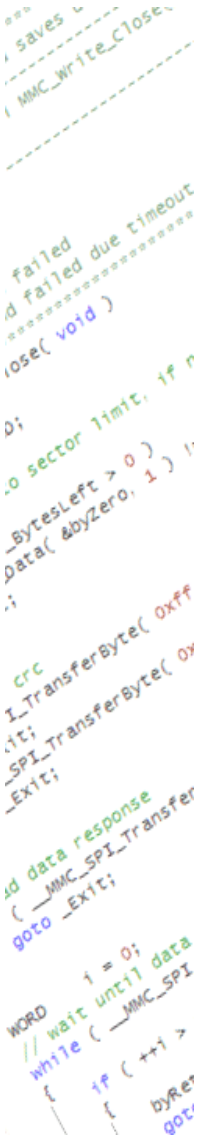
Diplomarbeit

Fachhochschule Augsburg
Studienrichtung Informatik

Fachhochschule Augsburg
University of Applied Sciences

GPS-Datenlogger

Entwicklung eines Langzeit-GPS-Datenaufzeichnungsgerätes auf der Basis eines RISC-Mikrocontrollers zur Analyse, Komprimierung und Speicherung von Wegdaten.



Verfasser: Christian Merkle
Abgabe: Wintersemester 2005 / 2006

Erstprüfer: Prof. Dr. Hubert Högl
Zweitprüfer: Prof. Georg Stark

Erstellungserklärung

Diplomarbeit gemäß § 31 der Rahmenprüfungsordnung für die Fachhochschulen in Bayern (RaPO) vom 18.09.97 mit Ergänzung durch die Prüfungsordnung (PO) der Fachhochschule Augsburg vom 15.12.94.

Ich erkläre hiermit, daß ich die Arbeit selbstständig verfaßt, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Königsbrunn, den 24. Oktober 2005

_____

Christian Merkle, Füssener Str. 61 ½, D-86343 Königsbrunn
http://www.cmerkle.de

ÜBERSICHT

INHALTSVERZEICHNIS

1. EINLEITUNG

1.1. AUFGABENSTELLUNG

Diese Diplomarbeit beschreibt die Planung und Entwicklung eines Gerätes zur Langzeitaufzeichnung von zurückgelegten Routen mit Hilfe eines GPS-Empfängers. Die Daten werden komprimiert auf einer Speicherkarte abgelegt und können bequem am PC analysiert und weiter verarbeitet werden. Der Schwerpunkt liegt im Entwurf und Aufbau der Hardware und dem Gehäuse des Prototyps, wie auch der Planung und Implementierung der Software für das Gerät. Zudem soll im Groben die Auswertsoftware am PC behandelt werden.

Die Hardware des mobilen Gerätes besteht aus einem Mikrocontroller, der mit folgender Peripherie verbunden ist:

- GPS-Empfänger zur Ermittlung der Position und Richtung
- LCD-Display zur Informationsausgabe und Bedienung des Gerätes
- IO-Modul mit Tastern und Leuchtdioden an der Frontplatte
- Speicherkarte zum Abspeichern der aufgenommenen Routen

Folgende Abbildung zeigt den groben Aufbau des Gerätes:

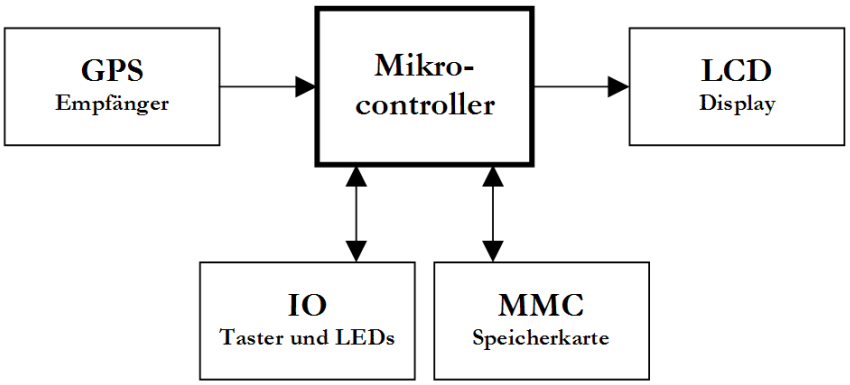


Abbildung 1 – Hardwareaufbau

1.2. ANWENDUNGSGEBIETE

Die Einsatzmöglichkeiten dieser Geräte sind nahezu unbegrenzt. Angefangen von Urlaubern, die sich später noch einmal in Ruhe die mit dem Motorrad oder dem Auto gefahrenen Strecken ansehen wollen über Wanderer, die ihren Bekannten die neu gefundenen Wanderwege zu einem einsamen See zeigen möchten. Sportler, die die Steigungen und die zurückgelegten Höhenunterschiede und damit die Schwierigkeit des Geländes analysieren müssen oder Firmen zur Straßen- oder Geländevermessung und anschließender Konvertierung in das Kartenformat. Kontrolle über die Fahrweise in vermieteten Fahrzeugen und die damit besuchten Orte oder sogar die Auswertung der Strategie eines polizeilichen oder militärischen Einsatzes. Prinzipiell

denkbar sind auch Routenaufzeichnungen in der Tierwelt um beispielsweise die Langzeitbewegung von Fischen oder Zugvögeln zu studieren.

1.3. AUFBAU DER DIPLOMARBEIT

Diese Arbeit gliedert sich in insgesamt acht Kapitel:

Kapitel	Nr.	Beschreibung
1	**Einleitung**	Diese Einleitung. Hier wird die Aufgabenstellung, die möglichen Einsatzgebiete, der Aufbau und der Syntax der Diplomarbeit näher erläutert.
2	**Grundlagen**	In diesem Kapitel wird grob auf die Grundlagen der verwendeten Teile eingegangen. Hier wird unter anderem der Mikrocontroller, GPS, MMC und das LCD-Display näher vorgestellt und die notwendigen Randbedingungen, wie Protokolle und Schnittstellen, erörtert.
3	**Hardware**	Hier wird die Erstellung der Hardware beschrieben: Von der Aufgabenstellung, der Planung der einzelnen Module, deren Entwicklung, Berechnungen, Aufbau und Tests bis hin zum Gehäuse und dem Zusammenbau der Komponenten.
4	**AVR-Software**	Die Planung, Implementierung und Tests der Software sind im vierten Kapitel das Thema und stellen damit das größte Kapitel in dieser Arbeit dar.
5	**PC-Software**	Das fünfte Kapitel beschäftigt sich mit der Auslesung und Auswertung der gespeicherten Daten auf dem PC und deren Analyse und Darstellung in den unterschiedlichsten Formen.
6	**Zusammenfassung**	Verbesserungsvorschläge, Anmerkungen, Problembeschreibungen und ein abschließendes Fazit.
7	**Begleit-CD**	Inhalt und Aufbau der Begleit-CD.
8	**Copyright und Lizenz**	Informationen zum Copyright und den verwendeten Lizenzen.
	Anhang	Glossar, Abbildungsverzeichnis, Tabellenverzeichnis, Listing-verzeichnis, Literaturverzeichnis und der Index.

