-Betrieb des Bausteins benötigt.

Auch die zweite Reset Möglichkeit für den MPI12x über einen Standard-Ausgang (DTIN1) wurde übernommen. Die Verknüpfung dieses Signals mit dem Reset-Ausgang (RSTOUT) des Prozessors erfolgt als Oder-Verknüpfung über zwei Dioden.

Da der Reset-Eingang des MPI12x high-aktiv ausgeführt ist, muss das generierte Reset-Signal über einen Inverter geführt werden.

Die Konfigurations-Pins werden jeweils über einen Pull-up bzw. Pull-down Widerstand angeschlossen.
So kann notfalls die Boot-Konfiguration geändert werden.

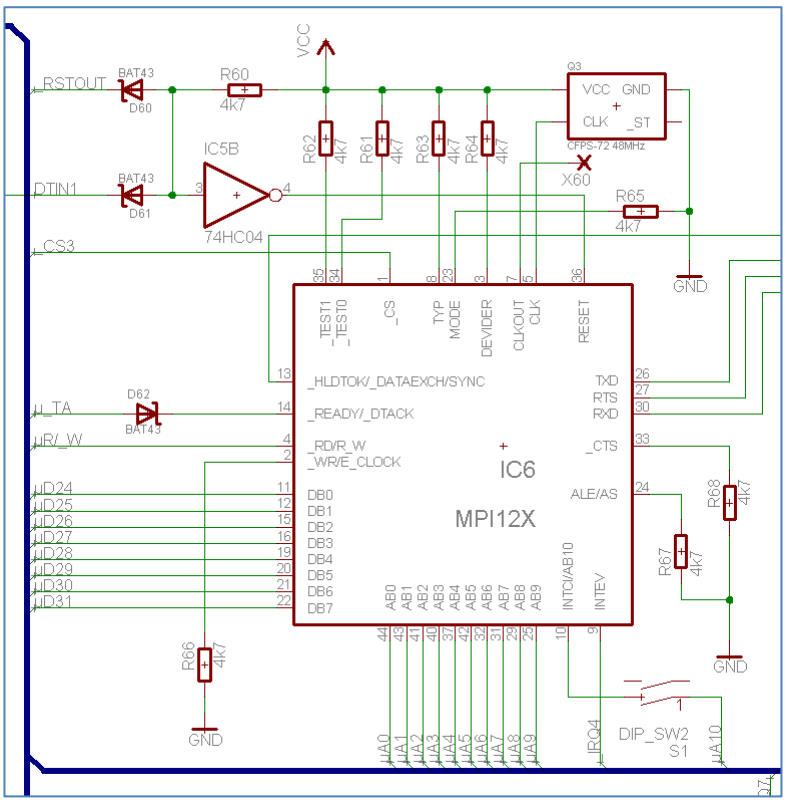


Bild 20: Anschluss MPI12x

Die gewählte Konfiguration stammt aus dem Datenblatt und ergibt sich wie folgt:

Pin	Signal	Wert	Bedeutung
3	Divider	0	Ausgangstakt = Eingangstakt / 4
		1 (gewählt)	Ausgangstakt = Eingangstakt / 2
8	Typ	0	Intel Interface
		1 (gewählt)	Motorola Interface
23	Mode	0 (gewählt)	separater Adress- und Datenbus
		1	Adress- und Datenbus multiplexed
34	Test0	0	alle Ausgänge hochohmig
		1 (gewählt)	Normalbetrieb
35	Test1	0	Test-Modus (genaue Funktion unbekannt)
		1 (gewählt)	Normalbetrieb

Tabelle 6: MPI12x Konfiguration

Außerdem wurden nach dem Anwendungsbeispiel im Datenblatt die Signalleitungen E-Clock (Pin 2), ALE (Address Latch Enable – Pin 24) und CTS (Clear to Send – Pin 33) über einen 4,7 kΩ Widerstand auf Masse gelegt.

5.1.1.3 Reset-Logik

Die Reset-Logik der Multi-Schnittstellenkarte konnte nicht direkt übernommen werden. Dem KNG fehlt dazu der Reset-Eingang vom 68k-Bus und der bisher eingeplante interne Jumper wurde durch den Reset-Taster (S2) vom User-Interface ersetzt. (siehe Bild 21)

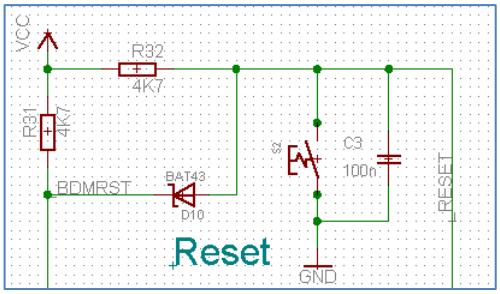


Bild 21: Reset-Logik

Zusätzlich ist noch ein 100 nF Kondensator vorgesehen worden, um die Reset-Leitung zu entstören.

5.1.1.4 LED Anzeigen (User-Interface) [32, 33]

Für das User-Interface des KNG wurden die bedrahteten Doppel-LEDs aus der Multi-Schnittstellenkarte gegen SMD LEDs von Osram ausgetauscht. (siehe Tabelle 7) Diese werden bereits in einem anderen KUHSE Gerät eingesetzt.
Die Anzeige am Gehäuse erfolgt über Lichtleiter aus Kunststoff.

LED	Bedeutung	Farbe	Typ	Betriebsdaten *1
D1	Betrieb	grün	Osram TOPLED LG T679	U_F=1.9-2.5V, I_F=2mA
D2	Reset (Signal RSTOUT)	rot	Osram TOPLED LS T679	U_F=2.0-2.6V, I_F=2mA
D3	Netzwerk Status (Link und Aktivität)	gelb	Osram TOPLED LY T679	U_F=1.8-2.5V, I_F=2mA
D4	EIA-485 Status (Aktivität)	gelb	Osram TOPLED LY T679	U_F=1.8-2.5V, I_F=2mA

Tabelle 7: User-Interface LEDs

*[1]) Werte aus dem Datenblatt

Angesteuert werden die LEDs ebenso wie in der Multi-Schnittstellenkarte über einen Inverter (74HC04) als Treiber.

Aufgrund der neuen LEDs mussten auch die Vorwiderstände neu berechnet werden. Für die Durchfluss-Spannung wurde der gemittelte Wert von 2 V verwendet:

$$R_V = \frac{U_H - U_F}{I_F} = \frac{3{,}3V - 2V}{2mA} = 650\Omega$$

U_H = Ausgangsspannung des Inverters im HIGH Zustand

Der E24 Reihe entsprechend wurde der Widerstandswert 680 Ω vorgesehen.

5.1.2 Netzteilplatine

Auch für die Netzteilplatine konnten Teile aus der Konzeption der Multi-Schnittstellenkarte übernommen werden, wie z.B. das Netzteil selber oder das EIA-485 Interface.

Andere Baugruppen wie beispielsweise die LWL Schnittstelle oder der SPI Bus mit Speicherkarteninterface und CAN Controller / Interface mussten neu erstellt werden.

5.1.2.1 LWL Schnittstelle [15, 34]

Der Empfangsteil der LWL Schnittstelle ist gemäß den Empfehlungen aus dem Datenblatt aufgebaut worden. Die Empfangsleitung des Prozessors (U1TXD) wird mit einem Pull-up Widerstand stabilisiert und auf einen festen Ruhepegel gebracht. (siehe Bild 22)

Der Sendeteil der LWL Schnittstelle weicht hingegen leicht von den Empfehlungen im Datenblatt ab. Problem war hier die niedrige Versorgungsspannung von 3,3 V bei einer Durchfluss-Spannung für die Sendediode von 1,45 V bis 2,02 V. Außerdem reduziert der in Reihe geschaltete Steuerungstransistor die an der Sendediode verfügbare Spannung. Da auf der Netzteilplatine durch die andere serielle Schnittstelle 5 V zur Verfügung stehen, wurde diese als Versorgung für die Sendediode verwendet. (siehe Bild 22)

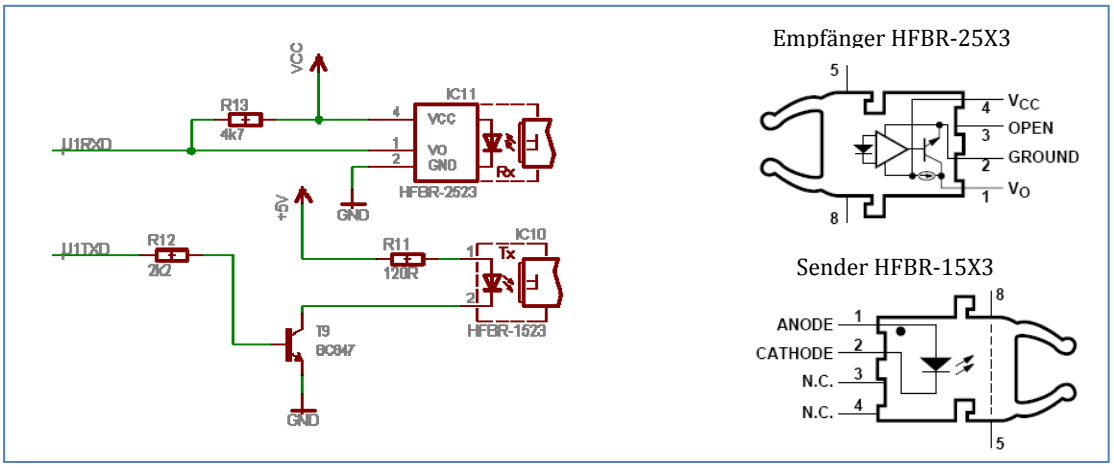


Bild 22: LWL Interface

Um den Stromverbrauch der Schaltung gering zu halten, wurde die Übertragungsstrecke der LWL Verbindung begrenzt. Es Wurde ein Wert von maximal 70 m festgelegt, der für das KNG ausreichend ist. Mit einem Durchfluss-Strom für die Sendediode von 30 mA wird laut Datenblatt eine maximale Strecke von 78 m erreicht. Damit sollten die festgelegten 70 m in jedem Fall eingehalten werden können.

Nach der Festlegung dieser Betriebsgrößen wurden folgende Berechnungen durchgeführt:

$$R_{11} = \frac{U_B - U_F - U_{CEsat}}{I_F} = \frac{5V - 1{,}6V - 0{,}11V}{30mA} = \frac{3{,}29V}{30mA} = 110\Omega$$

Die Durchfluss-Spannung der Sendediode von 1,6 V und die Kollektor-Emitter-Sättigungsspannung des Transistors von 110 mV wurden für den festgelegten Strom aus den jeweiligen Datenblättern ermittelt.
Aus der E24 Reihe wurde für R_{11} ein Wert von 120 Ω gewählt.

Der Basis-Vorwiderstand des Transistors (R_{12}) wurde wie folgt berechnet:

$$R_{12} = \frac{U_H}{I_B} = \frac{U_H}{I_C / 20} = \frac{3{,}3V}{30mA / 20} = 2{,}2k\Omega$$

5.1.2.2 Serielle Schnittstellen

Wie in der Spezifikation festgelegt, kann die serielle Schnittstelle entweder als EIA-232 (COM) Schnittstelle oder als EIA-485 Schnittstelle betrieben werden. In beiden Fällen ist sie als D-SUB Buchse (X7) ausgeführt. Die COM Schnittstelle ist außerdem auf eine interne 8-polige Stiftleiste (X6) gelegt. Die Betriebsart wird durch eine entsprechende Bestückung der 0-Ohm Widerstände R_{60} bis R_{65} gewählt. (siehe Bild 23)

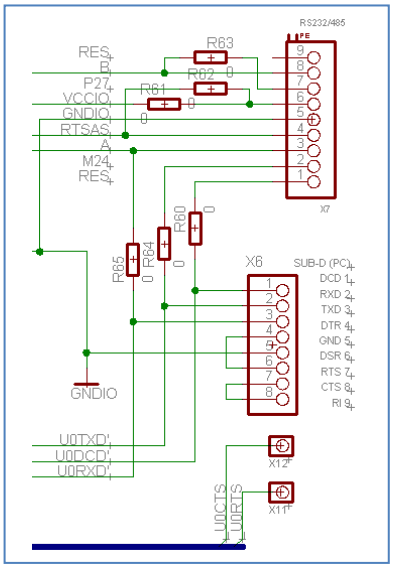


Bild 23: Serielle Schnittstelle

COM Schnittstelle [16, 17, 35]
Bei Verwendung der Buchse als COM Schnittstelle ist das KNG als Datenübertragungseinrichtung (DÜE) konfiguriert. (siehe Tabelle 8)

Pin	I/O	Signal
1	O	Data Carrier Detect (DCD)
2	O	Transmit Data (TxD)
3	I	Receive Data (RxD)
4	I	Data Terminal Ready (DTR)
5	-	Ground (GND)
6	O	Dataset Ready (DSR)
7	I	Request to Send (RTS)
8	O	Clear to Send (CTS)
9	-	nicht belegt

Tabelle 8: Belegung COM Schnittstelle (DÜE)

Die Signale DTR und DSR sind intern gebrückt, so dass das KNG immer Bereitschaft signalisiert, sobald die Gegenstelle / Datenendeinrichtung (DEE) bereit ist. Ebenso wird mit den Signalen RTS und CTS verfahren.
Die Signale U0RTS und U0CTS des Prozessors werden hier nicht eingesetzt, sondern sind auf Lötpads gelegt.

Der Schnittstellentreiber ist gemäß Datenblatt in die Sende- und Empfangsleitung geschaltet.
(siehe Bild 24)

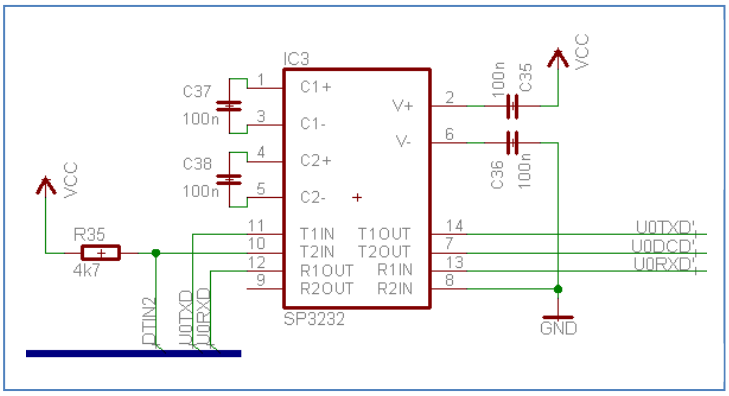


Bild 24: Beschaltung EIA-232 Treiber

Zur Steuerung des DCD Signals wird der als Ausgang konfigurierte I/O-Anschluss DT2IN des Prozessors verwendet. Das Signal wird auch über den Treiber geführt, um die notwendigen Pegel am Port zu erzeugen. Da das DCD Signal allerdings nicht in negativer Logik über die Leitung geführt wird, muss hier die Ansteuerung in der Firmware entsprechend invertiert programmiert werden. Es ist zusätzlich ein Pull-up Widerstand vorgesehen.

EIA-485 Schnittstelle [19, 20, 21]

Die EIA-485 Schnittstelle ist nach den Vorgaben aus dem Datenblatt des MPI12x realisiert worden. Der Schaltungsaufwand ist hier wegen der Optokoppler zur galvanischen Trennung höher als bei der COM Schnittstelle. Außerdem ist die Beschaltung des Bustreibers umfangreicher.

Der als Bustreiber eingesetzte Bustreiber DS75176 von National Semiconductor hat folgende Betriebszustände:

Driver Enabled (DE)	**Data In (DI)**	**Ausgang (A,B)**	**Eingang (A,B)**	**Receive Out (RO)**	**Betriebsart**
0	X	Z	(A-B) ≥ +0,2V	1	Empfangen
			(A-B) ≤ -0,2V	0	
			offen	1	
1	1	A = 5V, B = 0V		X	Senden
	0	A = 0V, B = 5V			

Tabelle 9: Betriebszustände EIA-485 Treiber National Semiconductor DS75176

Bei der Auflistung der Betriebszustände wurde das Signal Receiver Enable (RE) nicht berücksichtigt, da dieses Signal (negative Logik) fest auf Masse gelegt ist. Damit ist der Empfänger immer aktiv.
Das Ready to Send (RTS) Signal wird nach Spezifikation nicht am Bus benötigt. Allerdings wird es hier dazu verwendet, die Treiberrichtung zu wählen.

Da der Treiber im galvanisch getrennten Schnittstellenbereich liegt, wird das RTS Signal auch über einen Optokoppler geführt. Damit werden mögliche Störungen von Schaltungsteil ferngehalten. Der verwendete HCPL0601 invertiert das Signal, so dass es im Anschluss durch eine Transistorschaltung erneut invertiert werden muss. (siehe Bild 25)

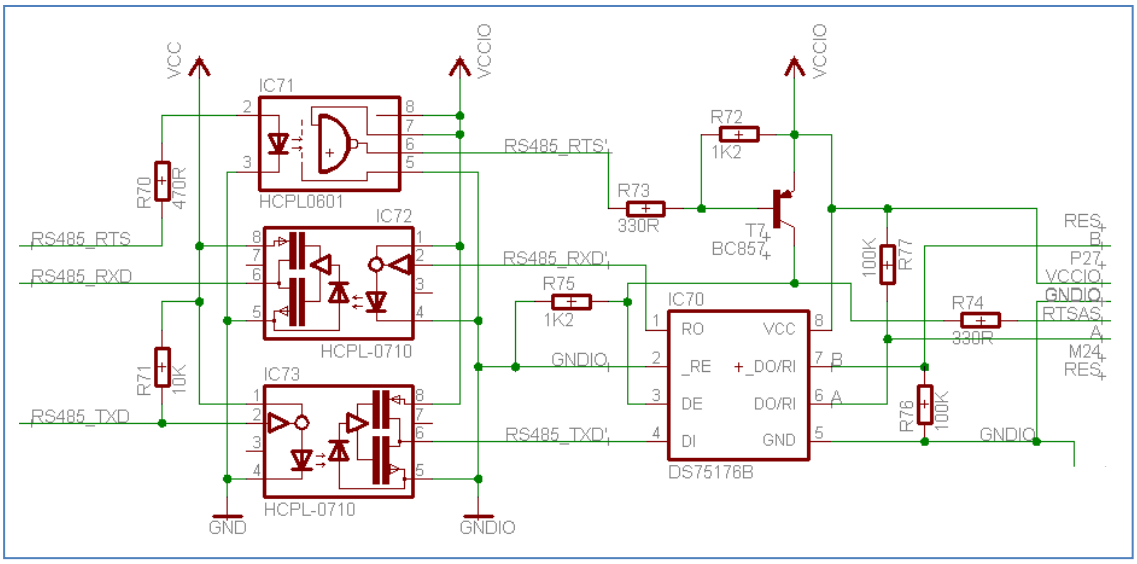


Bild 25: Beschaltung EIA-485 Treiber

Die Belegung der D-SUB Buchse ist gemäß den Anforderungen des Profibus / MPI wie folgt:

Pin	I/O	Signal
1	-	nicht belegt
2	-	nicht belegt
3	I/O	Datenleitung A
4	O	Request to Send (RTS) *1
5	-	Masse / Ground (GND)
6	-	Versorgungsspannnung (VCC)
7	-	nicht belegt
8	I/O	Datenleitung B
9	-	nicht belegt

Tabelle 10: Belegung EIA-485 Schnittstelle

*1) wird für Profibus / MPI nicht benötigt

Um an den Profibus / MPI anzukoppeln, reichen allerdings die Datenleitungen A und B sowie die Masse, die meist über den Kabelschirm verbunden wird.

5.1.2.3 Netzwerkschnittstelle [36]

Wegen der in die RJ-45 Buchse integrierten Übertrager ist für die Netzwerkschnittstelle kein großer Schaltungsaufwand nötig. (siehe Bild 26)
Die Abschlusswiderstände sind aus dem Schaltungsbeispiel des PHY übernommen worden, ebenso wie die Ableit- bzw. Stabilisierungs-Kondensatoren.

Die Signalleitungen zur Ansteuerung der LEDs sind mit 0-Ohm Serienwiderständen versehen, damit auch eine Buchse ohne LEDs verwendet werden kann und in diesem Fall keine Spannung angelegt wird.

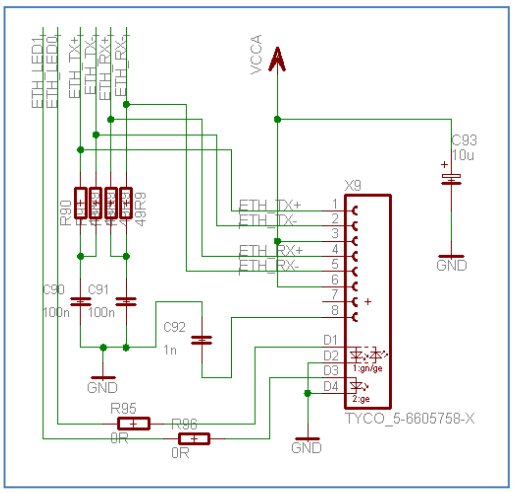


Bild 26: Netzwerkschnittstelle

5.1.2.4 CAN Schnittstelle [22, 23]

Der vorgesehene CAN Controller wird über das SPI Interface angesprochen. Außer einer Taktquelle ist keine weitere Beschaltung notwendig.

Der Treiber der CAN Schnittstelle wird über den Controller angesteuert und erzeugt das differentielle Bussignal.

Die CAN Schnittstelle selber ist über eine vierpolige Stiftleiste verfügbar. Hier sind die Datensignale High und Low sowie die Signal-Masse aufgelegt.