Tabelle 1 – Aufbau der Diplomarbeit

1.4. SYNTAX

Das vorliegende Dokument besitzt folgende Syntax:

Element	Beschreibung
Standard Blockschrift	Zusammenhängender, normaler Text
Kursiv	Hervorhebungen im Text und neu eingeführte Begriffe
`Feste Breite`	Befehle, Code, Funktionen, Variablen
`Codeblock`	Listings oder Tabellen
Fußnote[1]	Zusätzliche Erklärungen zum Begriff am Seitenende
[BEZ1]	Quellenangabe aus dem Literaturverzeichnis
• **`void MyFoo(...)`**	Public Funktionsdefinition
• **`void MyFoo(...)`**	Private Funktionsdefinition

Tabelle 2 – Syntax

[1] Eine Fußnote

2. GRUNDLAGEN

2.1. ATMEL AVR RISC MIKROCONTROLLER

Die Serie von AVR-Mikrocontrollern von Atmel[2] bietet ein komplettes *System-On-A-Chip*, d.h. sie vereinen Prozessor, Speicher und Peripherie auf einem Chip und benötigen nur noch wenig zusätzliche Hardware. Die extrem preiswerten ICs bekommen auch Privatanwender für 2 EUR für die einfachsten Versionen bis hin zu unter 10 EUR für die größten Ausstattungspakete.

Die Hauptbestandteile und Eigenschaften der AVR-Serie [AVRMega32] sind:

- Ein hochperformanter und energiesparender 8-Bit RISC-Prozessor mit 32 Registern
- 131 Befehle, von denen die meisten nur einen Taktzyklus benötigen
- Flash Speicher für statischen Programmcode, Controller ISP[3] geeignet
- SRAM als Hauptspeicher
- EEPROM als nichtflüchtiger Speicher
- Reichhaltige Peripherie, wie unter anderem 8- und 16-Bit Timer, RTC[4], PWM-Ausgänge[5], ADC-Wandler[6], 2-Draht serielles Interface, UART (seriell, synchron und asynchron), SPI[7], Watchdog Timer, Analoge Komparatoren, usw.
- Frei konfigurierbare IO-Leitungen (Ports)
- JTAG für standardisiertes Debuggen ohne kostspielige Extra-Hardware direkt im Zielsystem
- Breiter Versorgungsspannungsbereich von 2.7 Volt bei den *Low-Voltage*-Modellen bis hin zu 5.5 Volt bei den normalen Ausführungen
- Taktfrequenzen von bis zu 16 MHz, teilweise sogar 20 MHz

Diese Eigenschaften, die breite Verfügbarkeit und die einfache Programmierung dieser Mikrocontroller machen diese Serie optimal für unser Gerät. Die meisten Versionen sind als DIP-Version für Prototypen und als SMD-Version für die Serienproduktion erhältlich.

2.1.1. DER AVR MEGA32

Die Unterschiede in der AVR-Serie sind vor allem im vorhandenen Speicher, der Anzahl der IO-Leitungen und der Peripherie zu finden. Der für dieses Projekt letztendlich verwendete Mikrocontroller ist der AVR Mega32. Er bietet unter anderem folgende Eigenschaften:

- 32 kB Flash
- 2 kB SRAM
- 1 kB EEPROM
- 32 IO-Leitungen (vier 8-Bit Ports)

---

[2] Homepage: www.atmel.com
[3] In System Programming – Neuprogrammierung direkt im Zielsystem
[4] Real Time Counter / Real Time Clock
[5] Glitch-free, Phase Correct Pulse Width Modulator
[6] Analog To Digital Converter
[7] Serial Peripheral Interface

- Zwei 8-Bit und ein 16-Bit Timer, USART, SPI
- Hardware Multipliziereinheit in nur 2 Taktzyklen
- Und einige andere Peripherien, die allerdings nicht genutzt wird, wie eine RTC, vier PWM-Ausgänge, acht 10-Bit ADC-Kanäle, ein Two-Wire-Serial-Interface und einen Watchdog-Timer

Das Blockschaltbild aus dem Atmel Datenblatt [AVRM32]:

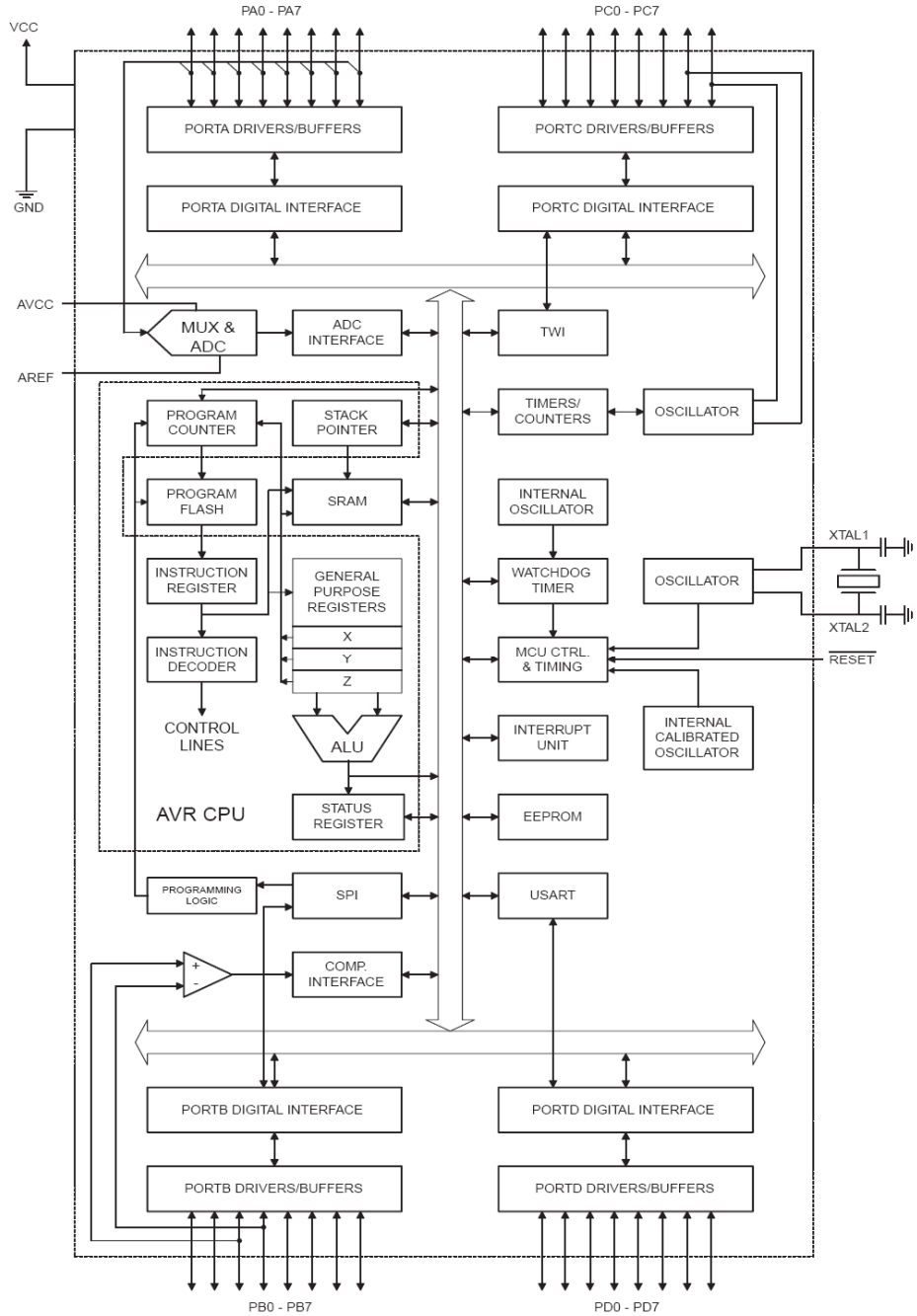


Abbildung 2 – Blockschaltbild Atmel AVR Mega32 aus [AVRM32]

Man benötigt zum Betrieb nur sehr wenig externe Komponenten: Einen Kondensator zum Auffangen von Stromspitzen und optional einen Quarz mit zwei Kondensatoren auf Masse zur Takterzeugung.

Für weniger zeitkritische Applikationen kann auch der interne Oszillator zum Einsatz kommen. Da mit Verwendung der seriellen Schnittstelle zur Vermeidung von Übertragungsfehlern die Taktzeiten ziemlich genau einzuhalten sind, verwendet dieses Projekt einen externen Quarz mit 8 MHz.

2.1.2. Die Pinbelegung eines AVR Mega32

Zwei Pins (10, 11) werden zur Stromversorgung benötigt, zwei weitere zur Takterzeugung (12, 13), einer als Reset-Leitung (9) und drei weitere für die AD-Wandler (30-32), die allerdings nicht benutzt werden. Die restlichen 32 Pins sind direkt die IO-Leitungen, teilweise mit zusätzlichen Sonderfunktionen:

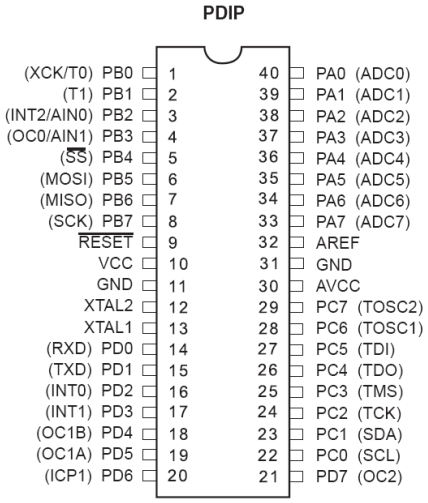


Abbildung 3 – Pinbelegung Atmel AVR Mega32 aus [AVRM32]

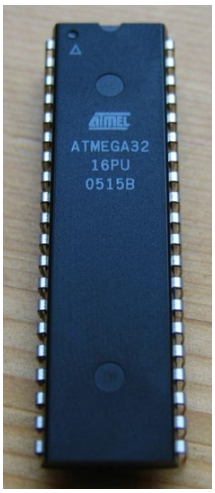


Abbildung 4 – Atmel AVR Mega32

2.1.3. Programmierung des Mikrocontrollers

Die AVR-Mikrocontroller können über verschiedene Methoden programmiert werden: Von der aufwendigen parallelen High-Voltage Methode, bei der die Resetleitung auf +12V gelegt wird und die Daten parallel über zwei Ports in den Flash- und EEPROM-Speicher geladen werden, wobei sich der Chip nicht im Zielsystem befindet darf, sondern in einem speziellen Programmiergerät. Bis hin zu der üblicherweise verwendeten Methode, die auch in diesem Projekt angewandt wird: Der Programmiervorgang läuft direkt im eingebauten Zustand im Zielsystem über die SPI-Schnittstelle ab und benötigt dort die Resetleitung und nur noch 3 weitere Pins des Controllers, die sich jedoch im normalen Betriebsmodus auch noch für alle anderen Aufgaben benutzen lassen. Diese Methode nennt man ISP (*In System Programming*).

2.1.3.1. Assembler Programmiersprache

Die RISC-Prozessoren werden mit einer einfachen und übersichtlichen, jedoch sehr mächtigen, Assemblerprogrammiersprache programmiert. Jeder Befehl ist 16-Bit breit, u.a. komplizierte Sprungbefehle benötigen jedoch zwei 16-Bit Slots. Bis auf Sprung-, Programmspeicher- und Multiplizierfunktionen benötigen die Befehle nur einen Takt zur Ausführung. Dem Prozessor stehen 32 allgemein verwendbare Register und für einen RISC-Prozessor überraschend viele Adressierungsmodi mit Pointer-Registern zur Verfügung.

2.1.3.2. HOCHSPRACHE C

Um den Anforderungen großer Projekte zu genügen und bestmöglich portabel zu sein, werden Hochsprachen eingesetzt. Auch hier gibt es eine schier unüberschaubare Masse von verschiedenen Sprachen und Tools für diese Prozessorgeneration. Unter der frei verfügbaren Software ist jedoch mit Abstand das avr-gcc Toolkit das Mächtigste. Für eine genauere Beschreibung der verwendeten Software sei hier auf das AVR-Softwarekapitel, *Kapitel 4.1 - Verwendete Software* verwiesen.

2.2. GPS – GLOBAL POSITIONING SYSTEM

2.2.1. FUNKTIONSWEISE VON GPS

GPS bedeutet *Global Positioning System* [GPS] und ist ein satellitengestütztes Navigationssystem, das vom US-Verteidigungsministerium betrieben wird, allerdings auch für zivile Nutzung freigegeben ist. Seit Mai 2000 wurde auch die künstlich verschlechterte Genauigkeit abgeschaltet, die eine zivile Nutzung von nur 100m Genauigkeit zur Folge hatte. Dadurch sind nun auch für Zivilisten Genauigkeiten von weniger als 10m zu erreichen.

Das GPS besteht aus einer Anzahl von derzeit etwa 31 Satelliten, die auf einer der sechs verschiedenen Bahnebenen um die Erde kreisen. Damit wird gewährleistet, daß von jedem Ort der Erde immer mindesten vier gleichzeitig sichtbar sind. An vielen Orten sind es aber meist deutlich mehr, praktische Werte reichen von fünf bis sogar neun gleichzeitig sichtbaren. Theoretisch wären allerdings nur insgesamt 24 Satelliten nötig, um die oben genannten Bedingungen zu erfüllen.