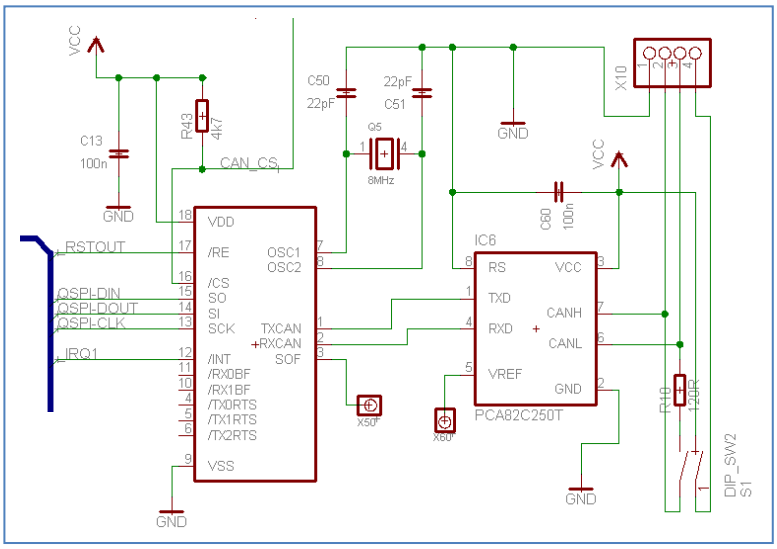


Bild 27: CAN Schnittstelle

Über einen zweipoligen DIP-Schalter (S1) kann außerdem Versorgungsspannung aufgeschaltet bzw. ein Abschlusswiderstand zwischen beide Signalleitungen geschaltet werden. (siehe Bild 27)

5.1.2.5 SD-Karten Interface [37]

Die SD-Karten werden ebenfalls über SPI angesprochen. Durch den Kartenhalter erfolgt eine einfache Zuordnung der SPI-Signale zu den Speicherkarten-Anschlüssen. (siehe Tabelle 11)

Pin	Belegung/Signal
1	Chip Select (CS)
2	Command/Data in (DIN)
3	Masse (GND)
4	Versorgungsspannung (V_{CC})
5	Takt (CLK)
6	Masse (GND)
7	Data out (DOUT)
8 -11	nicht verwendet (NC)

Tabelle 11: SPI Interface der SD-Karte

5.1.2.6 Netzteil [6]

Das Netzteil ist direkt aus der Multi-Schnittstellenkarte übernommen worden. Es wurde dabei allerdings um Schutzbeschaltung erweitert. (siehe Bild 28)

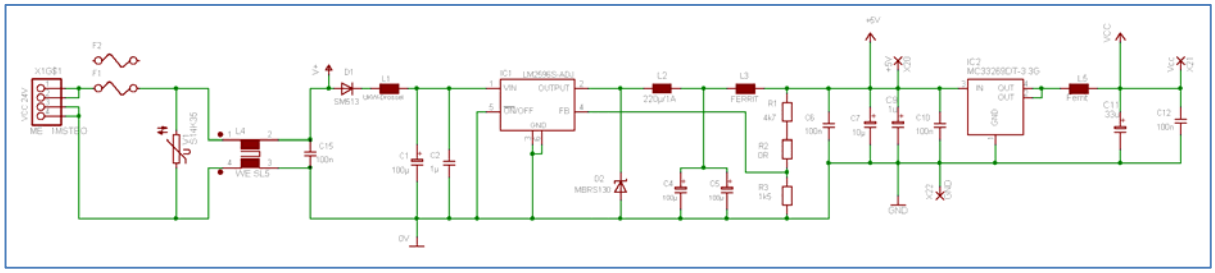


Bild 28: Netzteil-Schaltplan

Am Eingang des Netzteils wurde ein Varistor eingesetzt, der Spannungsüberhöhungen abschneidet. Dahinter ist eine stromkompensierte Drossel als Eingangs-Filter vorgesehen. Als Verpolschutz ist eine Diode (D1) in Reihe geschaltet.

Erzeugt werden vom Netzteil die internen Versorgungsspannungen mit 5 V und 3,3 V.
Die Dimensionierung der Beschaltung des Schaltreglers LM2596-ADJ wurde von der Multi-Schnittstellenkarte übernommen. Der Schaltregler erzeugt die 5 V Versorgungsspannung, aus der mit dem Linear-Regler MC33269DT-3,3G die 3,3 V Versorgungsspannung erzeugt wird. Auch hier wurde die Beschaltung von der Multi-Schnittstellenkarte übernommen.

5.1.2.7 Schnittstellenspannung

Um für die serielle Schnittstelle eine galvanisch getrennte Versorgungsspannung zur Verfügung zu haben, sind zwei DC-DC Umsetzer vorgesehen, von denen nur einer – je nach Verfügbarkeit – bestückt wird. (siehe Bild 29)

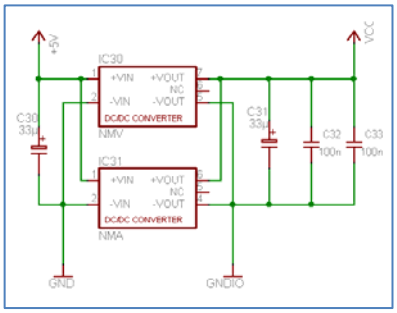


Bild 29: DC-DC Umsetzer (Schnittstellenspannung)

5.2 PCB Layout [38, 39]

Die Platinenmaße für das Printed Circuit Board (PCB) sind vom Gehäusehersteller Phoenix Contact passend zum gewählten Gehäuse vorgegeben. (siehe Bild 30) Es steht pro Platine eine Bestückungsfläche (einseitig) von rund 80 cm^2 zur Verfügung.

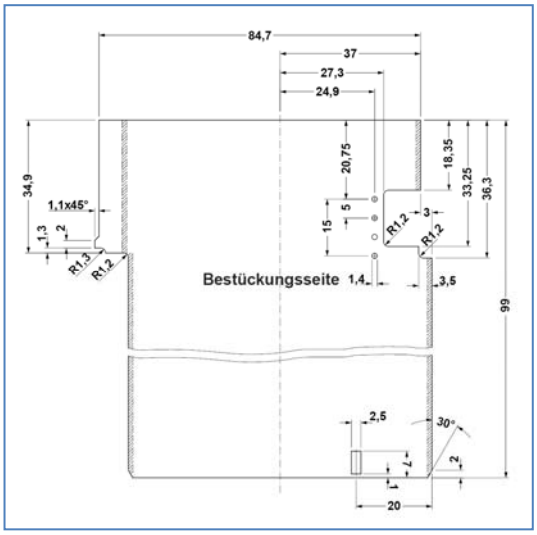


Bild 30: Platinenmaße

Die Platine wurde gemäß der vorgegebenen Maße als Package in Eagle angelegt. Die eingezeichneten Sperrflächen für die Platinenaufnahmen wurden ebenso übernommen. Außerdem wurden für den Bereich der Lüftungsschlitze im Gehäuse weitere Sperrflächen definiert, damit hier kein Kontakt zur Schaltung durch Einführen eines Gegenstandes hergestellt werden kann.

Bei der Positionierung der Bauteile sowie der Leiterbahnführung auf dem Board sind außer den Sperrflächen noch weitere Dinge zu berücksichtigen.
Alle Bauteile sollten soweit möglich in Baugruppen dicht beieinander platziert werden. So wird eine aufwändige Leiterbahnführung insbesondere über größere Distanzen vermieden. Damit wiederum bietet das Layout weniger Angriffsfläche für einstrahlende Störungen. Zusammengehörige Leitungen, wie z.B. Adress- oder Datenbus, sollten parallel geführt werden, um Probleme im Signaltiming durch unterschiedlich lange Verbindungen auszuschließen. Auch bei den dazugehörigen Steuerleitungen ist auf die Leiterbahnlänge zu achten.
Bauteile mit möglicher Störausstrahlung, wie z.B. Quarze oder Oszillatoren, sollten so positioniert werden, dass die entsprechenden Signalleitungen so kurz wie möglich gehalten werden können.

Versorgungsanschlüsse sollten – falls es entsprechende Versorgungslayer gibt – ebenfalls kurz und niederohmig angebunden werden. Bei Verwendung von Abblock-Kondensatoren an den Versorgungsanschlüssen ist darauf zu achten, dass die Versorgungsleitungen durch die Anschlüsse des Kondensators zum Bauteil geführt werden. (siehe Bild 31) Liegen die Anschlüsse für Versorgungsspannung (VCC) und Masse (GND) nicht passend nebeneinander, sollte nur die Versorgungsspannung über den Kondensator geführt werden.

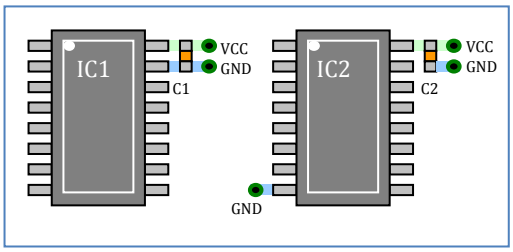


Bild 31: Anschluss von Versorgungsleitungen mit Abblock-Kondensatoren

5.2.1 CPU-Platine [10]

Auf der CPU-Platine sind die Baugruppen CPU und Peripherie, Schnittstellen-Interfaces sowie User-Interface untergebracht. Die grobe Lage dieser Baugruppen ergibt sich hier automatisch. Die Schnittstellen-Interfaces und das User-Interface müssen auf der oberen Platinenhälfte platziert werden, damit der Reset-Taster, die Status LEDs und die Schnittstellen an sich am Gehäusedeckel

herausgeführt werden können. Damit bleibt für den Prozessor und die Peripherie die untere Platinenhälfte.

Für den Prozessor als Kern-Bauteil der Platine wurde eine Skizze angefertigt, die Auskunft über die Position und Verwendung der Anschlüsse gibt. (siehe Bild 32)

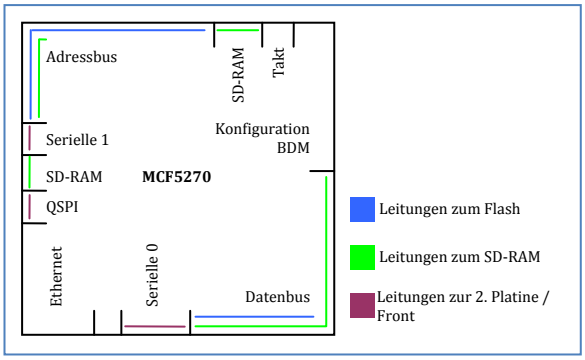


Bild 32: Skizze der Prozessoranschlüsse

Aus allen genannten Vorüberlegungen resultierte folgender Lageplan für die CPU-Platine:

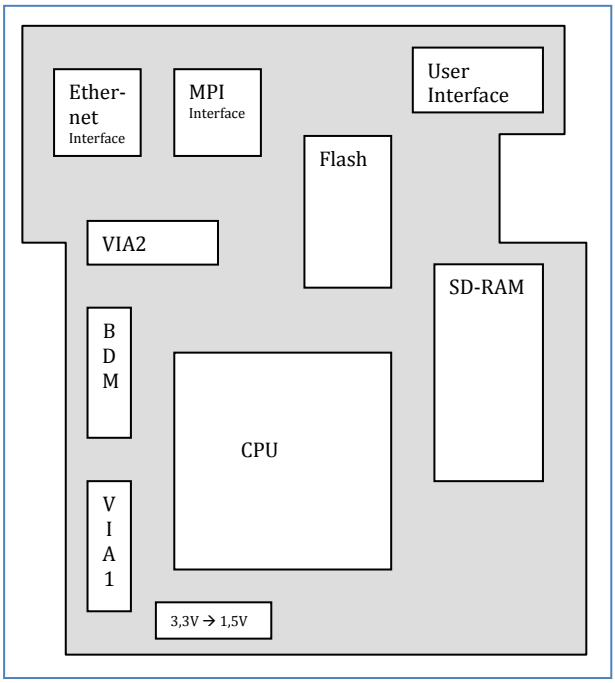


Bild 33: Baugruppen-Lageplan CPU-Platine

Die Position der Platinenverbindungen (VIA1 und VIA2) und des BDM Programmierinterfaces ergaben sich nach Positionierung aller anderen Baugruppen. Unter Berücksichtigung des freien Platzes und der entsprechenden Signale wurden diese gesetzt.

5.2.2 Netzteilplatine

Auch für die Netzteilplatine ergab sich die grobe Positionierung der Baugruppen weitestgehend automatisch. So muss das Netzteil an der Seite des Steckverbinders und die Schnittstellen an der oberen Kante untergebracht werden.

Die restliche Aufteilung ergab sich aus der Position der Leiterplattenverbindungen (VIA1, VIA2) und dem Aufbau der verbleibenden Baugruppe QSPI. (siehe Bild 34)

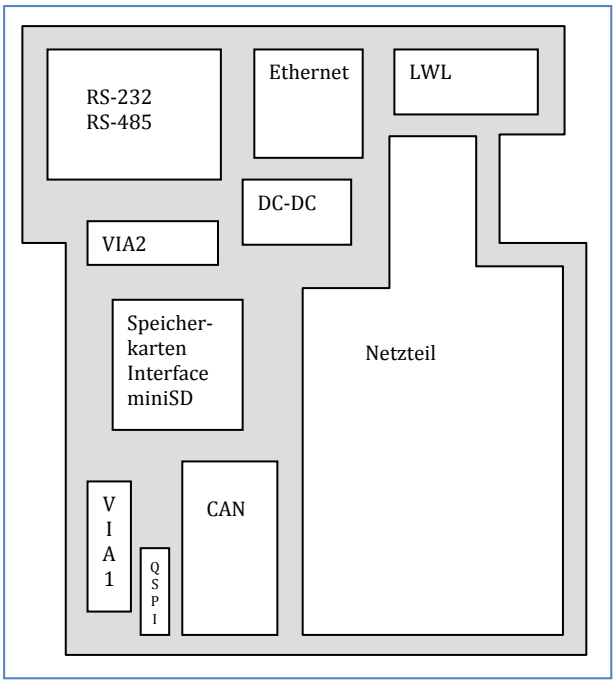


Bild 34: Baugruppen-Lageplan Netzteilplatine

5.3 Geräte-Aufbau [28, 29, 40]

Die einzelnen Platinen sind so im Gehäuse untergebracht, dass die Bauteile der Netzteilplatine zur Gehäuse-Außenseite zeigen. Damit wird die auf der anderen Seite befindliche CPU Platine besser vor Einstrahlungen aus dem Netzteil geschützt. (siehe Bild 35)

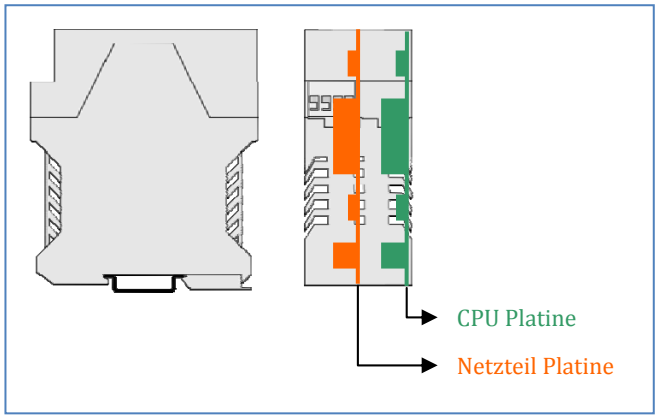


Bild 35: Innerer Aufbau des KNG

Aus der Lage der Platinen ergibt sich für die Gehäuse-Oberseite folgende Aufteilung:

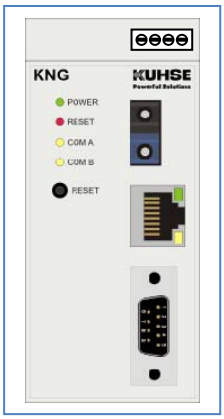


Bild 36: KNG Oberseite mit User-Interface und Schnittstellen

Für die internen Platinenverbindungen VIA1 und VIA2 ergibt sich aus der Aufteilung der Bauteile beider Platinen folgende Belegung:

Pin	Belegung VIA1	Belegung VIA2
1	Versorgungsspannung (VCC)	Ethernet Sendeleitung + (ETH_TX+)
2	Versorgungsspannung (VCC)	Ethernet Empfangsleitung - (ETH_RX-)
3	Masse (GND)	Ethernet Sendeleitung - (ETH_TX-)
4	Masse (GND)	Ethernet Empfangsleitung + (ETH_RX+)
5	Interrupt Leitung 1 (IRQ1)	Masse (GND)
6	Chip Select 6 (CS6)	Masse (GND)
7	Chip Select 1 (CS1)	LWL Empfangsleitung (U1RXD)
8	COM Signal RTS (U0RTS)	LWL Sendeleitung (U1TXD)
9	COM Signal CTS (U0CTS)	Ethernet LED1 (ETH_LED1)
10	COM Sendeleitung (U0TXD)	Ethernet LED0 (ETH_LED0)
11	COM Empfangsleitung (U0RXD)	Masse (GND)
12	Signal DTOUT0 (QSPI-CS)	Masse (GND)
13	Signal DTIN0 (QSPI-CS)	EIA-485 Signal RTS (RS485_RTS)
14	Signal DTIN2 (COM DCD / U0DCD)	EIA-485 Sendeleitung (RS485_TXD)
15	Masse (GND)	EIA-485 Empfangsleitung (RS485_RXD)
16	Masse (GND)	Masse (GND)
17	SPI Empfangsleitung (QSPI-DIN)	Masse (GND)
18	SPI Sendeleitung (QSPI-DOUT)	Masse (GND)
19	SPI Chip Select 0 (QSPI-CS0)	Masse (GND)
20	SPI Takt (QSPI-CLK)	Reset vom Prozessor (RSTOUT)

Tabelle 12: Singalbelegung Platinenverbindungen VIA1 und VIA2

Für die Platinenverbindung sind Stift- und Buchsenleisten der Firma Fischer ausgewählt worden. Da fertigungstechnisch keine ausreichend hohen Stiftleisten (ca. 25 mm) maschinell verarbeitet werden können, wurde eine Dreiteilung der Verbindungen vorgenommen. (siehe Bild 37)
Auf beiden Platinen sind flache Buchsenleisten vorgesehen, wobei die der Netzteilplatine von unten steckbar sind. Hier sind entsprechende Bohrungen in der Platine vorgesehen. Eine Stiftleiste als Verbindung wird bei der Montage dazwischen gesteckt.

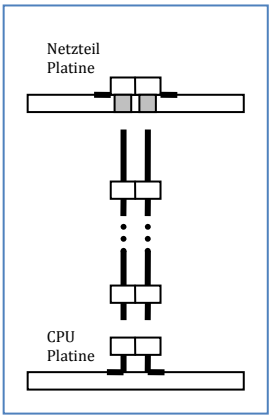


Bild 37: Platinen-verbindungen

5.4 Kosten

Die reinen Bauteilkosten für ein voll bestücktes KNG liegen bei rund 110 Euro. Daraus resultiert ein Produktionspreis von rund 150 Euro.
Die Preisvorstellungen aus der Spezifikation können somit erfüllt werden.

6 Realisierung der Software

6.1 Grundlagen

6.1.1 Entwicklungsumgebung (Toolchain) [41, 42]

Für die Entwicklung der Gerätesoftware (Firmware) wurde die unter GNU Public License (GPL) stehende und damit kostenfrei einsetzbare GNU Toolchain als Software ausgewählt. Die in dieser Toolchain zusammengefassten Open Source Entwicklungstools stammen ursprünglich aus der Linux Welt und basieren auf dem GNU C-Compiler (GCC). Es stehen Versionen für viele Zielplattformen, wie z.B. i386, m68k oder ARM zur Verfügung.
Für den Coldfire Prozessor wird die Version für Motorola 68000 Plattformen (m68k) benötigt. Diese wird von der Firma CodeSourcery als kompilierte MS Windows Version mit Installationsprogramm zum Download angeboten.

Die GNU Toolchain (für m68k Zielplattformen) besteht hauptsächlich aus folgenden Programmen:

Name	Programm
m68k-elf-gcc.exe	GNU C-Compiler
m68k-elf-g++.exe	GNU C++ Compiler
m68k-elf-as.exe	Assembler
m68k-elf-ld.exe	Linker
m68k-elf-gdb.exe	GNU Debugger

Tabelle 13: Hauptbestandteile der GNU Toolchain

Darüber hinaus gehören weitere Programme, die allerdings nicht verwendet wurden, und Standard-Bibliotheken, die bei Bedarf eingebunden werden, zur Toolchain.
Alle Programme sind WIN-32 Kommandozeilen Tools.

Es besteht die Möglichkeit, die Open-Source Software Eclipse der Eclipse Foundation als Integrierte Entwicklungsumgebung (IDE - integrated development environment) unter MS Windows zu verwenden. Eclipse vereint und verwaltet die Kommandozeilen Tools unter einer grafischen Oberfläche und bietet einen Texteditor für den Quellcode mit Syntax Highlighting.
Da Eclipse als IDE für verschiedene Systeme und Programmiersprachen einsetzbar ist, muss die Software für den Einsatz mit der GNU Toolchain für m68k Zielsysteme speziell angepasst werden. Die Firma CodeSourcery bietet auch ein Paket der GNU Toolchain inklusive Eclipse IDE an. Dieses Paket ist allerdings nicht kostenfrei erhältlich, da hier für die Anpassung der IDE bezahlt werden muss.

Für die Entwicklung der KNG Firmware wurde auf die kostenfreie Toolchain ohne IDE zurückgegriffen. Als Quellcode-Editor wurde UltraEdit-32 der Firma IDM Computer Solutions verwendet. Dieser Editor bietet auch Syntax Highlighting.
Für die meisten wiederkehrenden Abläufe wurden entsprechende Batch-Dateien erstellt, so dass nur wenig direkt auf der Kommandozeile gearbeitet werden musste.

6.1.2 Demo-Quellcode [43]

Im Bereich MCF527X der Freescale Webseiten steht ein Demo-Programmpaket für den MCF5272 zum kostenfreien Download und frei zur Weiterverwendung zur Verfügung. Neben einem Initialisierungs-/Boot-Programm sind folgende Demo-Programme enthalten:

- Einfaches Testprogramm mit Interrupt Funktionen
- Einfaches Programm mit Tests für CE Zertifizierung
- Einfaches Netzwerkprogramm (interne Sende-Empfangs-Schleife)
- Netzwerkprogramm mit TFTP Server
- USB Testprogramm (Kommunikation mit MS Windows Human Interface Device (HID) Treiber)

Der Quellcode der Netzwerkprogramme wurde als Grundlage für die KNG Software verwendet. Dabei wurde zunächst das Programm mit der internen Sende-Empfangs-Schleife sowie des Boot-Programm für den MFC5270 umgesetzt. Auf diesem Stand wurde dann die weitere KNG Firmware aufgebaut.

6.1.3 Ethernet Protokollfamilie

Für die Kommunikation über ein Ethernet Netzwerk gibt es verschiedene Protokolle, die je nach Anwendung eingesetzt werden. Diese werden zusammenfassend oft als „TCP/IP Protokollfamilie“ bezeichnet, obwohl sowohl das TCP (Transmission Control Protocol) als auch das IP (Internet Protocol) als eigenständiges Protokoll zur Familie gehören.

6.1.3.1 OSI/ISO Modell [44]

Die Ethernet Protokollfamilie lässt sich mit Hilfe des OSI (Open Systems Interconnection) / ISO (International Organization for Standardization) Modells wie folgt abbilden:

OSI Schicht	Ethernet Schicht	Protokolle (Beispiele)
7 Anwendung (Application)	Anwendung	HTTP SMTP KUHSE Telegramm
6 Darstellung (Presentation)		
5 Sitzung (Session)		
4 Transport	Transport	UDP, TCP
3 Vermittlung (Network)	Netzwerk	IP, ICMP
2 Sicherung (Data Link)	Netzzugang	Ethernet, ARP
1 (Bit-)Übertragung (Physical)		

Tabelle 14: OSI Modell für Ethernet Protokolle

Unter den aufgeführten Beispielen sind nur Protokolle genannt, die für das Netzwerk Gateway relevant sind. Insgesamt gibt es ca. 500 Protokolle, die hier eingeordnet werden können.

6.1.3.2 Ethernet Protokoll [44]

Das eigentliche Ethernet Protokoll bildet als Netzzugangsprotokoll die Grundlage der Ethernet Kommunikation.