Die einzelnen Satelliten strahlen nun ständig Signale aus, aus denen die GPS-Empfänger mit Hilfe der Signallaufzeiten die aktuelle Position bestimmen können. In der Theorie reichen sogar nur drei Satelliten aus, allerdings wäre dazu eine exakte Uhr nötig, die alle Empfänger ohne Atomuhr nicht besitzen. Mit Hilfe des vierten Signals können die fehlenden Informationen aber berechnet werden. Somit ist gewährleistet, daß an jedem Ort der Erde eine Positionsbestimmung möglich ist.

Da die Signale im GHz-Bereich liegen (genau 1575.45 MHz für L1 und 1227.60 MHz für L2), verhalten sich die Wellen ähnlich dem Licht. Deshalb ist, zumindest mit heutigen Empfängern, ein direkter Sichtkontakt zu den Satelliten notwendig. Schon ein Baum mit Laub kann einen billigen Empfänger empfindlich stören. Zur Zeit sind aber schon Geräte in der Entwicklung, die in Gebäuden, ja sogar noch in Tiefgaragen funktionieren sollen.

Der Satellit sendet die Signale auf zwei verschiedenen Frequenzen, den oben bereits erwähnten L1- und L2-Freuquenzen. Auf der ersten liegen die Nutzdaten mit Informationen zum jeweiligen Satelliten, wie Datum, ID-Nummer, Bahnen, verschiedene Korrekturen und ein paar anderen Daten. Auf der zweiten Frequenz wird nur ein pseudozufälliger Code aus Einsen und Nullen gesendet. Pseudozufällig deswegen, da damit Interferenzen verringert werden und alle Satelliten auf der gleichen Frequenz senden können. Es werden zwei verschiedene Frequenzen eingesetzt, da sich damit ionosphärische Effekte korrigieren lassen, die sonst die Signallaufzeiten beeinflussen.

Die erreichbare Genauigkeit ist in mehrere Klassen eingeteilt: Der SPS, *Standard Positioning Service*, ist für die zivile Nutzung vorgesehen und erreicht Genauigkeiten von 15m, meistens sogar jedoch besser als 10m. Die zweite Klasse ist der PPS (*Precise Positioning Service*), der allerdings nur für das amerikanische Militär nutzbar ist. Über die wirkliche Genauigkeit dieser Systeme existieren keine offiziellen Informationen, sie dürften aber deutlich besser als die des SPS sein.

Um nun noch genauere Werte zu bekommen, setzt man DGPS (*Differential GPS*) ein. Die Funktionsweise ist so einfach wie genial: Man stellt stationäre GPS-Empfänger auf, welche die Signale, wie jeder andere Empfänger auch, mit ihren Fehlern und Ungenauigkeiten, auswerten. Der Ort des stationären Empfängers ist exakt bekannt. Nun berechnet der Empfänger den Unterschied der GPS-Messung zu seiner genauen Position, den sogenannten Differenzwert. Dieser wird dann per Funk ausgestrahlt.

Jeder andere GPS-Empfänger kann nun gleichzeitig zu dem von ihm ermittelten GPS-Signal die differentiellen Funksignale empfangen und damit eine viel genauere Positionsbestimmung durchführen. Die neuen Werte sind damit auf 1-5m genau.

Es lassen sich aber nicht nur die absolute Position (Breiten-, Längengrad und die Höhe) berechnen, sondern auch die relativen Daten wie Geschwindigkeit und Richtung. Dies wird durch den Dopplereffekt ermöglicht, der die Signale staucht oder streckt (vgl. Audiotechnik). Die relativen Daten sind noch dazu wesentlich genauer. Somit läßt sich die Geschwindigkeit auf $0.1\,^{m}/_{s}$, als etwa $0.4\,^{km}/_{h}$ genau bestimmen.

Um die ermittelten Werte eines GPS-Empfängers zu verarbeiten, senden diese in einem standardisiertem Format, dem NMEA 0183 Format, das im nächsten Abschnitt genauer behandelt wird.

2.2.2. NMEA 0183

NMEA 0183 ist das Standard-Protokoll von GPS-Empfängern, um Daten zu übermitteln [NMEA]. Der Ausgang ist RS-232 kompatibel und verwendet 4800 Baud, 8 Datenbits, keine Parität und ein Stopbit, kurz *8N1*.

Alle NMEA Datensätze sind lesbarer ASCII-Text. Jeder davon beginnt mit einem Dollarzeichen (`$`) und endet mit einer neuen Zeile, also einem *carriage return linefeed* (`<CR><LF>`).

Die einzelnen Daten sind durch Kommas voneinander getrennt, wobei alle Kommas vorhanden bleiben müssen, auch wenn manche Felder nicht mitgeschickt werden, da sie als Trennzeichen dienen.

Ein Befehl startet mit `$iiccc`, wobei `ii` eine optionale Geräte-ID ist und `ccc` das Kommando. Optional wird jeder Datensatz mit einem Stern (*) und anschließend einer Checksumme abgeschlossen.

Beispiel:
`$GPBSP,p1,p2,,p4*hh`

Das Dollarzeichen markiert einen neuen Befehl. `GP` ist die Geräte-ID. Der eigentliche Befehl ist `BSP`. Nach dem ersten Komma kommen die einzelnen Parameter, angefangen mit `p1` und `p2`. Der dritte Parameter ist leer gelassen und anschließend folgt der vierte. Zuletzt steht noch ein Byte – geschrieben als HEX-Wert – als Checksumme zur Überprüfung des gesamten

Datensatzes auf Übertragungsfehler. Abgeschlossen wird der Befehl durch eine neue Zeile (<CR><LF>).

Für eine vollständige Liste der NMEA-Kommandos siehe die NMEA-Spezifikation auf der Begleit-CD. Die für das Projekt verwendeten Befehle sind:

2.2.2.1. GGA

GGA - Global Positioning System Fix Data

```
$GPGGA,hhmmss.sss,llll.llll,a,lllll.llll,a,x,xx,x.x,x.x,M,x.x,M,x.x,xxxx*hh
$GPGGA,130122.880,4816.8350,N,01052.8448,E,1,05,1.3,565.2,M,,,,0000*0C

130122.880       Uhrzeit                              13:01:22 (GMT Zeit)
4816.8350, N     Breitengrad                          48° 16.8350' Nord
01052.8448, E    Längengrad                           10° 52.8448' Ost
1                Qualität                             0 = ungültig
                                                      1 = GPS
                                                      2 = DGPS
05               Anzahl der empfangen Satelliten      5
1.3              Horizontale Abweichung
565.2, M         Höhe über Meeresspiegel und Einheit  565.2 m
```