Ethernet-Telegramme sind wie folgt aufgebaut:

Präambel [7 Bytes]	**Startzeichen** (SFD) [1 Byte]	**Ziel-Adresse** (MAC Adresse) [6 Bytes]	**Quell-Adresse** (MAC Adresse) [6 Bytes]	**VLAN Tag** (TPID, TCI) [4 (2, 2) Bytes]	**Typ** [2 Bytes]	**Daten** [0–1500 Bytes]	**Füllfeld** [0–1500 Bytes]	**Prüfsumme** (CRC) [4 Bytes]

Tabelle 15: Ethernet Telegrammaufbau

Die Präambel leitet ein Ethernet Telegramm ein. Sie besteht aus einer Folge von sieben Bytes, die jeweils den Hexadezimalwert 0x55 haben müssen.
Danach folgt das Startzeichen (SFD – Start of Frame Delimiter) mit dem Wert 0xD5.
Die folgenden Adressierungsinformationen bestehen aus Ziel- und Quelladresse, die jeweils in Form einer sechs Byte langen MAC (Media Access Control) Adresse angegeben ist. Diese Hardware-Adressen müssen weltweit eindeutig sein. Dafür können Gerätehersteller vom IEEE (Institute of Electrical and Electronics Engineers) einen Adressraum kaufen.
Das VLAN (Virtual Local Area Network) Tag ist ein optionaler Block, der ergänzt wurde, um große Netzwerke besser strukturieren zu können.
Über das Typ-Feld wird definiert, welches Protokoll in den Nutzdaten des Ethernet Telegramms und damit in der folgenden Schicht (Netzwerk) verwendet wird.
Das Datenfeld enthält ein Telegramm der Netzwerk-Schicht, das bis zu 1500 Bytes lang sein darf.
Da die minimale Länge eine Ethernet Telegramms ohne Präambel und Startzeichen 64 Bytes beträgt, muss je nach Datenlänge das Füllfeld verwendet werden, um diese Länge zu erreichen. Gefüllt wird es typischerweise mit 0-Bytes.
Die Prüfsumme ist eine 32-Bit CRC (cyclic redundancy check) Prüfsumme, die über das Telegramm, beginnend bei der Ziel-Adresse bis einschließlich Füllfeld, berechnet wird.

6.1.3.3 Address Resolution Protocol (ARP) [45, 46]

Das Address Resolution Protocol (ARP) wird der Netzzugangsschicht zugeordnet, obwohl es auf dem Ethernet Protokoll aufbaut. Es dient allerdings der Ermittlung von Ethernet Adressen aufgrund von IP Adressen. Spezifiziert ist ARP in der RFC 826.

ARP Telegramme sind wie folgt aufgebaut:

Bits **Byte Offset**	**0-7**	**8-15**	**16-23**	**24-31**
0	Hardware-Adresstyp		Protokoll-Adresstyp	
4	Hardware-Adressgröße	Protokoll-Adressgröße	Operation	
8	Quell-Hardware-Adresse, Quell-IP-Adresse, Ziel-Hardware-Adresse, Ziel-IP-Adresse			

Tabelle 16: ARP Telegrammaufbau

Wie aus dem Aufbau ersichtlich ist, sind ARP Telegramme nicht auf die Verwendung in Ethernet Netzwerken beschränkt. Für die Verwendung in Ethernet Netzwerken mit IP als Netzwerkprotokoll sind der Hardware-Adresstyp immer 0x0001 und der Protokoll-Adresstyp 0x0800. Die Hardware-Adressgröße beträgt 6 und die Protokoll-Adressgröße bei IPv4 beträgt 4. Alle Quell- und Zieladressen

sind entsprechend der Liste in der Tabelle hintereinander abgelegt, die jeweiligen Positionen ergeben sich aus den angegebenen Größen.
Im Operationsfeld wird übermittelt, ob es sich um eine Adressanfrage oder eine Antwort handelt.

6.1.3.4 Internet Protokoll (IP) [47, 48]

Das Internet Protokoll (IP) Version 4 hat folgenden Telegrammaufbau:

Bits Byte Offset	0-3	4-7	8-11	12-15	16-18	19-23	24-27	28-31
0	Version	Header-Länge	Service (TOS – Type of Service)		Gesamtlänge			
4	Identifikation				Flags	Fragment Offset		
8	Gültigkeitsdauer (TTL – Time to Live)		Protokoll		Prüfsumme			
12	Quell-Adresse (IP Adresse)							
16	Ziel-Adresse (IP Adresse)							
20	ggf. weitere Optionen / Daten							

Tabelle 17: IPv4 Telegrammaufbau

Der dargestellte Telegrammaufbau zeigt den Aufbau eines IPv4 Telegramms, das Netzwerk-Teilnehmer mittels einer vier Byte langen sogenannten IP Adresse identifiziert. Definiert wird das verwendete Format über das Versions-Feld, das in diesem Fall den Wert 0x4 enthalten muss.
Das noch relativ neue IPv6 Protokoll mit entsprechend sechs Byte langen Adressen wird bisher vom KNG nicht unterstützt.
Die Verwendung des TOS-Feldes wurde in der Vergangenheit mehrfach verändert – zuletzt im Jahr 2001. Generell wird es genutzt um die Priorität eines Telegramms zu steuern. Aktuell gibt es folgende Parameter:

Bits 7-6 ECN (Explicit Congestion Notification) zur IP Flusskontrolle
Bits 5-0 DSCP (Differentiated Services Code Point)

Das Identifikationsfeld wird mit den Flags und dem Fragment Offset verwendet, um fragmentierte Telegramme wieder zusammenzusetzen.
Fragmentierte Telegramme werden vom KNG nicht unterstützt, da die Datenlänge von maximal 1500 Bytes pro Ethernet Telegramm für die Anwendung ausreicht und außerdem das Handling fragmentierter Telegramme einen hohen Speicherbedarf und Programmaufwand benötigt.
Die Gültigkeitsdauer wird nicht mehr wie ursprünglich in Sekunden sondern als maximale Anzahl von Zwischenstationen (Hops), z.B. Router etc., für den Telegrammweg angegeben.
Über das Protokollfeld wird definiert, welches Folgeprotokoll in der nächsten Schicht (Transport) verwendet wird.
Die Prüfsumme wird ausschließlich über den Header (Telegrammkopf) gebildet. Dabei werden die Daten in 16 Bit Blöcken addiert und das Einerkomplement vom Ergebnis verwendet. Zur Generierung der Prüfsumme wird diese mit dem Wert 0 mit in die Berechnung einbezogen.
Vom KNG wird die Möglichkeit weitere Optionen im Header unterzubringen nicht verwendet.

6.1.3.5 Dynamic Host Configuration Protocol (DHCP) [49, 50]

Das Dynamic Host Configuration Protocol (DHCP) wird dazu eingesetzt, um in einem Netzwerk servergesteuert IP Adressen zu vergeben. Obwohl DHCP damit zur Netzwerkschicht gehört, werden die Telegramme per UDP versendet. DHCP ist für IPv4in RFC 2131 spezifiziert.

Die Telegramme haben folgenden Aufbau:

Bits Byte Offset	0-7	8-15	16-23	24-31
0	Operation	Hardware-Adresstyp	Hardware-Adresslänge	Anzahl der Vermittlungsstellen
4	ID			
8	Zeit		Flags	
12	Client (IP-Adresse)			
16	Eigene IP-Adresse			
20	Server (IP-Adresse)			
24	Vermittlungsstelle (IP-Adresse)			
28	Client (Hardware-Adresse)			
44	Server-Name (optional)			
108	Boot-Datei (Skript, optional)			
236	Optionen (optional)			

Tabelle 18: DHCP Telegrammaufbau

Das Operationsfeld gibt an, ob es sich bei dem Telegramm um eine Anfrage oder eine Antwort vom Server handelt.
Der Hardware-Adresstyp für Ethernet Netzwerke ist immer 0x01 und die Länge entsprechend sechs.
Die Anzahl der Vermittlungsstellen ist für größere Netzwerke relevant, die in Abschnitte unterteilt sind. Im KNG wird dieses Feld nicht verwendet.
Die ID ist frei wählbar und ermöglichet dem anfragenden Client eine Antwort eindeutig seiner Anfrage zuzuordnen.
Das Zeit-Feld ist im Ursprung dazu vorgesehen, die Zeit seit der ersten Anfrage bei Folgetelegrammen mit zu versenden. Aktuell wird das Feld als reiner Telegrammzähler verwendet.
DHCP unterstützt bisher nur ein Flag (MSB), über das die DHCP Antwort als Broadcast-Telegramm angefordert werden kann.
Alle folgenden Felder werden je nach Telegrammtyp und Verhandlungsstatus ausgefüllt.

6.1.3.6 Internet Control Message Protocol (ICMP) [51, 52]

Mit Hilfe des Internet Control Message Protocol (ICMP) können definierte Nachrichten versendet werden. Spezifiziert ist ICMP in der RFC 792.

ICMP Telegramme sind wie folgt aufgebaut:

Bits Byte Offset	0-7	8-15	16-23	24-31
0	Telegrammtyp	Telegrammcode	Prüfsumme	
4	Daten (optional)			

Tabelle 19: ICMP Telegrammaufbau

Folgende Typen und Codes sind spezifiziert:

Typ		Code	
0	Echo Antwort (Ping)	0	-
3	Ziel nicht erreichbar	0	Netzwerk nicht erreichbar
		1	Host nicht erreichbar
		2	Protokoll nicht erreichbar
		3	Port nicht erreichbar
		4	Fragmentierung erforderlich
		5	Routing nicht möglich
4	Quelle stoppen	0	Telegramm verworfen, da Puffer voll
5	Weiterleitung	0	Weiterleitung für Netzwerk
		1	Weiterleitung für Host
		2	Weiterleitung für TOS und Netzwerk
		3	Weiterleitung für TOS und Host
8	Echo Anfrage (Ping)	0	-
11	Timeout	0	TTL (IP Header) abgelaufen
		1	Timeout für Fragment(e) erreicht
12	Parameter Problem	0	Zeiger auf Fehler im Datenfeld

Tabelle 20: ICMP Typen und Codes

Das KNG verarbeitet nur die ICMP Echo (Ping) Telegramme. Damit kann festgestellt werden, ob eine bestimmte IP Adresse vergeben ist.

6.1.3.7 User Datagram Protocol (UDP) [53, 54]

Das User Datagram Protocol (UDP) als Transportprotokoll bildet die Schnittstelle zur Anwendungsebene. Es ist daher nicht notwendig, ein weiteres Folgeprotokoll zu verwenden. Die Daten können direkt über das UDP Telegramm versendet werden.
Um eine Kommunikation unterschiedlicher Anwendungen über eine Ethernet Verbindung zu ermöglichen, werden den Anwendungen bestimmte Ports zugewiesen. In der Praxis wird von den Anwendungen trotzdem ein weiteres Protokoll verwendet, das somit als Folgeprotokoll anzusehen ist. Dieses ist dann an den jeweiligen Port gekoppelt. Es ist auch möglich, dass eine Anwendung mehrere Anwendungsprotokolle und damit mehrere Ports verwendet.
Spezifiziert ist UDP in RFC 768.

UDP Telegramme sind wie folgt aufgebaut:

Bits Byte Offset	0-3	4-7	8-11	12-15	16-18	19-23	24-27	28-31
0	Quell-Port				Ziel-Port			
4	Länge				Prüfsumme			
8	Daten							

Tabelle 21: UDP Telegrammaufbau

Das Längenfeld des UDP Telegramms enthält die Gesamtlänge des Telegramms, bestehend aus Header und Daten, in Bytes.
Die Prüfsumme ist optional. Falls sie nicht verwendet werden soll, muss der Wert 0x0000 in das Feld eingetragen werden. Wird sie verwendet, erfolgt die Berechnung wie beim IP Telegramm. Allerdings wird bei der Berechnung hier ein verkürzter IP Pseudo-Header mit einbezogen, der für IPv4 wie folgt aufgebaut ist:

Bits Byte Offset	0-3	4-7	8-11	12-15	16-18	19-23	24-27	28-31
0	Quell-Adresse (IP Adresse)							
4	Ziel-Adresse (IP Adresse)							
8	0x00		0x17 (UDP Protokoll ID)		UDP Telegrammlänge			

Tabelle 22: UDP – IPv4 Pseudo-Header (Aufbau)

6.1.3.8 Transport Control Protocol (TCP) [55, 56]

Das Transport Control Protocol (TCP) bildet ebenso wie das UDP die Schnittstelle zur Anwendungsebene. Auch hier werden Portnummern wie bei UDP verwendet und bestimmten Folgeprotokollen sind bestimmte Ports zugeordnet.
Vorteil von TCP Übertragungen ist, dass Mechanismen zur Verbindungskontrolle in den Telegrammen eingebunden sind.
Spezifiziert ist TCP in RFC 793.

Ein TCP Telegramm hat folgenden Aufbau:

Bits Byte Offset	0-3	4-7	8-11	12-15	16-18	19-23	24-27	28-31
0	Quell-Port				Ziel-Port			
4	Sequenznummer							
8	Quittierungsnummer (Acknowledgement Number)							
12	Daten-Offset	Reserviert	Flags		Fenstergröße			
16	Prüfsumme				Urgent Pointer			
20	ggf. weitere Optionen / Daten							

Tabelle 23: TCP Telegrammaufbau

Die Verwendung der Ports ist beim TCP Telegramm ebenso wie beim UDP Telegramm.
Sequenznummer und Quittierungsnummer dienen der Verbindungskontrolle. Die Sequenznummer wird beim Aufbau einer Verbindung gewählt. Mit der Quittierungsnummer wird der Gegenstelle übermittelt, welche Sequenznummer als nächstes erwartet wird. So kann jeweils der Empfänger entscheiden, ob das empfangene Telegramm gültig ist, und entsprechend reagieren.
Über den Daten-Offset wird die Länge des TCP Headers in 4-Byte Blöcken angegeben. Werden keine weiteren Optionen übermittelt, muss hier der Wert 0x5 stehen.
Das Flags-Feld des TCP Telgramms besteht aus folgenden sechs 1-Bit Flags:

URG	ACK	PSH	RST	SYN	FIN

Tabelle 24: TCP Flags

Mit dem gesetzten Urgent-Flag (URG) wird dem Empfänger die Verwendung des Urgent-Pointers angezeigt.
Über das gesetzte Quittierungs-Flag (ACK) wird die Gültigkeit der Quittierungsnummer angezeigt. Es ist so möglich, die Verbindungskontrolle zu unterbinden. Beim Verbindungsaufbau dient dieses Flag zur Bestätigung.
Wird das Push-Flag (PSH) gesetzt, so wird der Empfänger angewiesen, das Telegramm sofort zu verarbeiten und nicht zwischenzuspeichern (puffern).
Das Reset-Flag wird dazu verwendet, eine Verbindung abzubrechen.
Mit dem Synchronisations-Flag (SYN) wird ein neuer Verbindungsaufbau angezeigt. Dagegen wird mit dem Beendigungs-Flag (FIN) angezeigt, dass die Verbindung beendet werden soll.

Die Fenstergröße gibt an, wie lang ein TCP Telegramm sein darf, das an den Sender zurück gesendet wird. Der Wert umfasst dabei die Länge des Headers ab der Quittierungsnummer (min. 12 Bytes) und die Datenlänge.
Auch die Berechnung der Prüfsumme des TCP Telegramms erfolgt wie beim IP Telegramm. Die Summe wird über das gesamte Telegramm (Header und Daten) gebildet.
Der Urgent-Pointer (Zeiger) zeigt auf Daten innerhalb des Telegramms, die vom Empfänger vorrangig bearbeitet werden sollen. Zur Verwendung des Zeigers muss das entsprechende Flag gesetzt sein.

Vom KNG wird die Möglichkeit weitere Optionen im Header unterzubringen nicht verwendet.

6.1.3.9 Hypertext Transfer Protocol (HTTP) [57, 58, 59]

Das Hypertext Transfer Protocol (HTTP) ist ein textbasiertes Protokoll, das dazu verwendet wird, Webinhalte (HTML-Seiten, Bilder, etc.) oder auch beliebige Daten über ein Netzwerk zu übertragen.

Obwohl für den HTTP Datenaustausch keine dauerhafte Verbindung benötigt wird, da es sich um ein simples Anfrage-Antwort-Protokoll handelt, wird zur Übertragung immer eine TCP Verbindung (Port 80) verwendet. In HTTP Version 1.0 (RFC 1945) wird dabei sogar für jedes übertragene Element eine neue Verbindung aufgebaut. In der neueren Version 1.1 (RFC 2616) dagegen können mehrere Elemente nacheinander über eine aufgebaute Verbindung übermittelt werden.

Da es sich um ein textbasiertes Protokoll handelt, gibt es für http Telegramme keine feste Struktur. Vielmehr gibt es einen Befehlssatz für verschiedene Arten der Datenanfrage. (siehe Tabelle 25) Die entsprechende Antwort benötigt keinen Befehl, allerdings gibt es verschiedene Parameter, die übertragen werden können.

Befehl	**Bedeutung**
GET	Anforderung von Inhalten / Daten (häufigster Befehl)
POST	auch Anforderung von Daten, allerdings werden zusätzlich Parameter übermittelt
HEAD	nur Anforderung der Header-Daten eines Dokuments
PUT	Hochladen von Daten zum Server (nicht mehr verwendet)
DELETE	Löschen von Daten auf dem Server (nicht mehr verwendet)
TRACE	Anfrage wird vom Server so zurückgesendet, wie sie empfangen wurde (Einsatz zum Debugging einer Verbindung)
OPTIONS	Anforderung der vom Server akzeptierten Parameter
CONNECT	Aufbau eines sicheren SSL Tunnels (Proxy-Server)

Tabelle 25: HTTP Befehle

Beispiel einer Anfrage der Seite http://www.google.de mit Antwort vom Server:

```
GET / HTTP/1.1
Host: www.google.de
Connection: close
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: de-de,de;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; de; rv:1.8.1.9) Gecko/20071025 Firefox/2.0.0.9

  HTTP Status Code: HTTP/1.1 200 OK
  Cache-Control: private
  Content-Type: text/html; charset=UTF-8
  Set-Cookie: PREF=ID=6c0cf3ed1634dc5d:TM=1195027161:LM=1195027161:S=ipLog_pGsbNya3YT;
              expires=Fri, 13-Nov-2009 07:59:21 GMT; path=/; domain=.google.de
  Server:        gws
  Transfer-Encoding: chunked
  Date: Wed, 14 Nov 2007 07:59:21 GMT
  <html><head><meta http-equiv="content-type" content="text/html; charset=UTF-8"><title>Google</title>
  <style>body,td,a,p,.h{font-family:arial,sans-serif}.h{font-size:20px}.h{color:#3366cc}.q{color:#00c}
  .ts td{padding:0}.ts{border-collapse:collapse}</style>
  [...]
  <font size=-1><a href="/intl/de/ads/">Werbeprogramme</a> -
  <a href="/services/">Unternehmensangebote</a> -
  <a href="/intl/de/about.html">Über Google</a> -
  <a href=http://www.google.com/ncr>Google.com in English</a></font>
  <p><font size=-2>&copy;2007 Google</font></p></center>
  </body></html>
```

Bild 38: HTTP Anfrage und Antwort

6.1.3.10 Simple Mail Transfer Protocol (SMTP) [60, 61, 62, 63]

Das Simple Mail Transfer Protocol (SMTP) dient dazu, E-Mails über ein Netzwerk zu versenden. Es ist wie das HTTP ein textbasiertes Protokoll und wird ebenso über eine TCP Verbindung (Port 25) übertragen. Spezifiziert ist das Protokoll ursprünglich in RFC 821 und überarbeitet in RFC 2821)

Zusätzlich zum Textinhalt wird vom Mailserver (SMTP-Server) mit jedem Telegramm ein Statuscode übertragen. Die Codes sind immer dreistellig und wie folgt geschlüsselt:

Nummer	Bedeutung
1xx	Anforderung akzeptiert aber Bestätigung erforderlich
2xx	Anforderung akzeptiert und ausgeführt
3xx	Anforderung akzeptiert aber weitere Daten erforderlich
4xx	Anforderung mit temporärem Fehler erhalten (erneute Übermittlung möglicherweise ausreichend)
5xx	Anforderung mit fatalem Fehler erhalten (nicht zu verarbeiten)

Tabelle 26: SMTP Statuscodes

Beispiel eines E-Mail-Versands von m.anetzberger@kuhse.de an anetzberger@gmx.de mit dem Inhalt „Test“ über das Mailprogramm MS Outlook 2003:

```
[TCP Verbindungsaufbau]
  220 mail.kuhse.de ESMTP Sendmail 8.9.3/8.9.3; Wed, 14 Nov 2007 11:07:28 +0100
EHLO manetzberger
  250-mail.kuhse.de Hello pc_1_142.kuhse.de [192.168.100.142], pleased to meet you
  250-8BIT MIME
  250-SIZE
  250-DSN
  250-ONEX
  250-ETRN
  250-XUSR
  250 HELP
MAIL FROM: <m.anetzberger@kuhse. de>
  250 <m.anetzberger@kuhse.de>... Sender ok
RCPT TO: <anetzberger@gmx.de>
  250 <anetzberger@gmx.de>... Recipient ok
DATA
  354 Enter mail, end with "." on a line by itself
From: "Marco Anetzberger" <m.anetzberger@kuhse.de>
To: "Marco Anetzberger" <anetzberger@gmx.de>
Subject: Test..Date: Wed, 14 Nov 2007 11:07:06 +0100
Message-ID: <000901c826a6$2927cd00$8e64a8c0@Kuhse.local>
MIME-Version: 1.0
Content-Type: text/plain; charset="us-ascii"
Content-Transfer-Encoding: 7bit
X-Mailer: Microsoft Office Outlook 11
X-MimeOLE: Produced By Microsoft MimeOLE V6. 00.2900.3198
Thread-Index: Acgmphvupn0/7fZHSuCUsWIxF+BZxg==
Test

.
  250 LAA16126 Message accepted for delivery
QUIT
  221 mail.kuhse.de closing connection
  [TCP Verbindungsaufbau]
```

Bild 39: E-Mail-Versand über SMTP

Grundsätzlich ist für den Mailversand über SMTP keine Authentifizierung vorgesehen. Dieser Sicherheitsmangel wurde durch eine Service Erweiterung (RFC 2554) behoben.

6.1.4 Point-to-Point Protocol (PPP) [64, 65, 66]

Das Point-to-Point Protocol (PPP) basiert auf dem High-Level Data Link Control (HDLC) Protokoll. Beide Protokolle sind standardisierte Sicherungsprotokolle für Punkt-zu-Punkt Verbindungen. PPP wird bei Einwahlverbindungen per Modem verwendet und dient dort als Sicherungsprotokoll für die Übertragungsstrecke zwischen Modem und Einwahlpunkt, dem Internet Service Provider (ISP). Innerhalb eines sog. PPP Rahmens (Telegramm) werden die Netzwerkprotokolle (z.B. IP, UDP, TCP) zum ISP übertragen, dort aus dem Rahmen genommen und über das Internet gesendet. Darüber hinaus werden mittels PPP Konfigurationsprotokolle (z.B. LCP, IPCP) zum ISP übertragen.
Spezifiziert ist PPP in RFC 1661.

Ein PPP Rahmen (Telegramm) ist wie folgt aufgebaut:

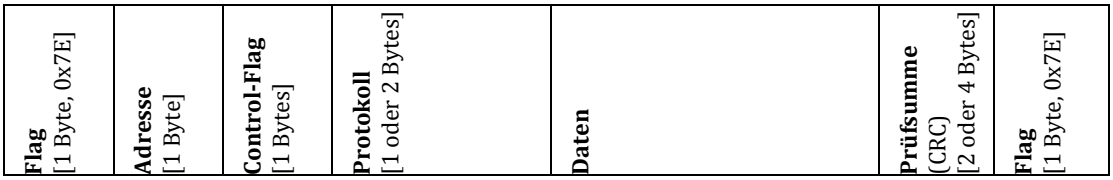


Tabelle 27: PPP Rahmenformat

Ein PPP Telegramm beginnt und endet mit einem definierten Flag, das den Hexadezimalwert 0x7E haben muss. Hieran erkennt der Empfänger das Telegramm.
Das Adress-Feld wird im PPP Datenverkehr nicht verwendet. Es stammt noch vom HDLC Protokoll. Trotzdem wird hier der Standardwert 0xFF erwartet, der angibt, dass alle Empfänger das Telegramm akzeptieren sollen (Broadcast).
Das Control-Flag stammt ebenso aus dem HDLC Protokoll und hat im PPP immer den Wert 0x03, der für einen unnummeriertes Telegramm steht.
Das Protokollfeld ist mit einer Größe von einem bis zwei Bytes spezifiziert, tatsächlich werden aber nur zwei Byte lange Protokoll-Definitionen verwendet. Spezifiziert wird hier das in den Nutzdaten übertragene Protokoll. Folgende Definitionen werden im KNG verwendet:

Wert	Protokoll
0x0021	Internet Protocol (IP)
0x8021	Internet Protocol Configuration Protocol (IPCP)
0xC021	Link Control Protocol (LCP)
0xC023	Password Authentication Protocol (PAP)

Tabelle 28: PPP Protokoll-Definitionen

Die Gesamtheit der möglichen Definitionen ist in RFC 1340 (Assigned Numbers) festgehalten.