2.2.2.2. RMC

RMC - Recommended minimum specific GPS/Transit data

```
$GPRMC,hhmmss.sss,A,llll.llll,a,lllll.llll,a,x.xx,x.x,ddmmyy,x.x,a*hh
$GPRMC,130122.880,A,4816.8350,N,01052.8448,E,0.05,020.3,290305,,*10

130122.880       Uhrzeit                              13:01:22 (GMT Zeit)
A                Navigationswarnung                   A = OK, V = Warnung
4816.8350, N     Breitengrad                          48° 16.8350' Nord
01052.8448, E    Längengrad                           10° 52.8448' Ost
0.05             Geschwindigkeit in Knoten            0.05 knots
020.3            Kurs in Grad                         20.3°
290305           Datum                                29.03.2005
```

2.2.2.3. GSA

GSA - GPS DOP and active satellites

```
GSA,A,3,04,05,,09,12,,,24,,,,,2.5,1.3,2.1*39

A                A = Automatischer Wechsel von 2D <-> 3D, M = manuell
3                3D Modus
04,05, …         ID der verwendeten Satelliten (Platz für maximal 12)
2.5              PDOP (dilution of precision)
1.3              Horizontal dilution of precision (HDOP)
2.1              Vertical dilution of precision (VDOP)
```

Die DOP-Werte geben die Qualität und die Genauigkeit des Signals an.

2.2.2.4. GSV

GSV - Satellites in view

```
GSV,2,1,08,01,40,083,46,02,17,308,41,12,07,344,39,14,22,228,45*75

2               Anzahl der GSV Datensätze für die Vollständige Information
1               Datensatz 1 von 2
08              Anzahl der sichtbaren Satelliten
01              Satelliten ID
40              Elevation in Grad
083             Azimuth in Grad
46              Signalstärke, je größer, um so besser
```

Bis zu vier Wiederholungen pro Datensatz und bis zu drei Datensätze (Maximale Anzahl von 12 sichtbaren Satelliten).

2.2.3. HOLUX GM-210

Als primärer Empfänger für den GPS-Datenlogger wird ein Holux[8] GM-210 verwendet. Holux hat den zur Zeit größten Marktanteil an sogenannten GPS-Mäusen. Alternativ können aber auch alle anderen NMEA kompatiblen Empfänger an der seriellen Schnittstelle angeschlossen werden.

Abbildung 5 – Holux GPS Empfänger

Einige technischen Daten [GM210] sind:

- SiRF Star II Chipsatz
- Bis zu 12 Satelliten parallel
- Positionsaktualisierungsrate von genau 1 Hz
- Startzeit kleiner als 3 Sekunden, wenn Gerät weniger als 25 Minuten ausgeschaltet war
- Genauigkeiten:
  - GPS: Position 5-25 m, Geschwindigkeit 0.1 m/s, Zeit 1 µS
  - DGPS[9]: Position 1-5 m, Geschwindigkeit 0.05 m/s, Zeit 1 µS

[8] Homepage: www.holux.com
[9] Differential GPS: Modus zur Erhöhung der Genauigkeit von Empfängern

- Maximale Werte:
  - Höhe: 18 km
  - Geschwindigkeit: 515 m/s
  - Beschleunigung: 4 g
  - Temperatur: -40° C bis +80° C
  - Versorgungsspannung: 4.5 bis 5.5 Volt
  - Stromaufnahme: < 170 mA
- NMEA 0183 v2.2 Protokoll, mit folgenden Kommandos: GGA, GSA, GSV, RMC und optional GLL, VTG und SiRF Binary
- Interface: RS-232 und CMOS-/TTL-Level

Der Holux GM-210 hat eine 6-poligen Mini-DIN Buchse mit folgender Belegung [GM210]:

Pin	Signal	Pegel
1	Transmit (TX)	RS-232
2	Versorgungsspannung (VDD)	+5V
3	Transmit (TX)	CMOS
4	Masse (GND)	0V
5	Receive (RX)	CMOS
6	Receive (RX)	RS-232

Tabelle 3 – Pinbelegung GM210

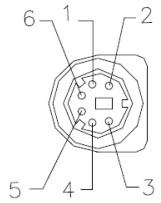


Abbildung 6 – Pinlayout, 6-polige Mini-DIN Buchse, aus [GM210]

2.3. MMC – MultiMedia Card

Um große Datenmengen persistent auf einem Medium zu speichern, benutzt man Speicherkarten. Mittlerweile gibt es diese in den unterschiedlichsten Formen, Speichergrößen, Geschwindigkeiten und Abmessungen. Für den Datenlogger sollte es eine von den Abmessungen möglichst kleine, weit verbreitete und damit günstige und einfach anzuschließende Speicherkarte sein.

Am besten dafür eignen sich MMC- und SD-Karten[10], die bis auf die Bauhöhe und ein paar Sicherheitsmerkmale identisch sind. Sie benötigen 2.7 bis 3.6 Volt Versorgungsspannung und nutzen zur Kommunikation das SPI Protokoll und benötigen damit nur vier Datenleitungen.

2.3.1. SPI – Serial Peripheral Interface

SPI steht für *Serial Peripheral Interface*, ein serielles Protokoll zur Kommunikation mit externer Peripherie. Ein Kommunikationspartner muß den Ablauf und Zeitpunkt der Übertragung bestimmen, meist der Hauptprozessor. Diesen nenn man dann *Master*. Der Gesprächspartner auf der andern Seite, der *Slave*, kann nicht selbständig eine Übertragung starten sondern nur auf Zutun des Masters antworten. Es ist synchron, d.h. es existiert eine zusätzliche Taktleitung zur Abgrenzung der einzelnen Bits. Wie schon angesprochen benötigt SPI insgesamt vier Datenleitungen:

[10] MultiMedia Card, Secure Digital Card

Pin	Beschreibung	Richtung, Master	Richtung, Slave
MOSI	Master Out, Slave In	Out	In
MISO	Master In, Slave Out	In	Out
SCK	Serial Clock	Out	In
/SS	Slave Select	Out	In

Tabelle 4 – Pinbelegung SPI

Die kleinste zu übertragende Einheit ist das Byte. Es wird immer eines auf einmal übertragen und zwar gleichzeitig vom Master zum Slave und umgekehrt. Pro Taktsignal überträgt die Hardware jeweils ein Bit mit Hilfe von Schieberegistern:

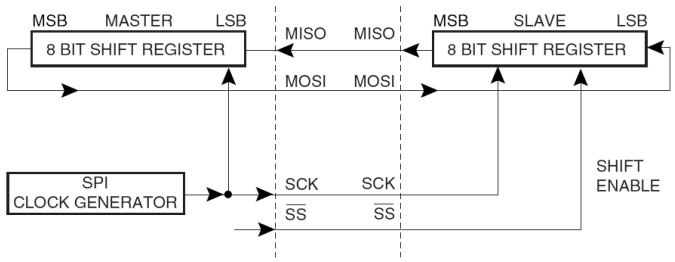


Abbildung 7 – Funktionsweise SPI, aus [AVRM32]

Der verwendete Mikrocontroller AVR Mega32 hat ein Hardware-SPI integriert, das die Low-Level Kommunikation über den SPI-Bus übernimmt.