Nach der Protokoll-Definition werden die Daten im angegebenen Protokollformat übermittelt, gefolgt von einer Prüfsumme. Die CRC Prüfsumme wird über das gesamte Telegramm gebildet, ausgenommen der PPP Flags am Anfang und Ende. Es wird für PPP eine zwei Byte lange Prüfsumme verwendet.

Da für PPP Telegramme der Hexadezimalwert 0x7E als Telegramm-Erkennungsflag verwendet wird, darf dieser Wert innerhalb des Telegramms nicht vorkommen. Aus diesem Grund muss das Tele-

gramm entsprechend bearbeitet werden. Jedes Byte mit dem Wert 0x7E wird durch das sog. Escape-Zeichen 0x7D gefolgt von 0x7E XOR 0x20 = 0X5E ersetzt. Da nunmehr der Wert 0x7D auch nicht mehr vorkommen darf, wird mit diesem Wert ebenso verfahren, d.h. jedes Vorkommen wird durch 0x7D und 0x7D XOR 0x20 = 0x5D ersetzt.
Nachteil dieses Verfahrens ist, dass das Telegramm sich theoretisch bis auf ca. die doppelte Ursprungslänge vergrößern kann.

Eine PPP Verbindung lässt sich mit folgendem Schema darstellen:

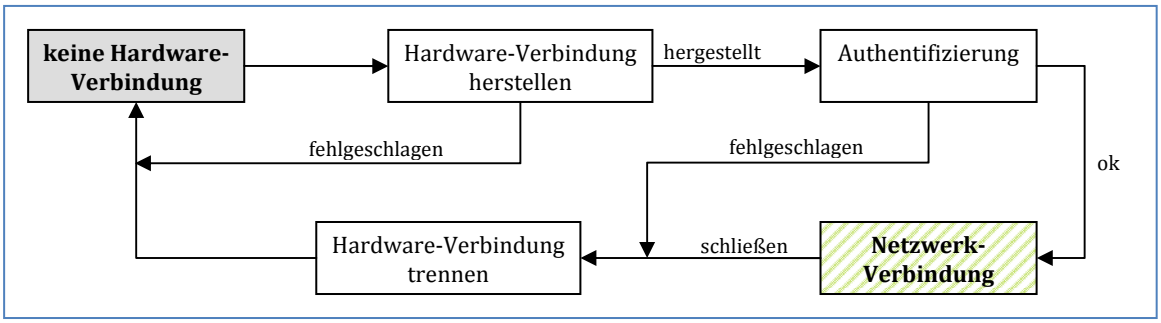


Bild 40: PPP Verbindungs-Schema

6.1.4.1 Network Control Protocol (NCP) [64, 66]

Das Network Control Protocol (NCP) wird verwendet, wenn über das PPP Telegramm eine Konfiguration übermittelt werden soll.

Ein NCP Telegramm ist wie folgt aufgebaut:

Bits Byte Offset	0-7	8-15	16-23	24-31
0	Code	Identifier	Länge	
4	Daten (optional)			

Tabelle 29: NCP Telegrammaufbau

Der Code gibt dabei Auskunft über die Art des Telegramms. Folgende Codes sind ebenfalls in RFC 1661 spezifiziert:

Code	Telegrammart
1	Konfigurations-Anfrage
2	Konfigurations-Bestätigung
3	Konfigurations-Ablehnung
4	Konfigurations-Zurückweisung
5	Abbruch-Anfrage
6	Abbruch-Bestätigung
7	Code-Zurückweisung
8	Protokoll-Zurückweisung
9	Echo-Anfrage
10	Echo-Antwort
11	Abschalt-Anfrage

Tabelle 30: NCP Codes

Der Identifier dient der Kennzeichnung und Identifizierung der Telegramme. Anfrage und Antwort müssen denselben Identifier haben.
Im Längen-Feld wird die Länge des kompletten NCP Telegramms angegeben.
In den Nutzdaten des NCP Telegramms können Konfigurationsblöcke übertragen werden, die wie folgt aufgebaut sind:

Bits Byte Offset	0-7	8-15
0	Typ	Länge
2	Daten	

Tabelle 31: NCP Konfigurationsblock-Aufbau

Je nach Konfigurationsblock-Typ werden im Datenblock bestimmte Parameter übertragen. Das Längenfeld gibt die Gesamtlänge des Konfigurationsblocks an.

6.1.4.2 Link Control Protocol (LCP) [64, 66]

Das Link Control Protocol (LCP) ist eine Form des NCP, mit dem Parameter der Hardware-Verbindung ausgehandelt werden. Diese Parameter sind z.B. das Authentisierungs-Protokoll oder Escape-Zeichen.
Folgende Typen für die LCP Konfigurationsblöcke sind in der RFC 1661 spezifiziert:

Typ	Bedeutung
1	Maximum-Receive-Unit (MRU) max. Telegrammlänge
2	Escape-Zeichen Definition
3	Authentifizierungs-Protokoll
4	Qualitäts-Protokoll
5	Magic-Number zur Verbindungssicherung
7	Protokollfeld Kompression
8	Adress- und Control-Flag-Kompression

Tabelle 32: LCP Konfigurationsblock-Typen

Nach dem Verbindungsaufbau mit einem ISP werden zunächst LCP Telegramme ausgetauscht.

6.1.4.3 Password Authentication Protocol (PAP) [64, 66, 67]

Auch das Password Authentication Protocol (PAP), das zur Authentifizierung dient, basiert ebenfalls auf dem NCP. Allerdings werden für das PAP Telegramm die NCP Codes anders interpretiert. (siehe Tabellen 30 und 33)

Code	Telegrammart
1	Authentifizierungs-Anfrage
2	Authentifizierungs-Bestätigung
3	Authentifizierungs-Ablehnung

Tabelle 33: PAP Codes

Im Datenfeld des PAP Telegramms stehen auch keine Konfigurationsblöcke sondern direkt Benutzername und Passwort. Jeweils vorangestellt ist ein Byte, das die entsprechende Länge angibt.

Die PAP Authentifizierung wird nach den LCP Verhandlungen durchgeführt.

6.1.4.4 Internet Protocol Control Protocol (IPCP) [64, 66]

Das Internet Protocol Control Protocol (IPCP) ist ebenfalls eine Form des NCP, mit dem eine IP Konfiguration vom ISP an den Client gesendet wird. Diese wird benötigt, um korrekte Internet Datenpakete im IP Format über die PPP Verbindung zu übermitteln.
Hier sind für die Konfigurationsblöcke folgende Typen definiert:

Typ	Bedeutung
1	IP Adresse abgelehnt
2	IP Kompressions-Protokoll
3	IP Adresse (Angebot)
4	Mobile-IPv4
129	Primärer DNS Server
130	Primärer NBNS Server
131	Sekundärer DNS Server
132	Sekundärer NBNS Server

Tabelle 34: IPCP Konfigurationsblock-Typen

Nachdem alle Hardware-Parameter festgelegt wurden und die Authentifizierung erfolgreich abgeschlossen ist, werden die Parameter der Internetverbindung vom ISP an den Client über IPCP Telegramme übermittelt.

6.2 Boot-Programm [43]

Bevor ein Programm auf dem Prozessor lauffähig ist, muss eine Basis-Initialisierung vorgenommen werden. Hierzu dient das Boot-Programm.

Das Boot-Programm des KNG basiert auf dem Demo-Quellcode für den MCF5272 von Freescale. Es ist in Assembler geschrieben, um mögliche Komplikationen durch den Compiler zu vermeiden. Außerdem stehen für bestimmte Register-Schreibbefehle keine entsprechenden Hochsprachenfunktionen zur Verfügung.

Da der Demo-Quellcode für den MCF5272 statt für den MCF5270 geschrieben ist, musste dieser angepasst werden. Außerdem wurden die Dateien vectors.s, mcf5xxx.s und mcf5272_lo.s in der Datei boot.s zusammengefasst.

Die aus der Datei vectors.s übernommenen Interrupt-Vektoren stehen am Beginn des Boot-Programms. Es handelt sich hierbei um eine Liste mit den Adressen der Interrupt Service Routinen (ISR) aller möglichen Interrupt-Quellen. (siehe Quellcode-Ausschnitt 1) Diese werden vom Prozessor an einer über das Vector Base Register (VBR) definierten Stelle benötigt. Meist werden sie aber am Anfang des Speichers angelegt.

```
.long   _irq_x          /* irq_064 = i_source_00 */
.long   _irq_x          /* irq_065 = i_source_01 = EXT_1 */
.long   _irq_x          /* irq_066 = i_source_02 = EXT_2 */
.long   _irq_x          /* irq_067 = i_source_03 = EXT_3 */
.long   _irq_x          /* irq_068 = i_source_04 = EXT_4 */
.long   _irq_x          /* irq_069 = i_source_05 = EXT_5 */
.long   _irq_x          /* irq_070 = i_source_06 = EXT_6 */
.long   isr_phy         /* irq_071 = i_source_07 = EXT_7 */
.long   _irq_x          /* irq_072 = i_source_08 = Software Watchdog TimeOut */
.long   _irq_x          /* irq_073 = i_source_09 = DMA_0 */
.long   _irq_x          /* irq_074 = i_source_10 = DMA_1 */
.long   _irq_x          /* irq_075 = i_source_11 = DMA_2 */
.long   _irq_x          /* irq_076 = i_source_12 = DMA_3 */
.long   isr_serio0      /* irq_077 = i_source_13 = UART0 */
.long   isr_serio1      /* irq_078 = i_source_14 = UART1 */
.long   _irq_x          /* irq_079 = i_source_15 = UART2 */
.long   _irq_x          /* irq_080 = i_source_16 = not used */
.long   _irq_x          /* irq_081 = i_source_17 = I2C  */
.long   _irq_x          /* irq_082 = i_source_18 = QSPI */
.long   isr_timer0      /* irq_083 = i_source_19 = TMR0 */
.long   _irq_x          /* irq_084 = i_source_20 = TMR1 */
.long   _irq_x          /* irq_085 = i_source_21 = TMR2 */
.long   _irq_x          /* irq_086 = i_source_22 = TMR3 */
.long   isr_fec         /* irq_087 = i_source_23 = FEC X_INTF */
.long   isr_fec         /* irq_088 = i_source_24 = FEC X_INTB */
.long   isr_fec         /* irq_089 = i_source_25 = FEC UN */
.long   isr_fec         /* irq_090 = i_source_26 = FEC RL */
.long   isr_fec         /* irq_091 = i_source_27 = FEC R_INTF */
.long   isr_fec         /* irq_092 = i_source_28 = FEC R_INTB */
.long   isr_fec         /* irq_093 = i_source_29 = FEC MII */
.long   isr_fec         /* irq_094 = i_source_30 = FEC LC */
.long   isr_fec         /* irq_095 = i_source_31 = FEC HBERR */
```

Quellcode 1: Interrupt-Vektoren (Ausschnitt) aus dem Boot-Programm

Das eigentliche Boot-Programm stammt aus der Datei mcf5272_lo.s und wurde in die Funktion _start übernommen. Um einen sicheren Systemstart zu gewährleisten wurden am Beginn der Start-Funktion drei NOP (no operation) Befehle eingefügt. Nach drei Taktzyklen ist das Gesamtsystem (Prozessor und Peripherie) als stabil anzusehen. Daraufhin werden zunächst alle Interrupt-Quellen deaktiviert, da vor der abgeschlossenen Initialisierung keine Interrupts auftreten sollen.

Nun wird das Vector Base Register gesetzt und das interne SRAM initialisiert, um dort den Stack einzurichten. Anschließend wird das C-Hauptprogramm (start_main) per Jump-Befehl aufgerufen. (siehe Quellcode-Ausschnitt 2)

```
_start:
  /* Ensure save start */
  nop
  nop
  nop

  /* Disable all interrupt-sources  */
  move.w  #0x2700, %sr

  /* Vector Base Register setup */
  move.l  #_vectors, %d0
  move.c  %d0, %VBR

  /* SRAM setup */
  move.l  #__SRAM, %d0
  or.l    #1, %d0
  move.c  %d0, %RAMBAR1

  /* Point Stack Pointer into SRAM */
  move.l  #(__SRAM), %d0
  add.l   #(__SRAM_SIZE), %d0
  add.l   #-4, %d0
  move.l  %d0, %sp

  /* Jump to main routine */
  jmp     start_main
```

Quellcode 2: Start-Funktion aus dem Boot-Programm

Außer der Start-Funktion sind in der Datei boot.s noch zwei weitere Funktionen untergebracht, die ursprünglich aus der Datei mcf5xxx.s stammen.
Eine dieser Funktion ist eine Interrupt Service Routine für unerwartete Interrupts (_irq_x). Diese Funktion dient dem Debugging der Firmware. Für alle Interrupt Vektoren, die nicht genutzt werden und damit nicht aufgerufen werden sollten, ist diese Funktion als Ziel eingetragen. Sollte einer der entsprechenden Interrupts doch auftreten, werden alle Registerinhalte auf dem Stack gesichert und der Exception Stack Frame des Prozessors ausgelesen. Dieses Register enthält Informationen zur Interrupt-Quelle. Diese Informationen werden zur Weiterverarbeitung an eine C-Funktion (isr_x) weitergegeben. Anschließend werden alle Registerinhalte wieder hergestellt.
Am Ende der Funktion wird ein Software-Reset durchgeführt. Damit wird im Normalbetrieb gewährleistet, dass das Programm nach einem unerwarteten Zustand sicher neu gestartet wird und nicht undefiniert weiter läuft. (siehe Quellcode-Ausschnitt 3)

```
/*****************************************************************
  _irq_x    (ISR!)
  This routine handles interrupt requests without specified handler.
  It calls: void isr_x(uint16 source)
 *****************************************************************/

_irq_x:
  link        %a6,#-60                /* create own stack area  */
  movem.l     %d0-%d7/%a0-%a6,(%sp)   /* save registers D0-D7, A0-A6 (A7 = SP!)  */

  move.l      4(%a6),%d0              /* current Exception Stack Frame  */

  move.l      %d0,-(%sp)              /* argument  */
  jsr         isr_x                   /* call isr_x  */

  lea         4(%sp),%sp              /* restore SP  */
  movem.l     (%sp),%d0-%d7/%a0-%a6   /* restore saved registers  */
  unlk        %a6                     /* free own stack area (restore A6)  */

  jmp         _start                  /* reset system */

  rte
```

Quellcode 3: Interrupt Service Routine für nicht genutzte Interrupts aus dem Boot-Programm

Die letzte Funktion des Boot-Programms dient der Manipulation des aktuellen Interrupt-Levels. Dieser wird vom Prozessor beim Auftreten eines Interrupts automatisch auf den Level dieses Interrupts gesetzt und nach Bearbeitung der Interrupt Service Routine wieder zurückgesetzt. So können hoch priorisierte Interrupts nicht durch niedriger priorisierte Interrupts unterbrochen werden.

Für die Bearbeitung von zeitkritischem Code kann es allerdings notwendig sein, alle Interrupt-Quellen abzuschalten bzw. nur Interrupts ab einem bestimmten Level zuzulassen. Hierzu kann die Funktion _asm_set_ipl aufgerufen werden. Diese Funktion erwartet als Übergabeparameter den zu setzenden Interrupt-Level und gibt als Rückgabewert den vorherigen zurück. (siehe Quellcode-Ausschnitt 4)

```
/*********************************************************************
  int asm_set_ipl(int)
  This routines changes the interrupt priority level (IPL) to the
  value passed into the routine. It also returns the old IPL value.
  Calling convention from C:
    old_ipl = asm_set_ipl(new_ipl);
  For the Diab Data C compiler, it passes return value thru D0.
  Note that only the least significant three bits of the passed
  value are used.
 *********************************************************************/

asm_set_ipl:
_asm_set_ipl:
  link    %A6,#-8                 /* create own stack area  */
  movem.l %D6-%D7, (%SP)          /* save registers D6 - D7 */

  move.w  %SR, %D7                /* current SR (Status Register) */

  move.l  %D7, %D0                /* prepare return value  */
  andi.l  #0x0700, %D0            /* mask out IPL (Interrupt Priority Level) */
  lsr.l   #8, %D0                 /* IPL   */

  move.l  8(%A6), %D6             /* get argument  */
  andi.l  #0x07, %D6              /* least significant three bits  */
  lsl.l   #8, %D6                 /* move over to make mask  */

  andi.l  #0x0000F8FF, %D7        /* zero out current IPL  */
  or.l    %D6, %D7                /* place-in new IPL  */
  move.w  %D7, %SR                /* write SR  */

  movem.l (%SP), %D6-%D7          /* restore registers D6 - D7  */
  lea     8(%SP), %SP             /* restore SP ???  */
  unlk    %A6                     /* free own stack area (restore A6)  */
  rts
```

Quellcode 4: Funktion zur Manipulation des Interrupt-Levels aus dem Boot-Programm

6.3 System-Initialisierung [68]

Nach der Basisinitialisierung aus dem Boot-Programm wird vom C-Hauptprogramm aus die weitere Initialisierung des Prozessors und der Schnittstellen aufgerufen. Die Routinen für die System-Initialisierung wurden mit Hilfe des kostenfreien Programms CFInit der Firma MicroAPL Ltd. erzeugt. Mit diesem Tool kann für alle Freescale Prozessoren eine Konfiguration erstellt und daraus der entsprechende C-Quellcode für die Initialisierung erstellt werden. (siehe Bild 41)

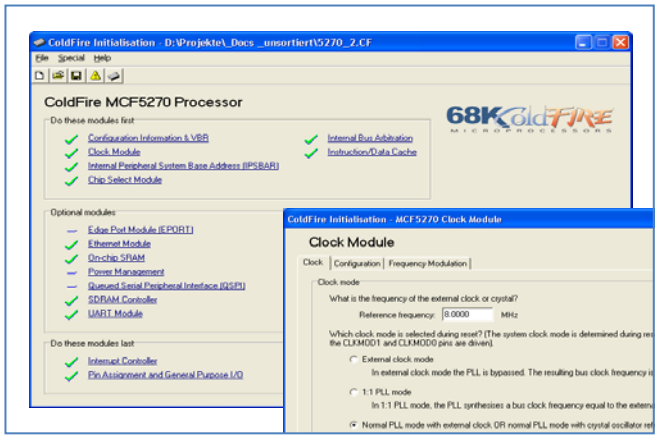


Bild 41: MicroAPL Tool CFInit

Alle Konfigurationsmöglichkeiten sind in Bereiche gegliedert, die im Quellcode durch separate Funktionen abgebildet werden.

Der erzeugte Quellcode ist in der Datei init.c ausgelagert. Vom Hauptprogramm aus wird die Funktion init_all() aufgerufen. (siehe Quellcode-Ausschnitt 5)

```
void init_all (void)
{
    /* Initialise base address of peripherals, VBR, etc */
    init_ipsbar ();
    init_basics ();
    init_clock_config ();

    /* Disable interrupts, watchdog timer, cache */
    disable_interrupts ();
    disable_watchdog_timer ();
    disable_cache ();

    /* Initialise individual modules */
    init_sram ();
    init_chip_selects ();
    init_bus_config ();
    init_cache ();
    init_eport ();
    init_power_management ();
//    init_uarts ();
//    init_ethernet ();
//    init_dma_timers ();
    init_interrupt_timers ();
    init_watchdog_timers ();
    init_pin_assignments ();

    /* Initialise SDRAM controller (must be done after pin assignments!) */
    init_sdram_controller ();

    /* Initialise interrupt controller */
    init_interrupt_controller ();

}
```

Quellcode 5: Initialisierungs-Routine (init_all)

Die Inhalte der Funktionen init_uarts(), init_ethernet() und init_dma_timers() wurden in separate Dateien ausgelagert und sind deshalb auskommentiert. Diese Funktionen wurden den speziellen Bedürfnissen angepasst und werden direkt vom Hauptprogramm aus aufgerufen.

6.4 Timer [10]

6.4.1 Funktionalität

Der MCF5270 verfügt über vier integrierte Hardware-Timer (Timer 0 bis Timer 3), von denen nur Timer 0 verwendet wird. Dieser Timer ist so konfiguriert, dass alle 10 ms ein Interrupt mit Priorität 6 ausgelöst wird. In der Interrupt Service Routine werden eine Software-Uhr und mehrere Software-Timer (einfache Abwärts-Zähler) entsprechend weitergezählt. Die hohe Interrupt-Priorität gewährleistet, dass Uhr und Timer in jedem Fall korrekt arbeiten.

Alle Timer-Funktionen sind in der Datei timers.c ausgelagert. Die Funktions-Prototypen stehen in der Datei timers.h.

6.4.2 Initialisierung

Die Timer-Initialisierung wurde aus der globalen Initialisierung (init.c) ebenfalls in die Datei timers.c ausgegliedert. Die Initialisierungsroutine timer_init() beinhaltet neben dem vom CFInit Tool erzeugten Quellcode die Initialisierung der Software-Timer und der Software-Uhr.
Die Konfiguration des Timers für die Auflösung von 10ms berechnet sich wie folgt.

$$T_{Timer\_out} = T_{Timer} \cdot [Referenzwert] = \frac{1}{f_{CLK}/2} \cdot [TeilerA] \cdot [TeilerB] \cdot [Referenzwert]$$

Der Eingangstakt der Timer-Einheit ist der interne Bustakt des Prozessors. Dieser beträgt immer die Hälfte vom Systemtakt. Da der Systemtakt des KNG mit 96 MHz festgelegt ist, beträgt der interne Bustakt 48 MHz. Dies entspricht einer Periodendauer von 20,8 ns.
Teiler A ist ein Eingangsteiler, der entweder 1 oder 16 sein kann. Da der Timer nur mit einer Auflösung von 10 ms laufen soll, wurde Teiler A mit 16 gewählt.
Teiler B ist von 1 bis 256 konfigurierbar. Um für die Timer-Periode auf einen rechnerisch guten Wert zu kommen, muss Teiler B bei einem Eingangstakt von 3 MHz (48 MHz / 16) ein Vielfaches von drei betragen. Es wurde hier der Wert 24 gewählt. Damit läuft der Timer mit einem Takt von 125 kHz. Die Periodendauer beträgt entsprechend 8 µs.
Der Referenzwert, nach dessen Erreichen der Interrupt ausgelöst wird, berechnet sich wie folgt:

$$[Referenzwert] = \frac{T_{Timer\_out}}{T_{Timer}} = \frac{10 \cdot 10^{-3} s}{8 \cdot 10^{-6} s} = 1250$$

Die Konfiguration von Timer 0 im Quellcode:

```
// Timer 0 is clocked at 125.00 KHz, expires after 10.00 ms, Timer resets on expiry & generates interrupt

MCF_TIMER_DTCN(0)   = 0;
MCF_TIMER_DTRR(0)   = 1249;                  // (1250-1)
MCF_TIMER_DTMR(0)   = MCF_TIMER_DTMR_PS(23)  // (24-1)
                    | MCF_TIMER_DTMR_ORRI
                    | MCF_TIMER_DTMR_FRR
                    | MCF_TIMER_DTMR_CLK_DIV16
                    | MCF_TIMER_DTMR_RST;
MCF_TIMER_DTXMR(0)  = 0;
```

Quellcode 6: Initialisierung von Timer 0

6.4.3 Software-Uhr

Da im KNG in der aktuellen Hardware-Version noch keine Echtzeituhr (RTC – real-time clock) integriert ist, wurde – zunächst für Debugging-Zwecke – eine Software-Uhr mit Hilfe des Timer 0 Interrupt realisiert.

Für die Software-Uhr wurde folgende Struktur definiert:

```
typedef struct PKT
{
  uint8   tt;
  uint8   mm;
  uint16  yy;

  uint8   h;
  uint8   m;
  uint8   s;

  uint16  ms;
  uint16  us;

} SW_RTC;
```

Quellcode 7: Struktur SW_RTC für Software-Uhr

Die Uhr, eine Variable vom Typ SW_RTC, wird beim Systemstart auf den 01.01.2007 00:00:00 Uhr gesetzt. Mit jedem Interrupt von Timer 0 wird in der Interrupt Service Routine (ISR) die Zeit um 10 ms weitergezählt und damit ggf. auch die Sekunden, Minuten etc. (siehe Quellcode-Ausschnitt 8)

```
    // Uhr (System-Timer)
    systimer.ms += 10;
    if (systimer.ms >= 1000)
    { // Sekunde
      systimer.ms = 0;
      if ((++systimer.s) >= 60)
      { // Minute
        systimer.s = 0;
        if ((++systimer.m) >= 60)
        { // Stunde
          systimer.m = 0;
          if ((++systimer.h) >= 24)
          { // Tage im Monat
            uint8 dpm = DAYSPERMONTH[systimer.mm];
            // Schaltjahr?
            if ((systimer.mm == 2) &&
               ((systimer.yy % 4 == 0 && systimer.yy % 100 != 0)  || systimer.yy % 400 == 0)) dpm++;
            // Tag
            systimer.h = 0;
            if ((++systimer.tt) > dpm)
            { // Monat
              systimer.tt = 1;
              if ((++systimer.mm) > 12)
              { // Jahr
                systimer.mm = 1;
                systimer.yy++;
              }
            }
          }
        }
      }
    }
```

Quellcode 8: Berechnung der Software-Uhr in der Timer 0 ISR

Über die Funktion systimer_set() kann die Software-Uhr gestellt werden. Übergabeparameter sind Tag, Monat, Jahr, Stunde, Minute und Sekunde. Millisekunde sowie Mikrosekunde werden auf Null gesetzt.

Um die Uhrzeit zu lesen gibt es die Funktion systimer_get(), in der eine Kopie der aktuellen Uhr angelegt und diese zurückgegeben wird. In dieser Funktion wird auch die Mikrosekunde berechnet. Hierzu wird der aktuelle Zählwert des Timers verwendet. Damit wird für die Software-Uhr eine Auflösung von 8 µs erreicht. (siehe Quellcode-Ausschnitt 9)

```
void
systimer_set (uint8 tt, uint8 mm, uint16 yy, uint8 h, uint8 m, uint8 s)
{
  systimer.tt = tt;
  systimer.mm = mm;
  systimer.yy = yy;

  systimer.h  = h;
  systimer.m  = m;
  systimer.s  = s;

  systimer.ms = 0;
  systimer.us = 0;
}

SW_RTC
systimer_get (void)
{
  int ipl;
  SW_RTC t;

  // Disable all(!) interrupts
  ipl = sys_set_ipl(7);

  systimer.us = (MCF_TIMER_DTCN(0) * 8);
  t = systimer;

  // Restore previous interrupt priority level (IPL)
  sys_enable_interrupts(ipl);

  return t;
}
```

Quellcode 9: Software-Uhr Funktionen (Setzen, Lesen)

Um Zähler-Überlaufprobleme zu vermeiden, werden alle Interrupt-Quellen für die Dauer der Ausführung der Funktion systimer_get() deaktiviert.

6.4.4 Software-Timer

Die Software-Timer sind einfache Abwärtszähler, die bei Erreichen vom Wert Null stehen bleiben. Für die Timer ist ein Feld von 32 Bit Variablen ohne Vorzeichen angelegt.
Die Auflösung der Software-Timer ist gleich dem Timer 0 und beträgt somit 10 ms. Es ist hier zu beachten, dass durch das asynchrone Setzen eines Timerwertes eine maximale Abweichung der gewünschten Zeit von ±10 ms auftreten kann.
Der Wert einer Timer-Variablen ist aufgrund der Auflösung als Vielfaches von 10 ms zu interpretieren. Daher sind Zeiten von 10 ms bis rund 497 Tage einstellbar.

Folgende Software-Timer sind per Definition implementiert:

Timer-ID	Name (per define)	Verwendung
1	TIMER_NETWORK	Timeout-Messung für Netzwerk-Adress-Anfragen (ARP)
2	TIMER_DHCP_1	Timeout-Messung 1 für DHCP (Netzwerk)
3	TIMER_DHCP_2	Timeout-Messung 2 für DHCP (Netzwerk)
4	TIMER_10MS_SIO0	Timeout-Messung für Serielle Schnittstelle #0
5	TIMER_10MS_SIO1	Timeout-Messung für Serielle Schnittstelle #1

Tabelle 35: Implementierte Software-Timer

Die Definition von Konstanten für die Timer-Nummer (ID) ermöglicht ein einfaches Handling in der Programmierung.

Zum Setzten und Auslesen der Timer sind die Funktionen timer_10ms_set() und timer_10ms_get() erstellt worden. (siehe Quellcode-Ausschnitt 10)

```
/*****************************************************************************
  > Timer
 *****************************************************************************/

void
timer_10ms_set (uint8 id, uint32 t)
{
  timer_10ms[id] = t;
}

uint32
timer_10ms_get (uint8 id)
{
  return timer_10ms[id];
}
```

Quellcode 10: Software-Timer Funktionen (Setzen, Lesen)

In der Interrupt Service Routine des Timers sorgt folgende Zeile für das Zählern der Timer:

```
    // Timer
    for (i=0;i<TIMER_10MS_MAX;i++) if (timer_10ms[i]) timer_10ms[i]--;
```

Quellcode 11: Zählen der Software-Timer in der Timer 0 ISR

6.5 Serielle Schnittstellen [10]

6.5.1 Funktionalität

Diese beiden seriellen Schnittstellen werden über je einen UART (Universal Asynchronous Receiver Transmitter) des MCF5270 betrieben.
Der Betrieb der EIA-485 Schnittstelle über den MPI12x Controller-Chip ist in der Hardware vorgesehen, obwohl diese Funktion für die erste Version der KNG Software nicht benötigt wird.

Über die EIA-232 (COM) Schnittstelle wird das KNG an die TeleControl Unterstation angeschlossen. Das KNG emuliert dabei ein Modem und leitet eingehende Datenpakete entsprechend über das Netzwerk weiter in das Internet. Datenempfang auf der Schnittstelle wird der Software über einen Interrupt gemeldet.
Die LWL Schnittstelle ist ebenfalls eine EIA-232 Schnittstelle und wird bisher verwendet, um Statusmeldungen für Debugging-Zwecke auszugeben.

Alle Funktionen für die seriellen Schnittstellen sind in die Datei serio.c ausgelagert. In der zugehörigen Header-Datei serio.h stehen die Funktions-Prototypen und Definitionen.

6.5.2 Initialisierung

Die Parameter der UARTs und damit der Schnittstellen wie z.B. Baudrate, Parität, Bitanzahl etc. werden über die Initialisierungsroutine serio_init_port() gesetzt. Es wurde eine spezielle Struktur definiert, in der alle Parameter gespeichert werden. (siehe Quellcode-Ausschnitt 12)
Außerdem sind in der Struktur ein Datenpuffer mit entsprechenden Zeigern und Statuszähler integriert. Diese werden beim Aufruf der Initialisierungsroutine zurückgesetzt.

```
/* Serial port object ------------------------------*/
typedef struct SERIAL_PORT_T
{
  /* Parameter ---------------------------------------*/
  uint8     id;           /* Port id                  */
  uint32    baud;         /* Baudrate                 */
  uint8     databits;     /* Data bits per frame      */
  uint8     parity;       /* Parity                   */
  uint8     stopbits;     /* Stop bits per frame      */
  uint8     flow;         /* Flow controlled by       */
  /* Flags (Status, ..) & statistics -----------------*/
  uint8     mode;         /* Port mode (see defines)  */
  vuint16   rxb;          /* Bytes received           */
  vuint16   txb;          /* Bytes sent               */
  /* Buffers -----------------------------------------*/
  uint8     rx_buf[SIO_RX_BUF_SIZE];      /* Buffer   */
  vuint16   rx_buf_id[2]; /* Buffer write / read index*/
  uint8     rx_fbuf[SIO_MAX_FRAMELENGTH]; /* Frame Buf*/
  uint16    rx_fbuf_id;
} SERIAL_PORT;
/*---------------------------------------------------*/
```

Quellcode 12: Struktur für serielle Schnittstellen

Entsprechend der Anzahl verfügbarer serieller Schnittstellen (COM, LWL) sind zwei globale Variablen vom Typ SERIAL_PORT mit folgender Aufteilung angelegt:

UART_0 = Port 0	LWL Schnittstelle
UART_1 = Port 1	COM Schnittstelle

Tabelle 36: Verwendung der UARTs

Die Schnittstellen-Nummer wird der Initialisierungsfunktion als Parameter übergeben, so dass diese universell verwendbar ist.
Zusätzlich wird der Initialisierungsfunktion ein Modus für die Schnittstelle übergeben, der bestimmt, wie empfangene Daten behandelt werden. Folgende Modi sind vorgesehen:

Modus	Beschreibung
SIO_MODE_DEFAULT	keine Verwendung
SIO_MODE_AT	Modem-Emulation (Empfang von AT Befehlen)
SIO_MODE_PPP	Modem-Emulation (Empfang von Datenpaketen im PPP Format)
SIO_MODE_K_TELE	Gerätekommunikation über KUHSE Telegramm

Tabelle 37: Modi für COM / LWL Schnittstelle

6.5.3 Datenempfang

Der per Interrupt gemeldete Datenempfang wird in der Interrupt Service Routine (ISR) verarbeitet.

Die Daten, die auf der COM Schnittstelle empfangen wurden, werden zunächst nur in einen Pufferspeicher umkopiert und die Zeiger entsprechend neu gesetzt. Zusätzlich wird ein Software-Timer gesetzt, um ein Datenverkehr Timeout zu überwachen. (siehe Quellcode-Ausschnitt 13)

```
void isr_serio0 (void)
{
  SAVEREG__();

  /* New data available? */
  if (MCF_UART_USR(UART_0) & MCF_UART_USR_RXRDY)
  {
    /* Write data into buffer and increment counter (pointer) */
    sio_port[UART_0].rx_buf[sio_port[UART_0].rx_buf_id[WRITE]] = MCF_UART_URB(UART_0);

    // Set Timeout-Timer //
    timer_10ms_set (TIMER_10MS_SIO0, 5);

    /* Increment pointer (counter) */
    if ((++sio_port[0].rx_buf_id[WRITE]) >= SIO_RX_BUF_SIZE)
      sio_port[0].rx_buf_id[WRITE] = 0;

    if (sio_port[0].rx_buf_id[WRITE] == sio_port[0].rx_buf_id[READ])
    {
      #ifdef DEBUG_PRINT
        serio_print(DEBUG_SIO, "\r\nRX-Buffer overflow!");
      #endif
      sio_port[0].rx_fbuf_id = 0;
    }

    /* Statistics */
    sio_port[0].rxb++;
  }

  RESTREG__();
}
```

Quellcode 13: ISR der COM Schnittstelle

Ist der Puffer voll und weitere Daten werden empfangen, so wird der Puffer gelöscht und damit das aktuelle Telegramm verworfen.

Die weitere Verarbeitung der Daten findet aus Timing-Gründen nicht in der ISR statt, sondern in der Funktion serio_main(), die zyklisch aus dem Hauptprogramm aufgerufen wird. Hier werden der Schreib- und Lesezeiger des Datenpuffers verglichen und je nach Modus der Schnittstelle die

entsprechend erwarteten Telegramme im Telegrammpuffer (rx_fbuf) zusammengesetzt. (siehe Quellcode-Ausschnitt 14)

```
for (pid=0; pid<SIO_PORTS; pid++)
{
  // Calc new data to avoid endless loop //
  if (sio_port[pid].rx_buf_id[READ] <= sio_port[pid].rx_buf_id[WRITE])
    i = sio_port[pid].rx_buf_id[WRITE] - sio_port[pid].rx_buf_id[READ];
  else
    i = SIO_RX_BUF_SIZE - sio_port[pid].rx_buf_id[READ] + sio_port[pid].rx_buf_id[WRITE];

  // No new data? -> break/continue !!! //
  if (!i) continue;

  for (;i;i--)
  {
    j = sio_port[pid].rx_fbuf_id;
    // Copy data //
    sio_port[pid].rx_fbuf[sio_port[pid].rx_fbuf_id++]
      = sio_port[pid].rx_buf[sio_port[pid].rx_buf_id[READ]];

    // Increment pointer (counter) //
    if ((++sio_port[pid].rx_buf_id[READ]) >= SIO_RX_BUF_SIZE)
      sio_port[pid].rx_buf_id[READ] = 0;

    switch (sio_port[pid].mode)
    {
```

Quellcode 14: Zusammensetzung von Telegrammen

Bei der Zusammensetzung der Telegramme wird zunächst die aktuelle Anzahl neuer Zeichen ermittelt und nur diese Anzahl auch kopiert. Damit wird vermieden, dass bei entsprechendem Datenaufkommen die Schleife nicht verlassen wird und weitere Programmfunktionen nicht mehr ausgeführt werden.
Die weitere Verarbeitung der Daten richtet sich nach dem aktuellen Schnittstellen-Modus.

6.5.4 AT-Modus (Modem-Modus)

Im AT-Modus erwartet das KNG AT-Befehle, die von der Unterstation zur Konfiguration an das Modem gesendet werden. Es wird dabei auf ein Carriage Return Zeichen (CR, Hexadezimal-Wert 0x0D) als Telegramm-Ende-Zeichen gewartet.
Alle empfangenen AT Telegramme werden verworfen, da das KNG nicht wie ein Modem konfiguriert werden kann und muss. Wird allerdings ein Wählbefehl (ATD) empfangen, so wird die Schnittstelle in den PPP-Modus geschaltet. (siehe Quellcode-Ausschnitt 15)

```
      default:  // Default & AT-Mode
       {
         // Frame complete? (<CR>) //
         if (sio_port[pid].rx_fbuf[j] == 0x0D)
         {
           if (sio_port[pid].mode == SIO_MODE_AT)
           {
             if ((sio_port[pid].rx_fbuf_id > 2) && (strncmp ((char *)sio_port[pid].rx_fbuf,"ATD",3) == 0))
             {
               #ifdef DEBUG_PRINT
                 serio_print(DEBUG_SIO, "\r\nDial!");
               #endif
               // Switch mode to ppp mode //
               sio_port[pid].mode = SIO_MODE_PPP;
               // Go online //
               ppp_go_on();
             }
             // else ignore AT frame //
           }
           // "Close" frame
           sio_port[pid].rx_fbuf_id = 0;
         }
       }
```

Quellcode 15: Auswertung von seriellen Telegrammen im AT-Modus

6.5.5 PPP-Modus

Im PPP-Modus werden vom KNG Telegramme im PPP (Point-to-Point Protocol) Format erwartet. Diese Telegramme werden zur Kommunikation zwischen Teilnehmer (Client) – in diesem Fall die Unterstation – und Internet Service Provider (ISP) verwendet.

PPP Telegramme beginnen und enden mit dem PPP Identifier (Hexadezimal-Wert 0x7E). Anhand dieses Identifiers werden die Telegramme ausgewertet. (siehe Quellcode-Ausschnitt 16)

```
        case (SIO_MODE_PPP):
        {
          uint8 err;
          err = 0;

          // Frame complete? //
          if ((j) && (sio_port[pid].rx_fbuf[j] == PPP_IDENTIFIER))
          {
            // Check identifier //
            if ((sio_port[pid].rx_fbuf[0] == PPP_IDENTIFIER))
            {
              ppp_fbuf fbuf;

              fbuf.length   = j - 1;
              fbuf.data     = &sio_port[pid].rx_fbuf[1];

              if (ppp_handler (&fbuf))
              {
                if (!DCD_STATUS)
                {
                  #ifdef DEBUG_PRINT
                    serio_print(DEBUG_SIO, "\r\nDCD cleared!");
                  #endif
                  // Switch back to AT mode
                  sio_port[pid].mode = SIO_MODE_AT;
                }
              }
              else err = 1;
            }
            else err = 1;

            // Error ?
            if (err)
            {
              #ifdef DEBUG_PRINT
                  serio_print(DEBUG_SIO, "\r\nPPP Frame (error)!");
                serio_send_data(DEBUG_SIO, (char *) sio_port[pid].rx_fbuf, 4);
              #endif
            }
            // "Close" frame
            sio_port[pid].rx_fbuf_id = 0;
          }
        }
```

Quellcode 16: Auswertung von seriellen Telegrammen im PPP-Modus

Alle PPP-Funktionen für die weitere Telegrammverarbeitung wurden in die Datei ppp.c ausgelagert. Auch hierfür existiert eine Header-Datei (ppp.h) mit den Funktionsprototypen und Definitionen.

Wird über die Funktion ppp_go_on() in den PPP-Modus gewechselt, wird in dieser Funktion das DCD Signal der Schnittstelle gesetzt und damit der Unterstation signalisiert, dass ein Träger (Carrier) verfügbar ist.
Damit ist für die Unterstation die Verbindung zum Internet Service Provider (ISP) hergestellt und nach einem Parametrierprozess können Daten über das Internet übermittelt werden.
(siehe Bild 42)

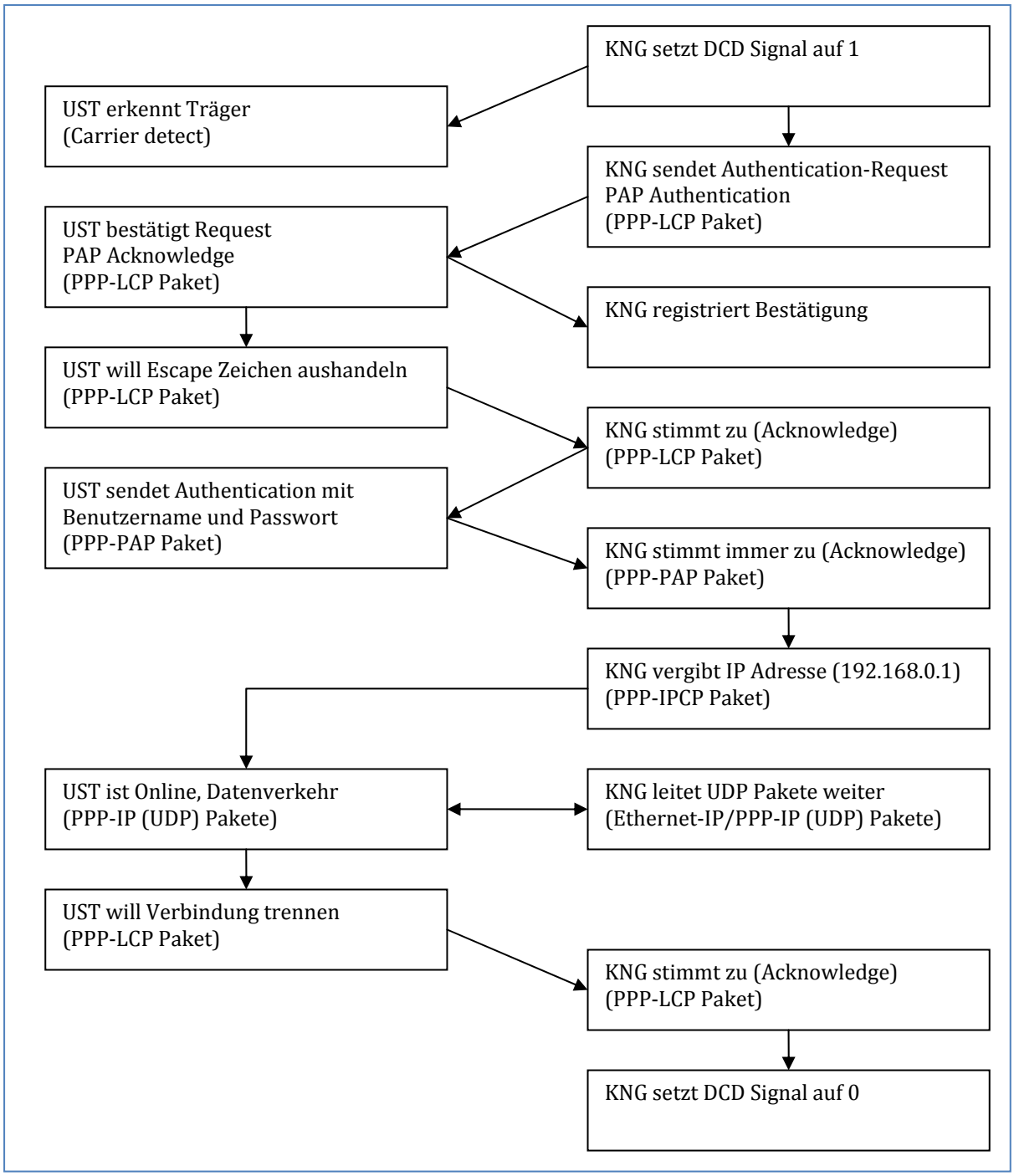


Bild 42: Ablauf der PPP-Verbindung

6.6 Netzwerk [43]

6.6.1 Funktionalität

Wird das KNG an ein Netzwerk angeschlossen, versucht es über das Dynamic Host Configuration Protocol (DHCP) eine gültige IP Adresse zu erhalten. Ist eine Verbindung zustande gekommen, kann das KNG Telegramme folgender Netzwerkprotokolle verarbeiten:

Ethernet Schicht	Protokolle
Transport	Datagram Protocol (UDP) Transmission Control Protocol (TCP)
Netzwerk	Internet Protocol Version 4 (IPv4) Internet Control Message Protocol (ICMP) Dynamic Host Configuration Protocol (DHCP)
Netzzugang	Ethernet Protocol Address Resolution Protocol (ARP)

Tabelle 38: Netzwerkprotokolle, die das KNG verarbeiten kann

Die grundsätzliche Verarbeitung der Protokolle wurde aus dem Freescale Demo-Code für den MCF5272 übernommen und entsprechend angepasst oder erweitert.
Die entsprechenden Funktionen sind wie folgt ausgegliedert:

Dateiname	Beschreibung
arp.c, arp.h	Verarbeitung des Address Resolution Protocol (ARP)
dhcp.c, dhcp.h	Verarbeitung des Dynamic Host Configuration Protocol (DHCP)
eth.c, eth.h	Verwaltung des Netwerk-Interfaces (Protokoll-Registrierung, Funktion eth_main() etc.)
fec.c, fec.h	Schnittstelle zum Fast Ethernet Controller (FEC) (Initialisierung, Interrupt Service Routine, etc.)
icmp.c, icmp.h	Verarbeitung des Internet Control Message Protocol (ICMP)
ip.c, ip.h	Verarbeitung des Internet Protocol Version 4 (IPv4)
ksz8041.h	Registerdefinitionen für den Ethernet PHY
nbuf.c, nbuf.h	Verwaltung der Netzwerkpuffer
nif.c, nif.h	Netzwerkinterface-Struktur und -Funktionen (Einbindung der Protokoll Handler)
phy.c, phy.h	Schnittstelle zum Ethernet PHY (Initialisierung etc.)
tcp.c, tcp.h	Verarbeitung des Transmission Control Protocol (TCP)
udp.c, udp.h	Verarbeitung des Datagram Protocol (UDP)

Tabelle 39: Funktionsaufteilung Ethernet

Für den Einsatz des KNG als Netzwerk Gateway für die TeleControl Unterstation werden UDP Datenpakete zwischen Unterstation und TeleControl Master vermittelt. Die Telegramme zwischen KNG und Unterstation werden eingebettet in PPP Telegrammen über die COM Schnittstelle übermittelt. Das KNG übernimmt hier die Funktion des Internet Service Providers (ISP) bei einer Modemeinwahl.