2.3.2. MMC SPI Protokoll

Die MMC-Karten werden über das SPI-Protokoll angesprochen. Dieses ist ziemlich komplex und die genaue Beschreibung würde den Rahmen dieses Dokumentes sprengen. Für genauere Informationen sei hier auf das Datenblatt von Hitachi [MMC] verwiesen.

In diesem Abschnitt wird eine Übersicht über die MMC-Karten und eine Einführung in deren Programmierung gegeben.

Der maximale SPI-Takt für die Karte ist laut Spezifikation auf 20MHz festgelegt.

Zunächst einmal die Pinbelegung:

2.3.2.1. Pinbelegung

Pin	Signal	Beschreibung
1	/CS	Chip Select
2	DI	Data In
3	GND	Ground
4	VDD	Power Supply
5	SCK	Serial Clock
6	GND	Ground
7	DO	Data Out

Tabelle 5 – Pinbelegung MMC-Karte

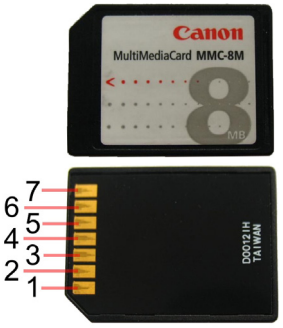


Abbildung 8 – MMC-Karte

2.3.2.2. BEFEHLSAUFBAU

Es gibt verschiedene Arten von Befehlen: Einfache Befehle, Lesebefehle, Schreibbefehle und Streamingbefehle. Letztere werden aber von uns nicht genutzt.

Die Karte beherrscht laut Spezifikation über 60 Befehle, wobei für die Grundkommunikation aber nur wesentlich weniger nötig sind: Hier eine Tabelle mit der Minimalkonfiguration der später von unserem Treiber verwendeten Befehle:

Befehl	Befehl	Argument	Response	Beschreibung
CMD0	RESET	-	R1	MMC-Karte führt Reset aus
CMD1	INIT	-	R1	Aktiviert den Initialisierungsprozeß
CMD9	SEND_CSD	-	R1	Anfrage nach card-specific-data
CMD17	READ	data address	R1	Liest Sektor ein
CMD24	WRITE	data address	R1b	Schreibt einen Sektor

Tabelle 6 – MMC, Minimalkonfiguration Befehle

BEFEHL

Ein kompletter Befehl besteht aus sechs Bytes: Dem Befehl an sich, dem Argument und einer Checksumme, die aber bei dem SPI-Modus, bis auf Befehl `CMD0`, nicht verwendet wird:

	Befehl	Argument				Checksumme
Byte	[0]	[1]	[2]	[3]	[4]	[5]

Tabelle 7 – MMC, 6 Byte Befehlsaufbau

RESPONSE

Nach jedem übertragenem 6-Byte-Befehl gibt die Karte eine kurze Antwort, einen Response, zurück. Laut Spezifikation gibt es mehrere verschieden aufgebaute Typen, aber bei den von uns verwendeten Befehlen müssen wir nur den Response R1 und R1b näher betrachten:

- **Response R1:**

Ein einzelnes Byte, das den aktuellen Status zurückliefert:

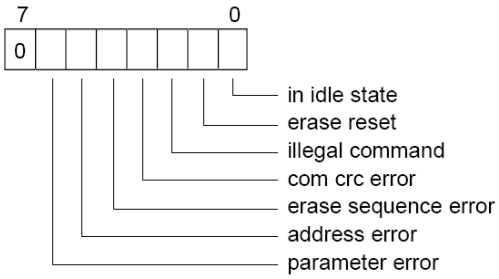


Abbildung 9 – MMC, Response R1, aus [MMC]

- **Response R1b:**

Wie bei R1, ein einzelnes Byte, das den aktuellen Status zurückliefert, nur danach kann noch eine Busy-Zeit kommen. Solange die Karte nach dem ersten Byte Nullen zurückliefert, ist sie noch beschäftigt.

Je nach Befehlstyp ist nach dem Response der Befehl schon abgeschlossen, oder es werden dann weitere Daten ausgetauscht.

2.3.2.3. BEFEHLSTYPEN

Jeder Befehlstyp, also einfacher Befehl, Lesebefehl oder Schreibbefehl hat einen bestimmten Aufbau und Ablauf. Im nächsten Abschnitt werden die von uns verwendeten Befehle näher erläutert:

EIN EINFACHER BEFEHL

Ein einzelner Befehl ohne Datenaustausch (Reset, Init, …):

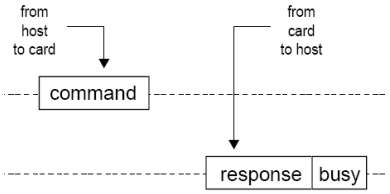


Abbildung 10 – MMC, einfacher Befehl, aus [MMC]

Und hier die genauere, bitweise Kommunikation:

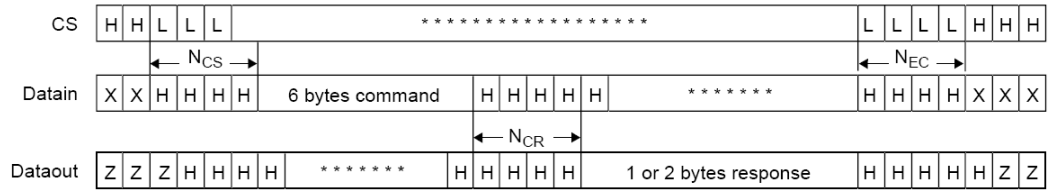


Abbildung 11 – MMC, einfacher Befehl, Details, aus [MMC]

EIN LESEBEFEHL:

Um Daten aus der Karte auszulesen, muß der Host erst ein Lesekommando schicken, dann den Response-Block abwarten und anschließend alle Daten empfangen. Damit ist dieser Befehl abschlossen und das nächste Kommando kann kommen:

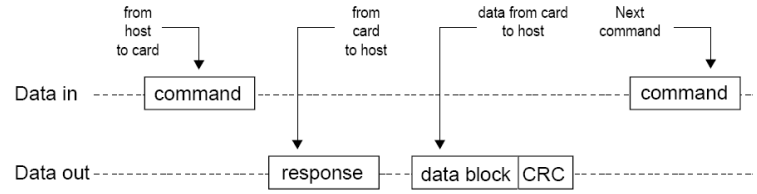


Abbildung 12 – MMC, Lesebefehl, aus [MMC]

Auf Bit-Ebene sieht die Kommunikation dann so aus:

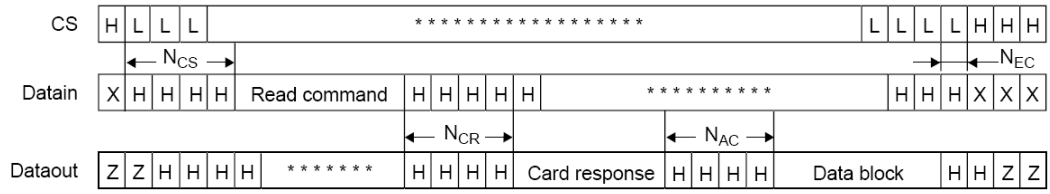


Abbildung 13 – MMC, Lesebefehl, Details, aus [MMC]

EIN SCHREIBBEFEHL:

Beim Schreiben von Daten ist, wie sonst auch, erst der Befehl zu senden, danach die Response-Antwort abzuwarten. Dann müssen alle Daten übertragen werden, die dann von der Karte abschließend noch bestätigt werden:

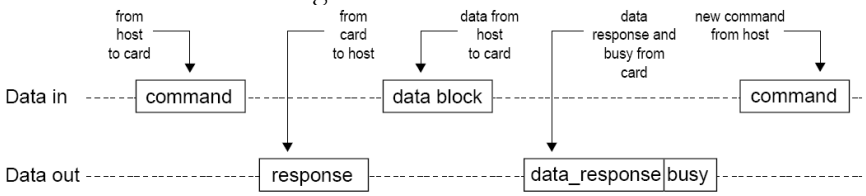


Abbildung 14 – MMC, Schreibbefehl, aus [MMC]

Auch hier wieder auf Bit-Ebene:

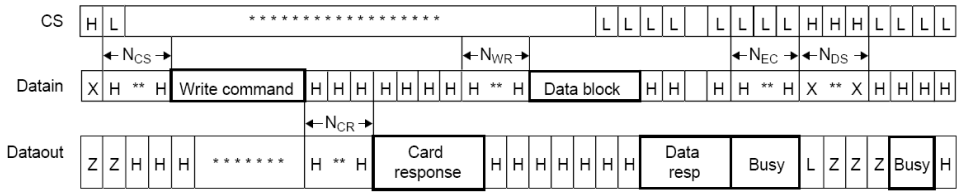


Abbildung 15 – MMC, Schreibbefehl, Details, aus [MMC]

2.4. LCD – HD61830 CONTROLLER

2.4.1. SPEZIFIKATIONEN

Die Wahl des LCD-Displays fiel in diesem Projekt auf das LMG6381 von Hitachi[11] [LCD]. Es ist ein graphisches Display mit 256 x 64 Pixeln, das auch im Textmodus betrieben werden kann, mit blauer Schrift auf grauem Hintergrund:

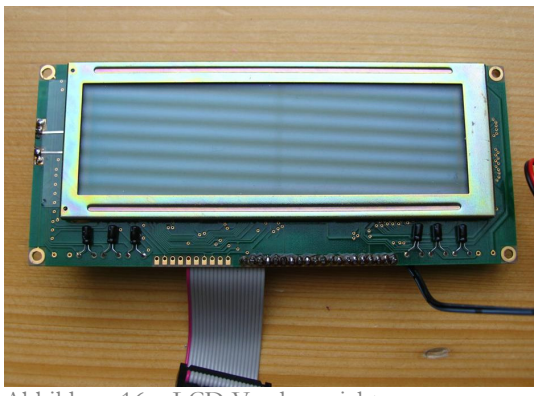


Abbildung 16 – LCD Vorderansicht

Das LCD-Display LMG6381 von vorne. Links sind die Anschlüsse für die Hintergrundbeleuchtung und unten die restlichen Leitungen.

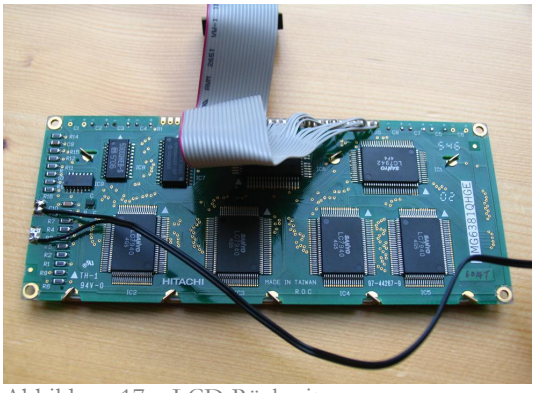


Abbildung 17 – LCD Rückseite

Von hinten sieht man den Controller und den Speicher.

[11] Homepage: www.hitachi.com

Das Display benötigt eine Logikspannung von 5V und eine LCD-Betriebsspannung von etwa -13V und zusätzlich noch einen Spannungsseingang mit einem Wert zwischen -13V bis +5V zum Einstellen des Kontrastes. Zur Hintergrundbeleuchtung besitzt es eine EL-Folie, die mit etwa 100V Wechselspannung bei 400Hz angeregt werden kann.

Für den Datenaustausch ist auf der Displayplatine der Standard-Controller HD61830 [LCDCTL] verbaut. Dieser verwendet zur Kommunikation folgende Signale:

Signal	Name	LOW (0)	HIGH (1)
/RESET	Reset	active	Shut down
/CS	Chip Select	selected	unselected
E	Enable	disabled	enabled
RS	Register Select	data	command
R/W	Read / Write	write	read
D0	Data Bit 0		
…	…		
D7	Data Bit 7		

Tabelle 8 – LCD-Controller Signale

2.4.2. KOMMUNIKATIONSPROTOKOLL

Die für uns relevanten Hauptdaten des Controllers sind [LCDCTL]:

- Graphikmodus: 512k Pixel
- Textmodus: 4096 Zeichen
- Zeichensatz: 160 5x7 Zeichen, 32 5x11 Zeichen, zusammen 192 Zeichen
- 8-Bit Interface
- Verschiedenste Funktionen: Scrollen, Cursor an/aus/blinken, Zeichen blinken, …

Um das Display zu aktivieren, um es also ansprechen zu können, muß die `/RESET`-Leitung auf `LOW` gesetzt werden und dort für die gesamte Betriebszeit gehalten werden.

Mit dem `R/W`-Signal kann ausgewählt werden, ob bei `LOW` (`write`) mit den Datenbytes `D0…D7` in das Display geschrieben, oder bei `HIGH` (`read`) entsprechend aus dem Display gelesen werden soll.

Die Ansteuerung des LCD-Displays läuft immer so ab: Ein Befehl besteht aus zwei Bytes, sprich zwei Teilen: Der erste Teil ist der Befehl, der zweite der zu setzende Wert. Das jeweilige Byte wird mit einem kurzen Strobe von Enable übernommen, das bedeutet, man setzt `E` auf `HIGH` und anschließend wieder auf `LOW`. Dabei ist die maximale Taktrate von 1MHz zu beachten.

Zuerst wird der Controller aktiviert, indem `/CS` auf `LOW` gelegt wird. Danach legt man das Befehlsregister als Ziel fest, indem `RS` auf `HIGH` (`command`) gesetzt wird und stellt dann auf Schreiben (`R/W` auf `LOW` (`write`)). Jetzt das Befehlsbyte auf der Datenleitung ausgeben und mit einem Enablestrobe übernehmen. Danach das Zielregister auf Daten umstellen (`RS` auf `LOW` (`data`)), die Daten anlegen und erneut mit Hilfe von `E` übertragen. Zuletzt nur noch den Controller wieder abselektieren (`/CS` auf `HIGH`).