6.6.2 Netzwerkinterface-Struktur

Im übernommenen Demo-Code wird eine Netzwerkinterface-Struktur (NIF) verwendet, um darüber alle Funktionen und Parameter für die Netzwerk-Kommunikation zentral zu verwalten.
Diese Struktur ist wie folgt in der Datei nif.h definiert:

```
typedef struct
{
  uint16  protocol;
  void  (*handler)(void * /* nif */, NBUF * /* buffer */);
  void  *info;  /* pointer to protocol defined config info */
} SUP_PROTO;

typedef
uint8 HWA_ADDR_P[];

typedef struct
{
  vuint8    rxb;
  vuint8    rxf;
  vuint8    txf;
  vuint8    rxerr;
  vuint8    txerr;
} NIF_STAT;

typedef struct NIF_t
{
  char        name[16];   /* hardware name */
  ETH_ADDR    hwa;        /* hardware address (MAC) */
  int         hwa_size;
  ETH_ADDR    broadcast;  /* broadcast address */

  SUP_PROTO       protocol[MAX_SUP_PROTO];    /* Handler information for supported protocols */
  unsigned short  num_protocol;               /* Pointer to next free sup. protocol mem */

  int   (*reset)    (struct NIF_t *);
  void  (*start)    (struct NIF_t *);
  void  (*stop)     (struct NIF_t *);
  int   (*send)     (struct NIF_t *, HWA_ADDR_P, HWA_ADDR_P, uint16, NBUF *);
  void  (*receive)  (struct NIF_t *);

  NBUF* (*rx_alloc) (void);
  NBUF* (*tx_alloc) (void);
  void  (*rx_free)  (NBUF *);
  void  (*tx_free)  (NBUF *);

  void          *nic;         /* base address of NIC chipset (PHY base address) */
  int           vector;       /* vector used by device */

  vuint8        status;       // Status of Network Interface (NIF) //
  NIF_STAT      stat;         // Statistics //

} NIF;
```

Quellcode 17: Netzwerkinterface-Struktur (NIF)

Folgende Funktionen werden den Funktionszeigern der Struktur zugewiesen:

Funktionsname (Zeiger)	Funktion	Datei	Beschreibung
reset(…)	fec_reset(…)	fec.c	Register-Initialisierung des FEC
start(…)	fec_start(…)	fec.c	PHY-Initialisierung, Interrupt-Freigabe
stop(…)	fec_stop(…)	fec.c	Übertragungsstopp, Interrupt-Deaktivierung
send(…)	fec_send(…)	fec.c	Senden eines Ethernet-Telegramms
receive(…)	fec_receive(…)	fec.c	Verarbeitung von empfangenen Telegrammen
rx_alloc(…)	nbuf_rx_allocate(…)	nbuf.c	Puffer zum Empfang von Daten reservieren
tx_alloc(…)	nbuf_tx_allocate(…)	nbuf.c	Puffer zum Senden von Daten reservieren
rx_free(…)	nbuf_rx_free(…)	nbuf.c	Puffer zum Empfang von Daten freigeben
rx_free(…)	nbuf_rx_free(…)	nbuf.c	Puffer zum Senden von Daten freigeben

Tabelle 40: Funktionszuordnung Netzwerkinterface-Struktur

Der Vorteil der NIF Struktur ist, dass beim Aufruf von Unterfunktionen selten mehr als ein Zeiger auf die Struktur selber übergeben werden muss. Dies ist besonders in der Verarbeitung der Netzwerkprotokolle nützlich, da hier durch die geschachtelten Protokolle oft mehrere Funktionen (Handler) nacheinander aufgerufen werden.
In der Datei eth.c ist eine globale Variable vom Typ NIF angelegt. Hier finden über die Funktion eth_init() auch die Initialisierung und die Zuweisung der Handler statt.

6.6.3 Initialisierung

Für die Netzwerk-Initialisierung muss neben dem FEC des Prozessors auch der externe PHY Baustein und die Pufferstruktur initialisiert werden. Die Initialisierung läuft nach folgendem Schema:

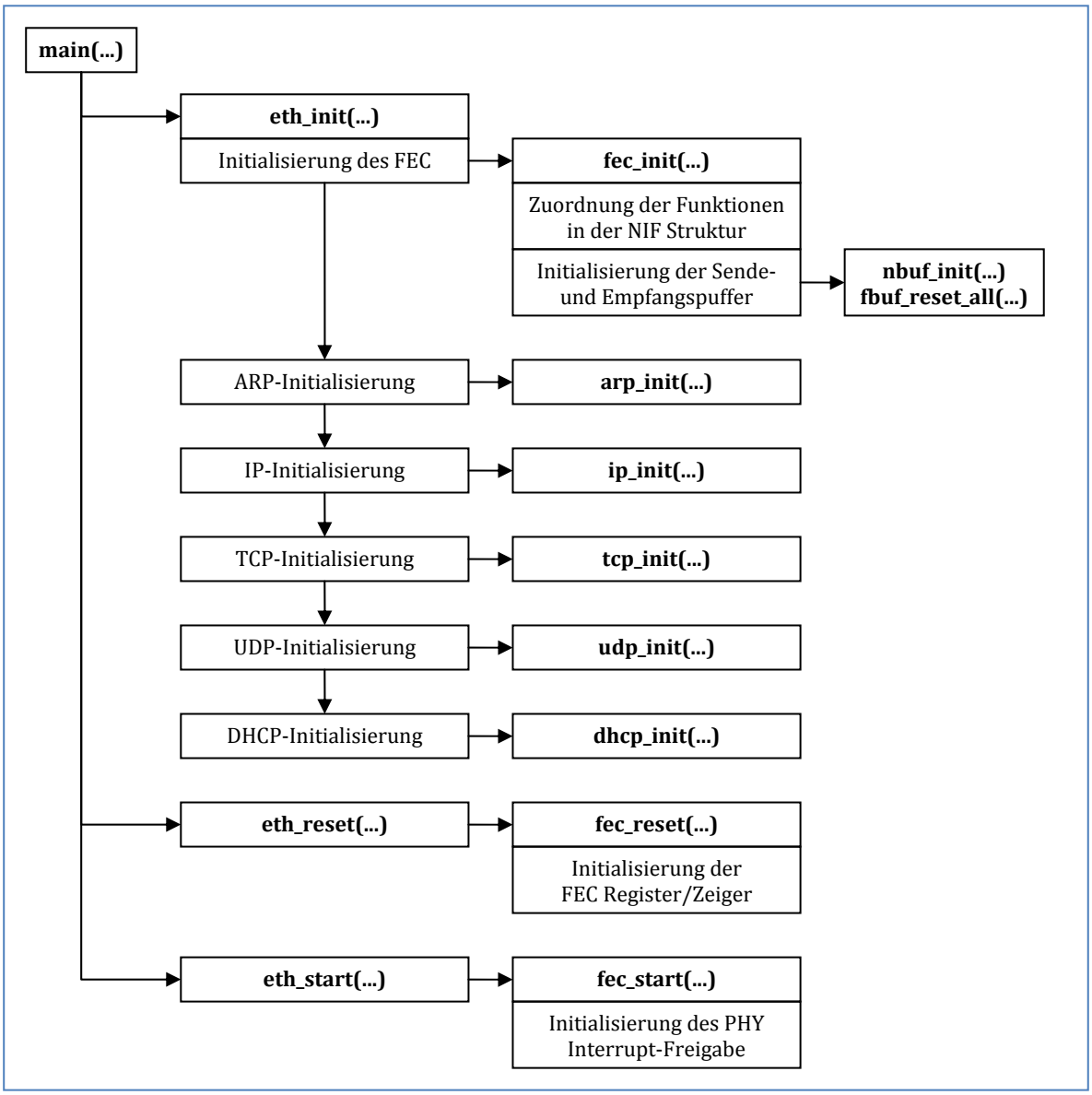


Bild 43: Initialisierungsablauf Netzwerk

In der Darstellung sind die tatsächlich aufgerufenen Funktionen angegeben und nicht die Zeigernamen der NIF Struktur. In der Firmware werden diese Zeiger allerdings verwendet.

6.6.3.1 PHY [12]

Im PHY muss die physikalischen Schnittstellenparameter wie Geschwindigkeit, Duplex Betrieb etc. konfiguriert werden. Außerdem sind auch hier Interrupts parametrierbar.

Der PHY im KNG ist so konfiguriert, dass alle Schnittstellenparameter per Auto-Negotiating vom PHY selber ermittelt werden.
Folgende Interrupts sind freigegeben:

- Link-Down
- Link-Up
- Link-Partner-Acknowledge
- Remote-Fault
- Receive-Error

6.6.3.2 Telegrammpuffer [10]

Für den FEC muss ein Speicherbereich als Puffer für ankommende und abgehende Telegramme reserviert werden. Der FEC greift darauf mittels Direct Memory Access (DMA) zu. Für ein besseres Handling wird der reservierte Puffer jeweils in mehrere kleinere Teilbereiche unterteilt. Die einzelnen Teilbereiche werden über sog. Puffer-Deskriptoren definiert und verwaltet. Diese Deskriptoren müssen separat angelegt werden und enthalten je ein Status- und Längenfeld sowie einen Zeiger auf den Datenspeicher. Die Statusfelder für Empfangs- und Sendepuffer enthalten dabei unterschiedliche Flags.
Alle Deskriptoren für eine Übertragungsrichtung müssen im Speicher hintereinander abgelegt sein. Die Adresse des jeweils ersten ist in einem bestimmten Register abgelegt. (siehe Bild 44) Über das Statusfeld wird dem Prozessor angezeigt, ob der letzte Deskriptor erreicht ist und wieder beim ersten begonnen werden soll. Der Puffer wird damit als sog. Ringspeicher genutzt.

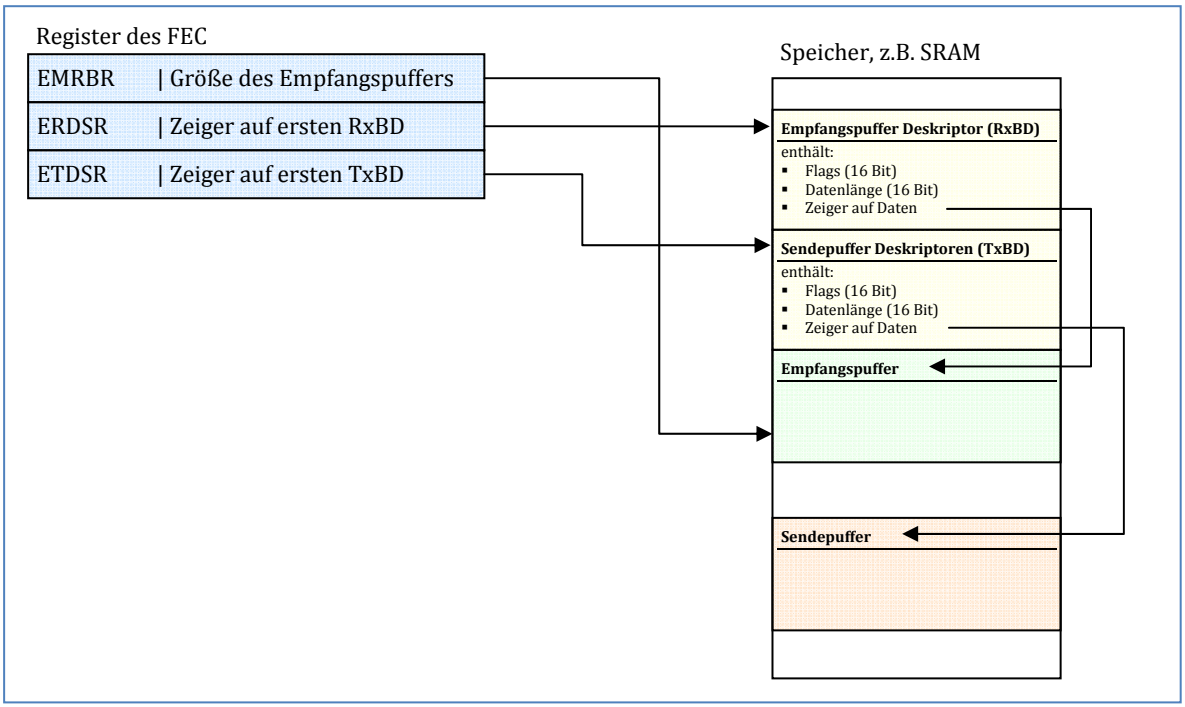


Bild 44: FEC Empfangs- und Sendepuffer

Der FEC kann maximal einen Empfangspuffer mit einer Größe von 2032 Bytes verwalten. Der Sendepuffer hingegen muss von der Software verwaltet werden und ist damit in der Größe nicht beschränkt. Für das KNG wurden ein Empfangspuffer mit einer Größe von 1728 Byte (3x576 Byte) und ein Sendepuffer mit einer Größe von ebenfalls 1728 Byte vorgesehen.
Die Teilgröße von 576 Byte wurde gewählt, damit auch TCP Telegramme mit bis zu 512 Bytes Nutzdaten in nur einem Puffer gespeichert werden können. Zur Berechnung wurden die Längen der einzelnen Header addiert:

$$l = l_{Data} + l_{TCP} + l_{IP} + l_{ETH} = 512 + 20 + 20 + 14 = 566$$

Da die Größe der Puffer ein Vielfaches von 16 betragen muss, ist die Größe auf 576 Bytes aufgerundet worden.

Um die Verarbeitung der Puffer-Deskriptoren in der Software zu erleichtern, wurde deren Struktur entsprechend angelegt. (siehe Quellcode-Ausschnitt 18)

```
/* Buffer Descriptor Format */
typedef struct PKT
{
  uint16 status;  /* control and status */
  int16  length;  /* transfer length */
  uint8  *data;   /* buffer address */
} NBUF;
```

Quellcode 18: Struktur der Netzwerkpuffer-Deskriptoren

Hier wurde der Demo-Code modifiziert, da damit nur Telegramme verarbeitet werden können, die komplett in einen Teilpuffer passen. Diese in der Größe auf das Maximum zu ändern hätte eine ineffektive Verarbeitung kleinerer Telegramme zur Folge. Ein generelles Umkopieren der Daten in einen anders aufgeteilten Speicherbereich ist bezüglich kurzer Telegramme auch nicht sinnvoll, da durch das Kopieren eine deutlich längere Zeit zur Verarbeitung benötigt wird.
Mit der folgenden Struktur wurde eine praktikable Lösung geschaffen:

```
/* Frame BUFfer */
typedef struct PKT
{
  uint16        status;                 /* Status flags */
  int16         length;                 /* transfer length */
  uint8         *data;                  /* pointer to data */
  uint8         fdata[FBUF_MAX_SIZE];   /* frame data  */
  uint8         fbuf_status;            /* FBUF status */
} FBUF;
```

Quellcode 19: Struktur für größere Netzwerkpuffer

Diese Struktur kann durch ihren Aufbau wie eine Deskriptoren-Struktur per Cast an weiterführende Funktionen (Protokoll-Handler) übergeben werden. Es muss nur darauf geachtet werden, dass der Zeiger auf die Daten (*data) immer auf den eigenen Datenpuffer (fdata) zeigt.
Zur Verwaltung der Datenpuffer ist ein zusätzliches Statusfeld integriert.

Im KNG sind vier dieser Puffer mit einer Puffergröße von je 1514 Bytes vorgesehen. Dies entspricht der maximalen Länge von Ethernet Telegrammen ohne Präambel und Checksumme, die vom FEC nicht im Puffer gespeichert werden.

6.6.4 Adressverwaltung

Zur Adressierung von Netzwerktelegrammen dient grundsätzlich die Hartware-Adresse. Diese sog. MAC Adresse (Media Access Control) muss für jede netzwerkfähige Hardware eindeutig sein.
Für die Kommunikation ab der Netzwerk-Schicht des OSI Modells, die praktisch immer verwendet wird, gibt es eine weitere Adressierung mittels IP Adressen.

6.6.4.1 Ethernet Protokollschicht

Um Datenpakete über das Netzwerk zu versenden, wird immer die Hardware-Adresse des Empfängers benötigt, auch wenn die Kommunikation auf der höheren Schicht über IP Telegramme läuft.
Um die Hardware-Adresse für eine bekannte IP Adresse zu ermitteln, werden ARP Telegramme (Address Resolution Protocol) verwendet.

In der KNG Firmware ist eine ARP Tabelle hinterlegt, damit eine einmal bekannte Adresse nicht immer wieder neu angefragt werden muss. Die Tabelle ist für 10 Einträge ausgelegt. Es werden das verwendete Protokoll (IP) und das Adresspaar (Ethernet und IP) eingetragen. Außerdem gibt es für jeden Eintrag eine Lebensdauer, die bestimmt, ob ein Eintrag permanent ist oder bei Bedarf gelöscht werden darf.

Folgende ARP Funktionen sind in der Datei arp.c implementiert:

Funktionsname	Funktionsbeschreibung
arp_init(...)	Initialisiert die ARP Tabelle.
arp_find_pair(...)	Sucht je nach übergebenen Parametern entweder IP- oder Hardware-Adresse aus der ARP Tabelle und gibt bei Erfolg einen Zeiger auf diese zurück.
arp_merge(...)	Fügt Adressinformationen in die ARP Tabelle ein.
arp_remove(...)	Löscht einen Eintrag aus der ARP Tabelle.
arp_request(...)	Sendet ein ARP Request Telegramm, um die Hardware-Adressinformationen für eine angegebene IP Adresse zu ermitteln.
arp_resolve_pa(...)	Sucht eine angegebene IP Adresse in der ARP Tabelle und gibt bei Erfolg einen Zeiger auf die zugehörige Hardwareadresse zurück.
arp_resolve(...)	Sucht eine angegebene IP Adresse in der ARP Tabelle und gibt bei Erfolg einen Zeiger auf die zugehörige Hardwareadresse zurück. Ist kein Eintrag in der Tabelle gefunden worden, wird arp_request(...) ausgeführt.
arp_handler(...)	Verarbeitet eingehende ARP Telegramme.

Tabelle 41: ARP Funktionen

Eingehende ARP Telegramme werden in der Funktion arp_handler() nach folgendem Schema verarbeitet:

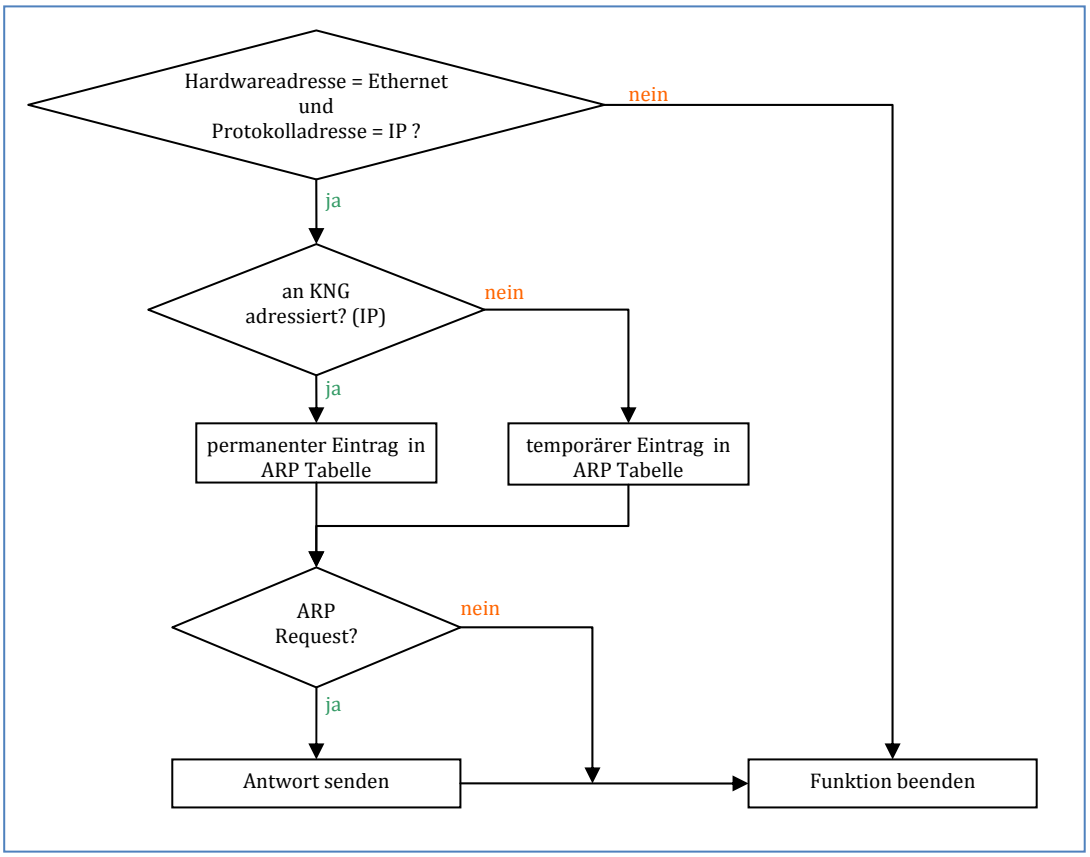


Bild 45: ARP Telegrammverarbeitung

6.6.4.2 IP [49]

Anders als die Hardware-Adressen müssen die IP Adressen lediglich in einem abgeschlossenen Netzwerk eindeutig sein. Dies kann ein Local Area Network (LAN) oder auch ein Wide Area Network (WAN) wie das Internet sein.
Da die manuelle Verwaltung der Adressen eines Netzwerks aufwändig ist und man außerdem ein Gerät wie das KNG einfach anschließen möchte, gibt es eine automatische, servergesteuerte Adressvergabe mittels Dynamic Host Configuration Protocol (DHCP). DHCP Telegramme werden über UDP Telegramme versendet. Sendeportnummer ist dabei die 68 und Empfangsportnummer die 67.

Die DHCP Funktionen sind in der Datei dhcp.c implementiert. Folgende Hauptfunktionen gibt es, die von extern aufgerufen werden können:

Funktionsname	**Funktionsbeschreibung**
dhcp_init(...)	Initialisiert DHCP. (UDP Port, Status-Flag)
dhcp_reset(...)	Löscht die aktuelle IP Adresse und setzt den Status zurück.
dhcp_main(...)	Prüft aktuellen DHCP Status und sendet ggf. DCHP Telegramme.
dhcp_handler(...)	Verarbeitet eingehende DHCP Telegramme.

Tabelle 42: DHCP Funktionen

Die Funktionen dhcp_main() und dhcp_handler() verarbeiten in Kombination die Aushandlung einer IP Adresse über DHCP. Generell lässt sich die Aushandlung wie folgt darstellen:

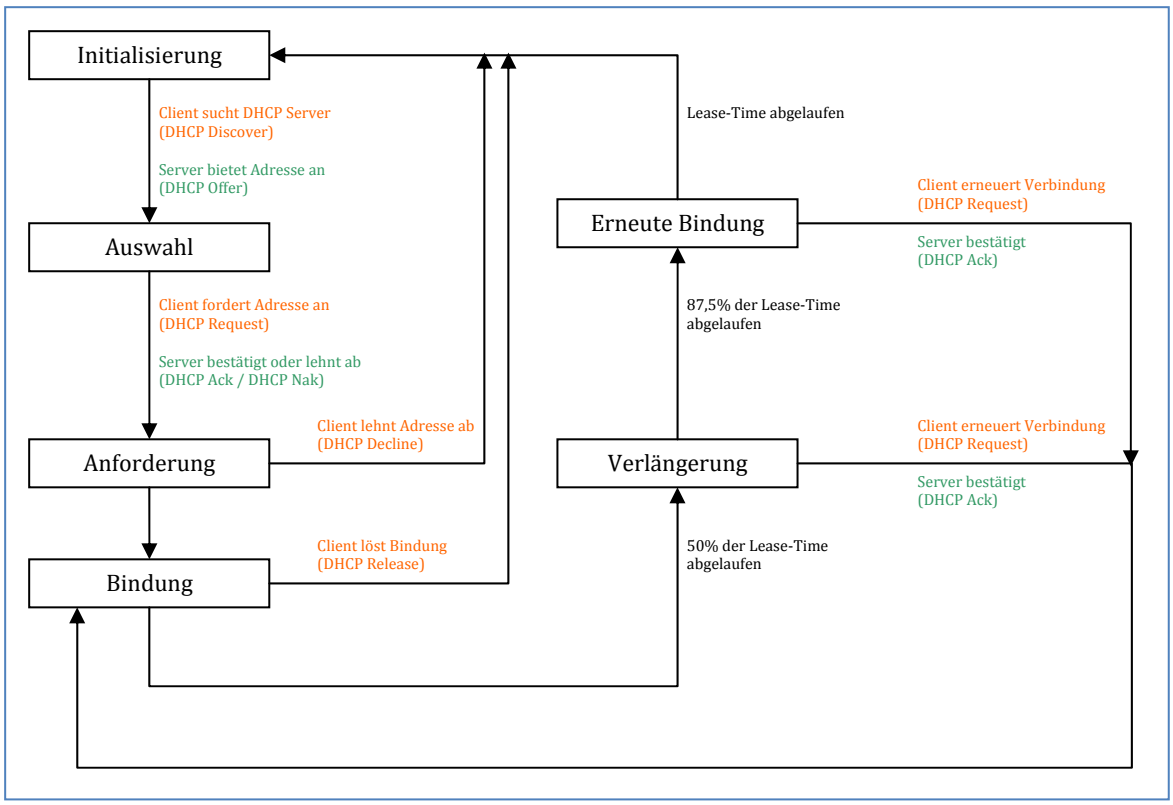


Bild 46: DHCP Verfahren

6.6.5 Verbindung

Soll eine Verbindung zum KNG über das Netzwerk aufgebaut werden, wird die IP Adresse benötigt.

6.6.5.1 Ping

Um zu prüfen, ob sich hinter einer IP Adresse ein Teilnehmer, wie z.B. das KNG, befindet, wird im Netzwerkbereich ein sog. Ping verwendet. Dahinter verbirgt sich das Internet Control Message Protocol (ICMP) Echo-Telegramm. Dieses ist auch im KNG implementiert.

Die Verarbeitung des ICMP Echo-Telegramms ist relativ simpel. Es muss lediglich der Header eines empfangenen Telegramms angepasst werden. Hier wird der Typ von Anfrage auf Antwort geändert, die Adressen getauscht und die Checksumme neu berechnet. Das Paket wird daraufhin an den Sender zurückgeschickt.

6.6.5.2 Verbindungsaufbau UDP / TCP [55]

Für Netzwerktelegramme ist es generell nicht notwendig eine Verbindung gezielt aufzubauen. Es sind sowohl für UDP als auch für TCP Telegramme bestimmte Ports für bestimmte Protokolle vorgesehen oder können definiert werden.
Bei einer TCP Verbindung sind allerdings noch Software-Sicherungsmechanismen vorgesehen, die einer Initialisierung bedürfen. Daher muss eine TCP Verbindung über einen bestimmten Telegramm-

ablauf aufgebaut und auch abgebaut werden. (siehe Bild 47) Nur über eine korrekt aufgebaute Verbindung können Daten ausgetauscht werden.

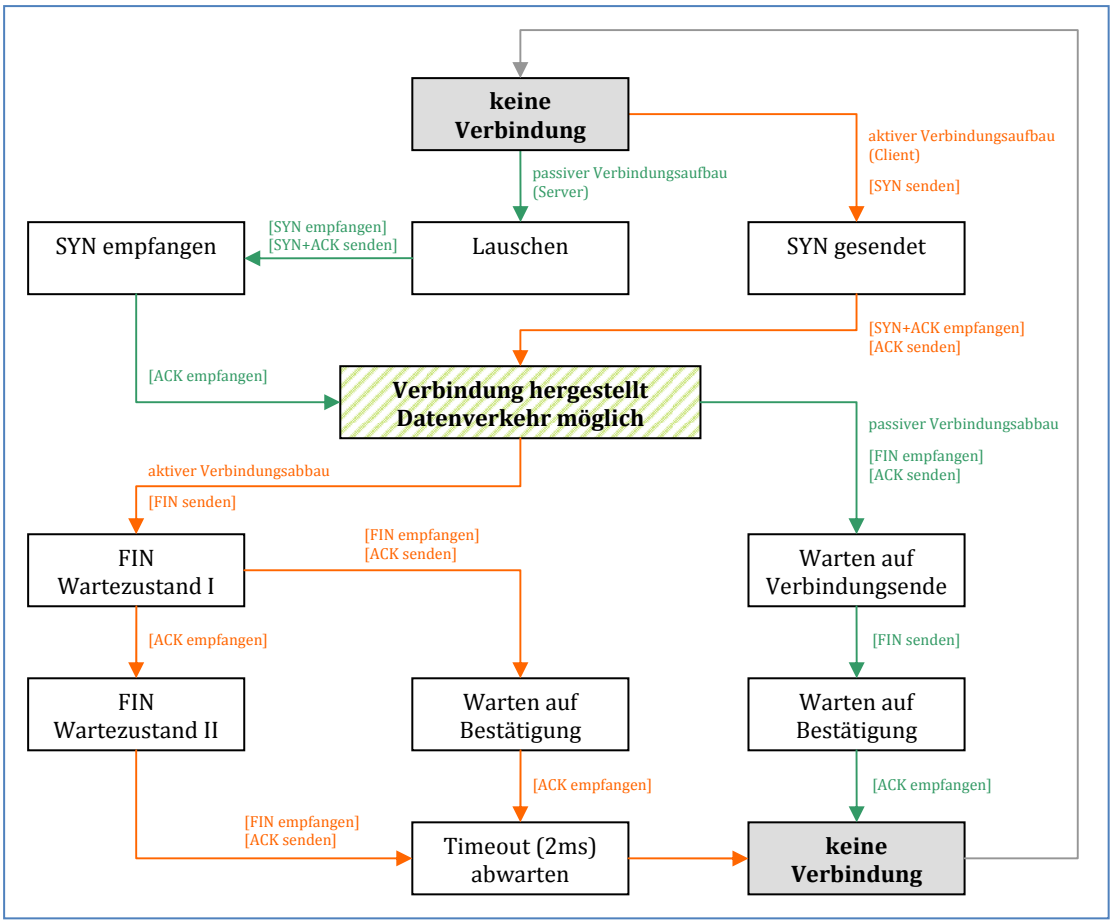


Bild 47: TCP Verbindungsdiagramm

Der aktive TCP Verbindungsaufbau erfolgt immer von einem Client, der sich mit einem Server verbinden möchte. Dieser muss entsprechend auf die Anfragen reagieren. Der Verbindungsabbau hingegen kann sowohl von einem Client als auch von einem Server aktiv begonnen werden. Der jeweilige Ablauf kann aus Bild 47 abgelesen werden. Die Schritte der aktiven Seite sind dabei immer orange, die der passiven Seite (Gegenseite) immer grün dargestellt.
Nach RFC gibt es weitere mögliche Pfade, die aber üblicherweise keine Verwendung finden und aus Gründen der Übersichtlichkeit im Diagramm weggelassen wurden.

6.6.6 Datenempfang

Datenempfang auf der Netzwerkschnittstelle wird per Interrupt angezeigt. In der entsprechenden Interrupt Service Routine (ISR) werden allerdings nur Software-Flags gesetzt. Die eigentliche Verarbeitung der Telegramme findet aus Timing-Gründen in einer separaten Funktion statt, die zyklisch aus dem Hauptprogramm aufgerufen wird.

In der Verarbeitungsfunktion wird ermittelt, ob das Telegramm komplett oder auf mehrere Teilpuffer aufgeteilt ist. Müssen mehrere Teile zusammengefügt werden, so wird einer der großen Telegrammpuffer (FBUF) verwendet. (siehe Bild 48)

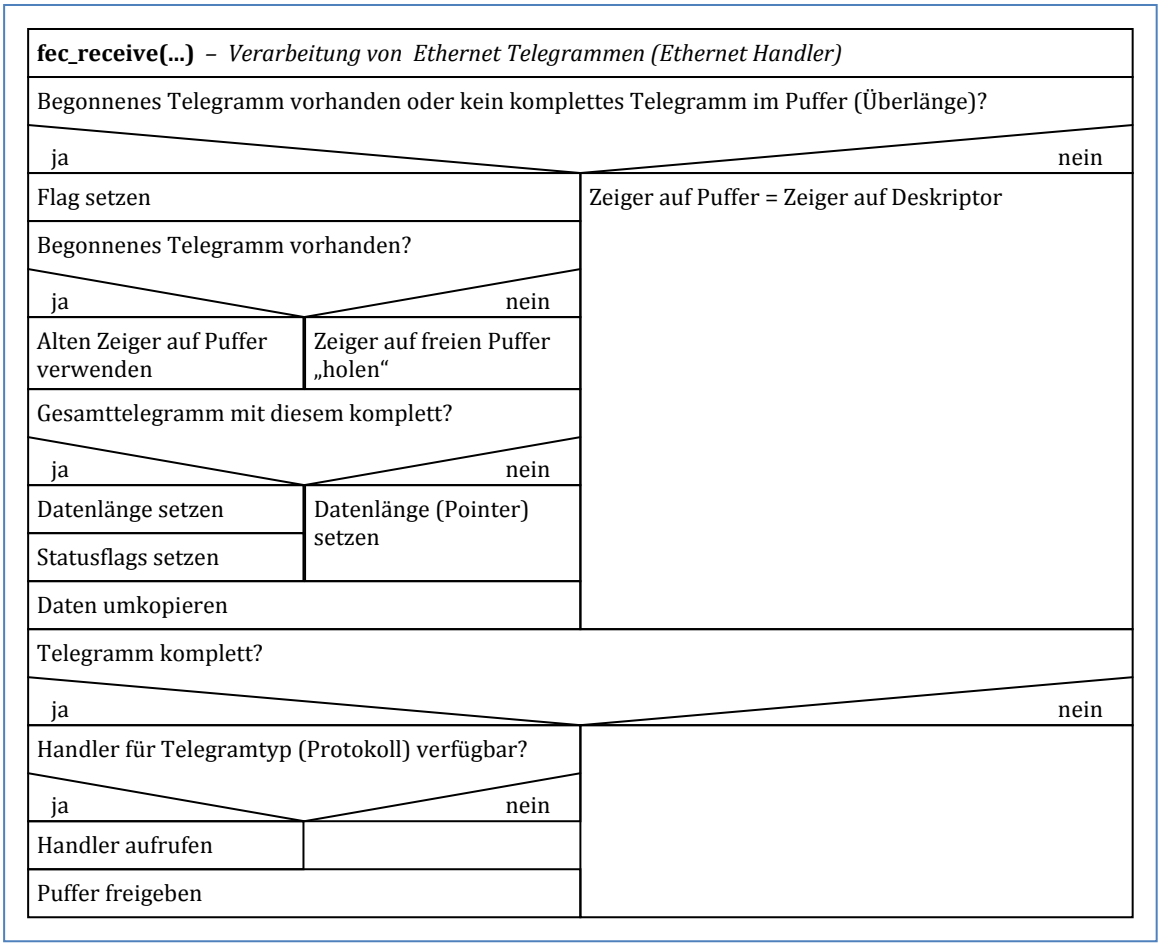


Bild 48: Verarbeitung von Ethernet Telegrammen

Die vorhandene Fehlerüberwachung in der Verarbeitung wurde aus Gründen der Übersichtlichkeit im Bild weggelassen. Es wird an den entsprechenden Positionen geprüft, ob benötigte Puffer frei sind und ob die Datenlängen passen. Bei Problemen wird das Gesamttelegramm verworfen.

Die endgültige Verarbeitung der Daten geschieht über die entsprechenden Handler. Folgende Struktur stellt die Möglichkeiten der Kommunikation mit dem KNG dar:

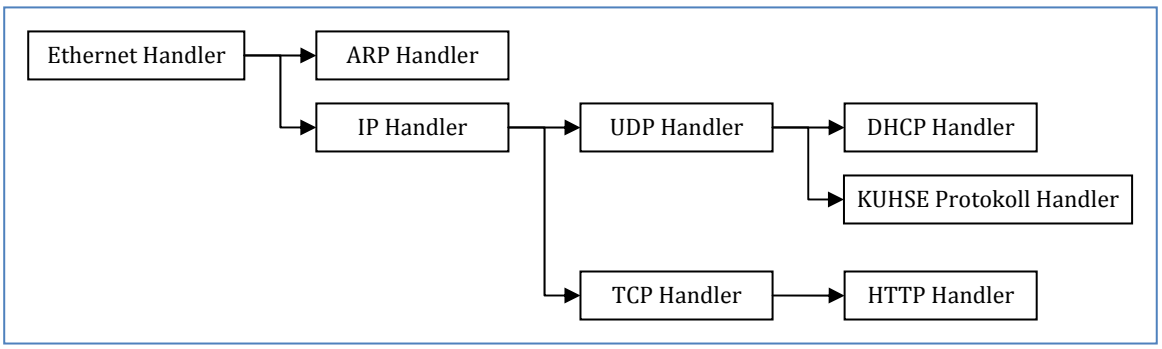


Bild 49: Netzwerk Telegrammverarbeitung (Handler)

Der KUHSE Protokoll Handler und der HTTP Handler für das vorgesehene Web-Interface sind noch nicht umgesetzt.

Der Ablauf in den jeweiligen Handlern ist nahezu identisch. Es werden die Daten im Telegramm-Header geprüft und die Nutzdaten verarbeitet. Handelt es sich bei den Nutzdaten um ein eingebettetes Protokoll, das unterstützt wird, so werden diese an den entsprechenden Handler übergeben.

6.6.7 Datenversand

Für den Datenversand sind in entsprechend umgekehrter Reihenfolge Funktionen implementiert, denen die zu versendenden Daten und die jeweils entsprechend notwendigen Header-Parameter übermittelt werden. Das Datenpaket wird dann entsprechend zusammengestellt.
Grundsätzlich wird ein Telegramm-Puffer vom Typ FBUF verwendet, in dem die Daten schon ab Beginn des Telegrammaufbaus an die entsprechende Position geschrieben werden. So wird vermieden, dass bei dem geschachtelten Aufbau der Telegramme ein häufiges Umkopieren der Daten erfolgen muss.

Für die Daten, die von der Unterstation seriell über PPP Telegramme an das KNG gesendet werden, wurde die Funktion udp_repeat() implementiert. Hier wird das UDP Telegramm aus dem PPP Rahmen übergeben und vom KNG über die Netzwerk-Schnittstelle weiter gesendet. Dabei wird auch die an die Unterstation vergebene IP Adresse (192.168.0.1) gegen die IP Adresse des KNG ersetzt. Anschließend muss die Prüfsumme neu berechnet werden. (siehe Quellcode-Ausschnitt 20)

```
int
udp_repeat ( NIF *nif, uint8 *dest, uint8* data, uint16 length)
{
  FBUF            Fbuf;
  uint8           *src;
  udp_frame_hdr   *udpframe;

  Fbuf.data = Fbuf.fdata;

  udpframe = UDP_MSG(Fbuf.data);

  memcpy ((void *) udpframe, data, length);

  // Set length //
  udpframe->length = length;

  // No checksum calcualation needed //
  udpframe->chksum = (uint16) 0;

  src = ip_get_myip(nif_get_protocol_info(nif,FRAME_IP));

  // Set packet length to length of the UDP frame //
  Fbuf.length = udpframe->length;

  return ip_send( nif,
                  dest,
                  src,
                  IP_PROTO_UDP,
                  (NBUF *) &Fbuf);
}
```

Quellcode 20: Weiterleitungs-Funktion udp_repeat()

7 Zusammenfassung und Ausblick

Entwickelt werden sollte ein Gateway Modul für das KUHSE TeleControl System, das zusammen mit dem Visualisierungs- und Steuerungssoftwarepaket KUHSE Data Publisher zur Fernwartung von Energieerzeugungsanlagen eingesetzt wird. Mit dem Gateway sollte die in den Anlagen befindliche TeleControl Unterstation mit einer Ethernet Netzwerkschnittstelle ausgerüstet werden. Statt über die bisherige Modem-Einwahlverbindung sollte über diese Schnittstelle eine Datenverbindung über das Internet zum TeleControl Master hergestellt werden. Die Internetverbindung muss dabei von einem im Netzwerk befindlichen Router zur Verfügung gestellt werden.
Fernziel ist, dass das KUHSE Netzwerk Gateway (KNG) selber zur Unterstation wird und die bisher eingesetzte Hardware ersetzt werden kann.
Als optionales Feature soll mit dem KNG auch eine serielle Datenverbindung per LWL zu anderen KUHSE Elektronikgeräten bzw. per EIA-485 und MPI zu Siemens SPS Steuerungen hergestellt werden können. Daten und Parameter dieser Geräte sollen ebenfalls per Netzwerk abrufbar bzw. einstellbar sein.

7.1 Stand der Entwicklung

Entwickelt wurde ein Gateway auf Basis des Freescale Coldfire Prozessors MCF 5270, der bereits über ein integriertes Netzwerkinterface und serielle Schnittstellen verfügt.
Extern wurden SD-RAM als Arbeitsspeicher und Flash Speicher als Programmspeicher sowie der Anschluss einer MiniSD Speicherkarte als Datenspeicher vorgesehen.
Das KNG verfügt über eine RJ-45 10/100-Base-T Ethernet Schnittstelle, eine LWL Schnittstelle nach KUHSE Standard und eine 9-polige serielle D-SUB Schnittstelle zur Anbindung der TeleControl Unterstation über EIA-232 bzw. zur Anbindung einer Siemens SPS über den auf EIA-485 basierenden MPI Bus.
Betriebs- und Fehlerzustände werden dem Benutzer über vier integrierte LEDs angezeigt.

Der Schaltplan der KNG Hardware wurde erstellt und ein PCB Layout angefertigt. Es wurde ein Prototyp gefertigt. (siehe Bild 50)

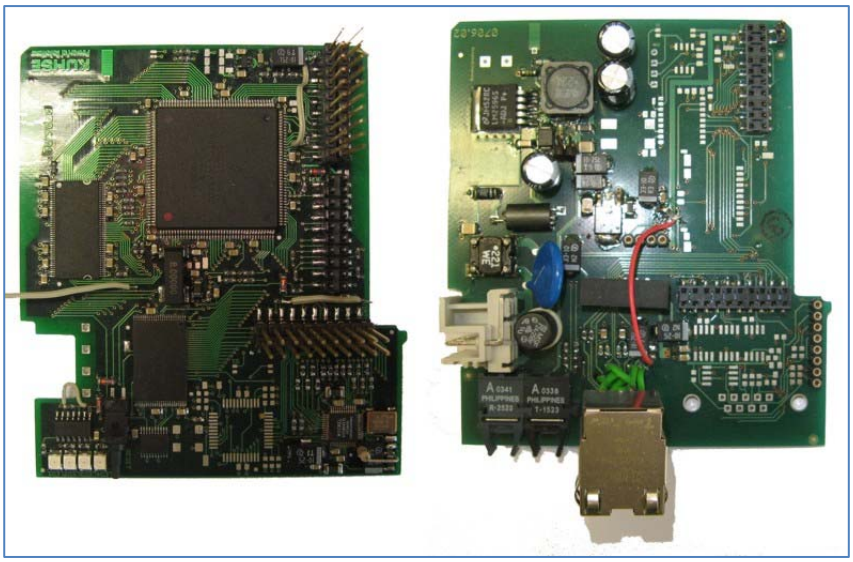


Bild 50: KNG Prototyp

Ein Firmware Programm wurde erstellt und die Grundfunktionalität getestet. Der Prototyp erfüllt bereits die Funktion des Netzwerk Gateways für die TeleControl Unterstation. Dabei wurde das Programm so gestaltet, dass für bestehende Anlagen das eingesetzte Modem der TeleControl Unterstation durch das KNG ersetzt werden kann, ohne dass eine Anpassung des Unterstations-Programms notwendig ist. Das KNG emuliert dazu das Modem und simuliert die Einwahl bei einem Internet Service Provider.

Gesamter Testaufbau mit TeleControl Unterstation:

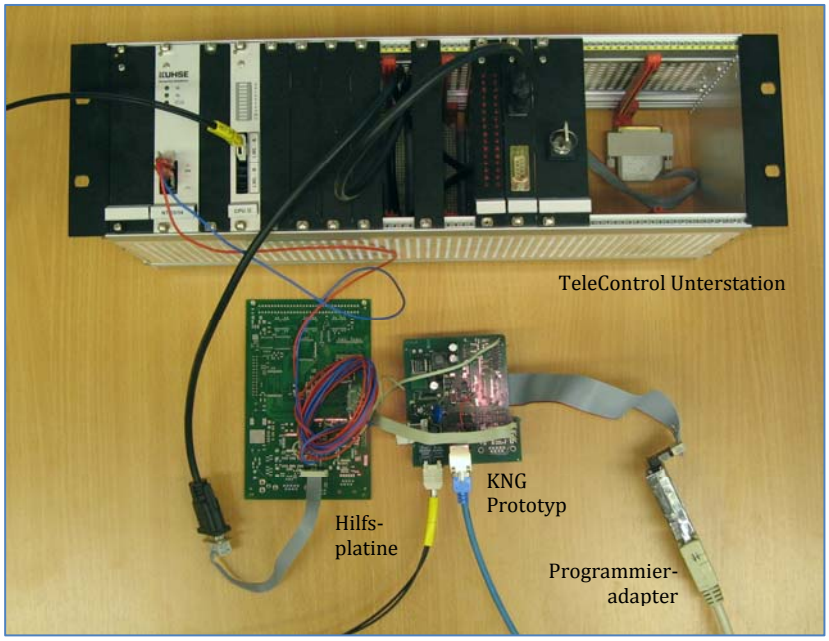


Bild 51: KNG Testaufbau

7.2 Bewertung der Arbeit

Das entwickelte Netzwerk Gateway erfüllt die geforderte Funktionalität und ist im gesteckten Preisrahmen zu fertigen. Das KNG ist in der gewählten Gehäuseform in der vorgesehenen Anlagenumgebung optimal einsetzbar.
Das KNG bietet durch den umfangreichen aber modular gestalteten Hardwareaufbau eine Vielzahl von zusätzlichen Einsatzgebieten, die kostengünstig durch entsprechende Hardware-Bestückung und Anpassung des Firmware-Programms erschlossen werden können. Dabei sind die Kosten für das Gerät selber immer dem Verwendungszweck angemessen.

Vor Erreichen der Serienreife sind noch einige Hardware-Fehler im PCB Layout zu beseitigen, die in den Schaltplänen bereits korrigiert sind. Außerdem muss die Software in einem umfangreichen Test ausgiebig geprüft werden.

7.3 Ausblick

Für das KNG ist eine Nullserien-Fertigung von zehn Stück für Ende Januar / Anfang Februar 2008 geplant. Bis dahin sollen die Hardware-Fehler beseitig sein und neue Platinen zur Verfügung stehen.

Mit der neuen Hardware sind dann EMV-Tests durchzuführen. Daran anschließend wird ein Feldtest mit ausgewählten Kunden gestartet – begleitet von internen Dauertests. Diese Tests dienen dazu, das Gerät zur Serienreife zu bringen und dabei gleichzeitig auf spezielle Anmerkungen der Kunden reagieren zu können. Damit ist gewährleistet, dass ein optimal gestaltetes Produkt in den Markt eingeführt werden kann.

Bis Ende des ersten Quartals 2008 wird außerdem bereits eine Sonderlösung des KNG als Netzwerk Gateway für eine Fremdsteuerung benötigt. Hiefür muss die Software entsprechend erweitert werden und ebenfalls ein Testlauf erfolgen.

Noch während den Testphasen wird die KNG Software daraufhin erweitert, dass das Gerät die TeleControl Unterstation auch vollständig ersetzen kann. Danach werden die Datenkopplungen per CAN oder MPI Bus integriert und getestet.

Tabellen-Verzeichnis

Bilder-Verzeichnis

Quellcode-Verzeichnis

Literaturverzeichnis

[1] BMWI Publikation „E-Energy – Informations- und kommunikationstechnologiebasiertes Energiesystem der Zukunft“ [Stand Mai 2007]

[2] Wikipedia Artikel „Regelleistung“ http://de.wikipedia.org/wiki/Regelleistung [15.10.2007]

[3] Artikel „Stadtwerke übermitteln Fernwirkdaten über Internetprotokoll“ etz Elektronik + Automation, VDE Verlag GmbH [Ausgabe 9/20007]

[4] KUHSE Data Publisher (DP) Handbuch TeleControl Client Kapitel 1 [Stand: 14.12.2004 08:09 Rev.Nr: 2]

[5] KUHSE Entwicklungsdokumentation „KNG Spezifikation“ von Marco Anetzberger

[6] Studienarbeit „Konzeption einer Multi-Schnittstellenkarte“ Marco Anetzberger (HAW Hamburg)

[7] Freescale Selectors Guide für Coldfire Prozessoren

[8] Datenblatt Spansion S29AL004D

[9] Datenblatt Micron MT48LC2M32B2

[10] Benutzer-Handbuch Freescale Coldfire MCF5270 [Revision 2]

[11] Übersicht Micrel Ethernet PHYs http://www.micrel.com/page.do?page=product-info/fastether_trans.jsp [19.10.2007]

[12] Datenblatt Micrel Ethernet PHY KSZ8041TL

[13] Datenblatt Tyco RJ-45 Buchsenserie 6605758

[14] RS Online-Katalog – Artikelbild von Artikel Nr. 171-1587 (fiber optic transmitter) http://www.rsonline.de [23.10.2007]

[15] Datenblatt Avago HFBR-0501 Series

[16] Datenblatt EIA-232 Treiber Sipex SP3232EU

[17] Wikipedia Artikel „EIA232“ http://de.wikipedia.org/wiki/EIA232 [23.10.2007]

[18] ProfiChip Produktinformation zum MPI12x

[19] Datenblatt National Semiconductor DS75176B

[20] Datenblatt Avago Optokoppler HCPL-0710

[21] Datenblatt Avago Optokoppler HCPL-0601

[22] Datenblatt Microchip SPI CAN-Controller MCP2515

[23] Datenblatt NXP CAN Bustreiber PCA82C250

[24] Wikipedia Artikel „SD Memory Card“ http://de.wikipedia.org/wiki/SD_Memory_Card [23.10.2007]

[25] RS Online-Katalog – Artikelbild von Artikel Nr. 529-3695 (SD-Kartenhalter) http://www.rsonline.de [23.10.2007]

[23] Datenblatt Molex SD-Kartenhalter Nr. 500525-1100

[27] Datenblatt Analog Devices Multiplexer ADG704

[28] Zeichnung Phoenix Contact Gehäuseunterteil ME 45 UT/FE KMGY

[29] Zeichnung Phoenix Contact Gehäuseoberteil ME 45 OT-1MSTBO KMGY

[30] CadSoft Online Herstellerwebseite EAGLE (Einfach anzuwendender grafischer Layout Editor) http://www.cadsoft.de [30.10.2007]

[31] Datenblatt ProfiChip MPI12x [Revision 2.01]
[32] Datenblatt Osram LEDs Typ TopLED
[33] Datenblatt Inverter 74XX04 (NXP)
[34] Datenblatt Transistor BC847 (Fairchild)
[35] Roboternetz Wissen – Artikel „RS232“
http://www.roboternetz.de/wissen/index.php/RS232 [01.11.2007]
[36] Schaltplan Demo-Board KSZ8041TL/FTL
[37] miniSD Pinout
[38] Beschreibung Phoenix Contact ME.../TBUS Elektronikgehäuse
[39] EM TEST Seminarunterlagen “EMV-gerechtes Design” [Version 2005]
[40] Fischer-Elektronik Katalog
[41] CodeSourcery GNU Toolchain für Coldfire Prozessoren
http://www.codesourcery.com/gnu_toolchains/coldfire [06.11.2007]
[42] IDM Computer Solutions Inc. UltraEdit Produktwebseite
http://www.idmcomp.com/index.php?name=UE_MoreFeatures [06.11.2007]
[43] Demo-Code für Freescale Coldfire MCF5272 Version/Revision 1c
[44] Wikipedia Artikel „Ethernet“
http://de.wikipedia.org/wiki/Ethernet [07.11.2007]
[45] RFC 826 – ARP
[46] Wikipedia Artikel „Address Resolution Protocol“
http://de.wikipedia.org/wiki/Address_Resolution_Protocol [07.11.2007]
[47] RFC 791 – IPv4
[48] Wikipedia Artikel „Internet Protocol“
http://de.wikipedia.org/wiki/Internet_Protocol [07.11.2007]
[49] RFC 2131 – DHCP für IPv4
[50] Wikipedia Artikel „Dynamic Host Configuration Protocol“
http://de.wikipedia.org/wiki/Dhcp [07.11.2007]
[51] RFC 792 – ICMP
[52] Wikipedia Artikel „Internet Control Message Protocol“
http://de.wikipedia.org/wiki/Icmp [07.11.2007]
[53] RFC 768 – UDP
[54] Wikipedia Artikel „User Datagram Protocol“
http://de.wikipedia.org/wiki/User_Datagram_Protocol [07.11.2007]
[55] RFC 793 – TCP
[56] Wikipedia Artikel „Transmission Control Protocol“
http://de.wikipedia.org/wiki/Transmission_Control_Protocol [08.11.2007]
[57] RFC 1945 – HTTP Version 1.0
[58] RFC 2616 – HTTP Version 1.1
[59] Wikipedia Artikel „Hypertext Transfer Protocol“
http://de.wikipedia.org/wiki/Hypertext_Transfer_Protocol [08.11.2007]
[60] RFC 821 – SMTP (ursprüngliche Version)
[61] RFC 2821 – SMTP
[62] Wikipedia Artikel „Simple Mail Transfer Protocol“
http://de.wikipedia.org/wiki/Smtp [08.11.2007]

[63] RFC 2554 – SMTP Erweiterung “Authentification”
[64] RFC 1661 – PPP
[65] RFC 1340 – “Assigned Numbers”
[66] RFC Sourcebook
http://www.networksorcery.com/enp/default0903.htm [09.11.2007]
[67] RFC 1340 – PPP Authentisierungsprotokolle
[68] MicroAPL Ltd Homepage (Tool CFInit)
http://www.microapl.co.uk [02.11.2007]

Anhang

A1: BMWI Publikation „E-Energy – Informations- und kommunikationstechnologiebasiertes Energiesystem der Zukunft“ [e-energy-richtlinien.pdf]

A2: KUHSE Data Publisher (DP) Handbuch TeleControl Client Kapitel 1 [HB_DP_TCCL_Kp1Einfuehrung_de17.pdf]

A3: KUHSE Entwicklungsdokumentation „KNG Spezifikation“ von Marco Anetzberger [KNG_Spezifikation.pdf]

A4: Studienarbeit „Konzeption einer Multi-Schnittstellenkarte“ [Konzeption_Multi-Schnittstellenkarte.pdf]

A5-26: Datenblätter:

- Freescale Selectors Guide für Coldfire Prozessoren [SG1006.pdf]
- Spansion S29AL004D [S29AL004D_00_a3_e.pdf]
- Micron MT48LC2M32B2 [MT48LC2M32_64MSDRAMx32_5_Micron.pdf]
- Freescale Coldfire MCF5270 Benutzer-Handbuch [MCF5271RM_Rev2.pdf]
- Micrel Ethernet PHY KSZ8041TL [KSZ8041TL-FTL.pdf]
- Tyco RJ-45 Buchsenserie 6605758 [ENG_CD_6605758_A.pdf]
- Avago HFBR-0501 Series [5988-1765EN.pdf]
- Sipex SP3232EU [sp3222eu_3232eu.pdf]
- ProfiChip Produktinformation zum MPI12x [DB_MPI12xx_2005_45-10.pdf]
- National Semiconductor DS75176B [DS75176B.pdf]
- Avago Optokoppler HCPL-0710 [AV02-0641EN.pdf]
- Avago Optokoppler HCPL-0601 [AV02-0170EN.pdf]
- Microchip SPI CAN-Controller MCP2515 [MCP2515_SPI-CAN-Controller_21801d.pdf]
- NXP CAN Bustreiber PCA82C250 [PCA82C250_5.pdf]
- Molex MiniSD-Kartenhalter [MiniSD-Conn_500525_0900766b80689659.pdf]
- Analog Devices Multiplexer ADG704 [ADG704.pdf]
- ProfiChip MPI12x [MPI12x_UM201.pdf]
- Osram LEDs Typ TopLED [T67kSMD_TOPLED.pdf]
- Inverter 74XX04 (NXP) [74LV04_2.pdf]
- Transistor BC847 (Fairchild) [BC-BC847.pdf]
- Schaltplan Demo-Board KSZ8041TL/FTL [KSZ8041TL-FTL Eval Board rev1.1.pdf]
- Beschreibung Phoenix Contact ME.../TBUS Elektronikgehäuse [me35-tbus.pdf]

A27-28: Zeichnungen:

- Phoenix Contact Gehäuseunterteil ME 45 UT/FE KMGY [2909358.dxf]
- Phoenix Contact Gehäuseoberteil ME 45 OT-1MSTBO KMGY [2709192.dxf]

A29: Demo-Code für Freescale Coldfire MCF5272 Version/Revision 1c [MCF5272SC.zip]

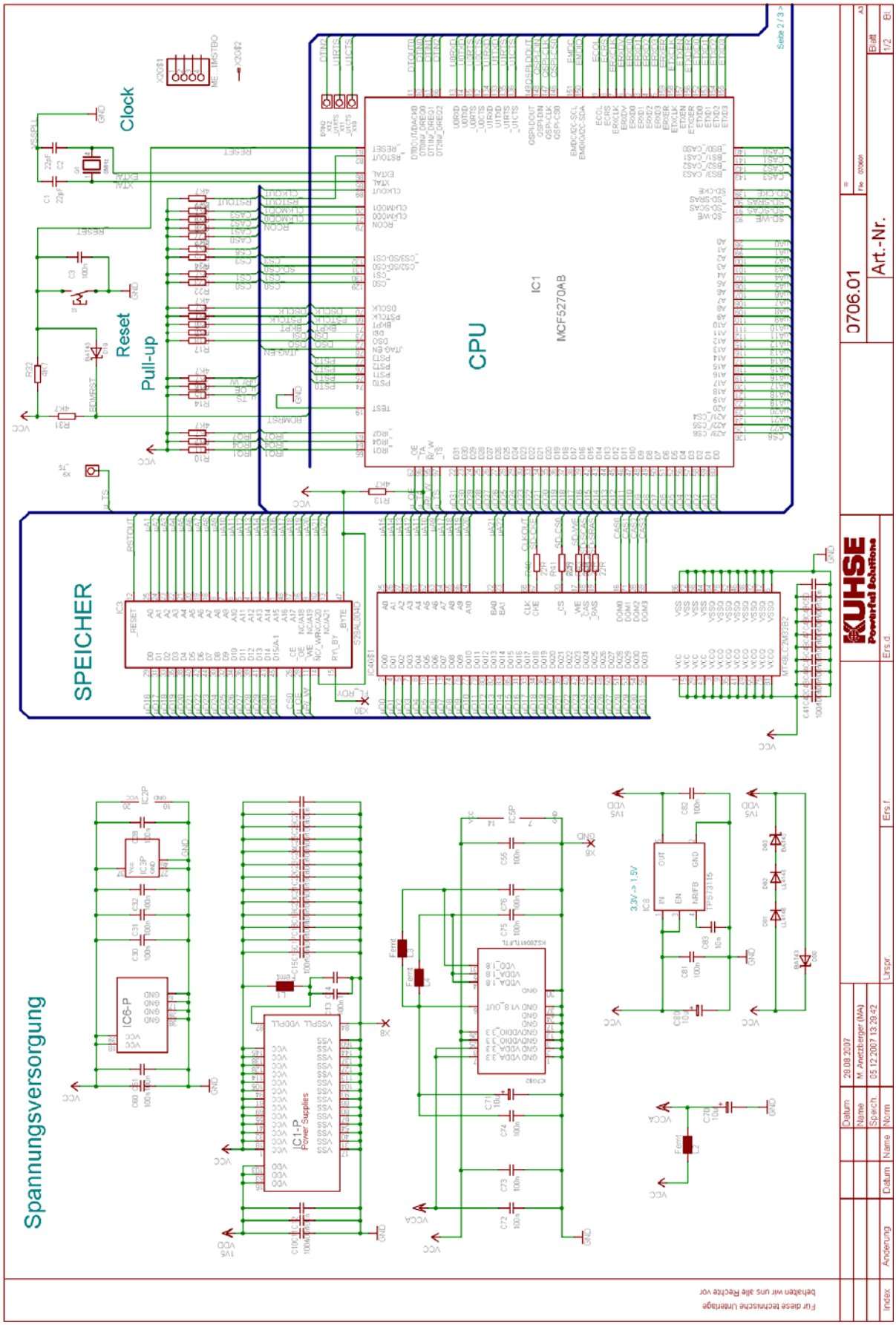
A30: RFC Quellen

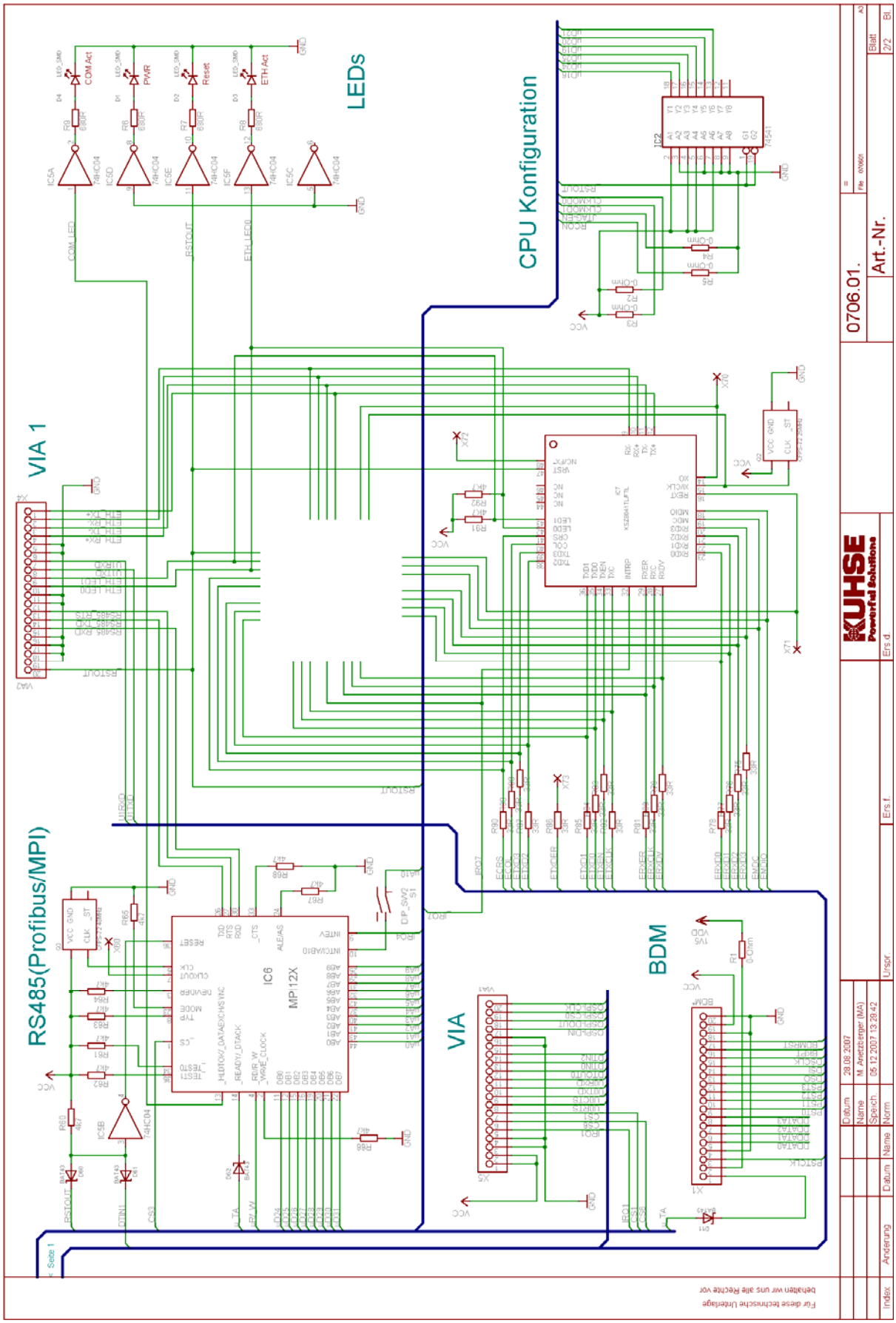
A31: EAGLE Dateien für Schaltplan und PCB-Layout

A32: Quellcode der KNG Software

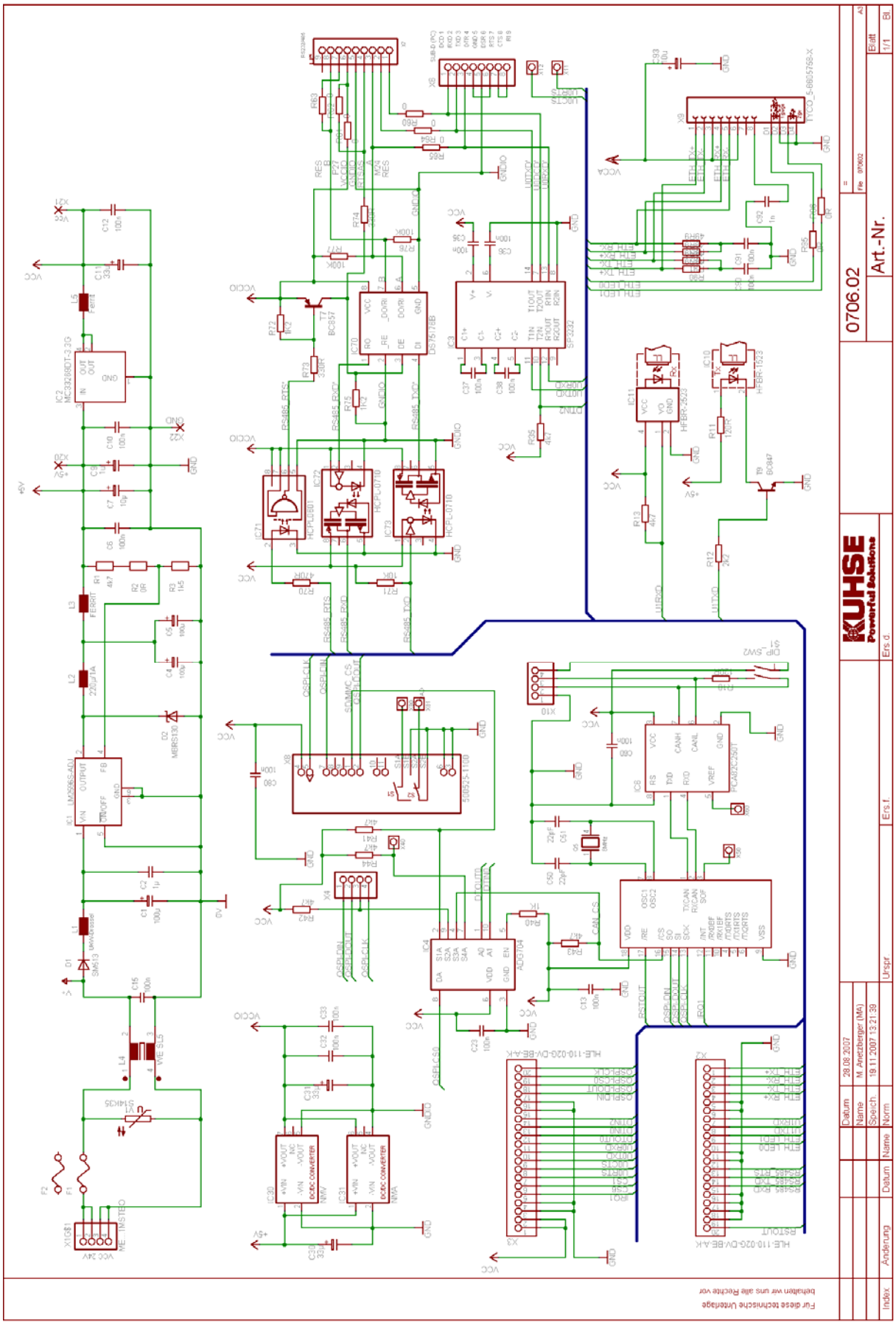
Hinweis: Die Anhänge A1 bis A32 sind in elektronischer Form auf einer CD abgelegt und beim Prüfer Professor Dr. Franz Schubert einzusehen.
Der Schaltplan des KNG (Teil des Anhangs A31) ist außerdem auf den folgenden Seiten abgedruckt.

Schaltplan des KNG (CPU-Platine)




LEDs
COM Act
PWR
Reset
ETH Act
CPU Konfiguration
VIA 1
RS485(Profibus/MPI)
VIA
BDM
IC6
MP112X
KUHSE
Powerful Solutions
0706.01.
Art.-Nr.

Schaltplan des KNG (Netzteilplatine)



Versicherung über die Selbständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §25(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Winsen (Luhe), den 05.12.2007

_____

(Marco Anetzberger)