Vor jedem Befehl muß abgeprüft werden, ob der Controller auch bereit ist. Das geschieht, indem man das Befehlsregister ausliest (sowohl `RS`, als auch `R/W`, also auch `E` auf `HIGH` (`command`), (`read`), (`enabled`)). Danach liegt auf Bit 7 das Busy-Signal an. `LOW` bedeutet, das Display ist bereit und `HIGH` bedeutet, daß es noch beschäftigt ist. Sobald es fertig ist setzt es die Leitung auf `LOW`, somit kann im Busy-Zustand auch aktiv auf die Fertigstellung gewartet werden.

Der Ablauf eines kompletten Befehls sieht also so aus:

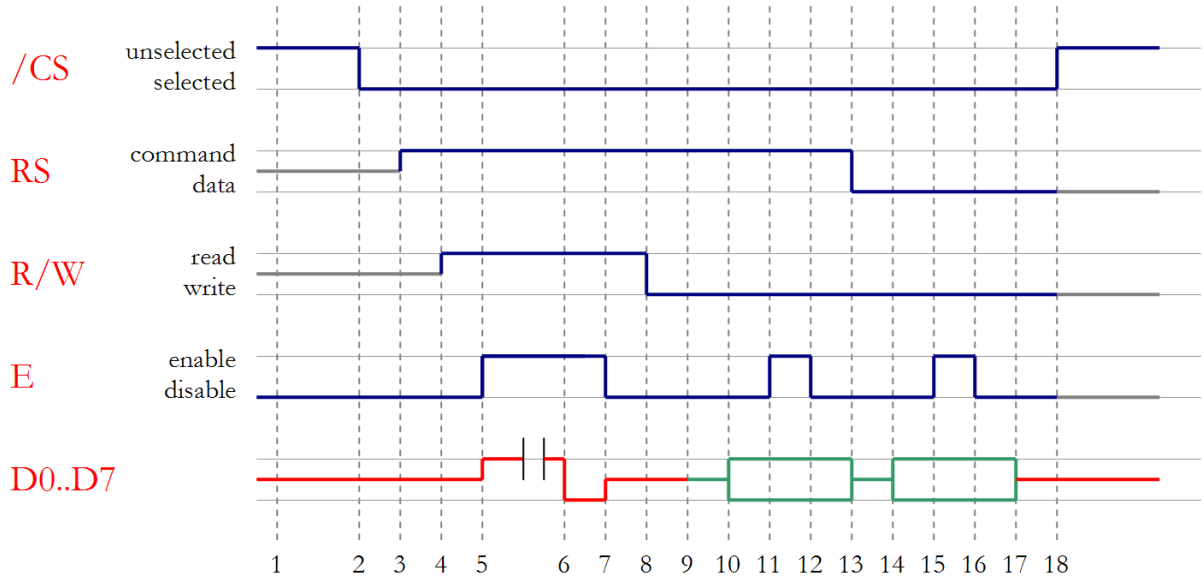


Abbildung 18 – LCD-Timing Diagramm

Farbe	Bedeutung (Aus Sicht des Steuergerätes, also AVR)
Blau	Ausgang, gesetzt vom Steuergerät
Rot	Datenrichtung Eingang (Lesen von LCD)
Grün	Datenrichtung Ausgang (Schreiben in LCD)

Tabelle 9 – LCD-Timing Diagramm Legende

Zur Verständlichkeit das Ganze nochmals in Textform:

Schritt	Signal	Auf	Aktion	Beschreibung
1				Ausgangszustand, unselected, disabled, Datenrichtung Eingang, Rest egal
2	/CS	LOW	selected	LCD selektieren
3	RS	HIGH	command	Befehlsregister auswählen
4	R/W	HIGH	read	Befehlsregister auslesen
5	E	HIGH	enable	Busy-Signal auslesen
6	D7	LOW	card ready	Warten bis Karte bereit
7	E	LOW	disable	Auslesen beenden
8	R/W	LOW	write	Schreiben in Befehlsregister
9	D0..7	OUT	set data direction output	Datenrichtung auf Ausgang setzen
10	D0..7	INSTR	set instruction	Befehl auf Datenleitung setzten
11	E	HIGH	enable	Befehl schreiben
12	E	LOW	disable	Befehl schreiben beenden
13	RS	LOW	data	Datenregister auswählen
14	D0..7	DATA	set data	Datenwert auf Datenleitung schreiben
15	E	HIGH	enable	Daten schreiben
16	E	LOW	disable	Daten schreiben beenden
17	D0..7	IN	set data direction input	Datenrichtung wieder auf Eingang
18	/CS	HIGH	unselect	LCD wieder abselektieren

Tabelle 10 – LCD-Timing Ablauf

2.4.3. BEFEHLE

Der Controller beherrscht insgesamt 13 Befehle. Die genaue Beschreibung würde hier zu weit gehen. Bei Interesse sei hier auf das Datenblatt [LCDCTL] verwiesen.

Befehl	Name	Beschreibung	Unser Wert
0x00	**Mode Control**	Setzt den allgemeinen Displaystatus, unter anderem Display An/Aus, Cursor An/Aus/Blinken, Graphik-/Textmodus und Internes-/Externes-Zeichen-ROM.	Textmodus, Internes ROM, Rest variabel
0x01	**Set Character Pitch**	Setzt die horizontalen und vertikalen Abmessungen eines Zeichens im Textmodus in gewissen Grenzen. Horizontal sind Werte von 1 bis 16 möglich, vertikal jedoch nur von 6 bis 8.	6x9
0x02	**Set Number of Characters**	Setzt die Anzahl der Buchstaben je Zeile auf gerade Werte zwischen 2 und 128.	42
0x03	**Set Number of Time Divisions**	Setzt die horizontale Auflösung des Displays, zwischen 1 und 128 Pixel.	64
0x04	**Set Cursor Position**	Legt die Zeile, in der der Cursor gesetzt wird, fest: 1 ist ganz oben, maximal kann die Zeichenhöhe angegeben werden.	9
0x08	**Set Display Start Low**	Setzt das untere Byte der Startadresse des Displays. Das RAM ist 4096 Bytes lang. Das ausgewählte Zeichen beginnt dann links oben im Display. Wenn man zeilenweise diese Adresse verändert, dann sieht das aus wie Scrollen.	variabel, meist 0
0x09	**Set Display Start High**	Setzt das obere Byte der Startadresse. Falls die Adresse neu gesetzt werden soll, muß zuerst das `LOBYTE` und danach das `HIBYTE` übertragen werden.	variabel, meist 0
0x0a	**Set Cursor Address Low**	Gleicher Aufbau wie Startadresse des Displays, allerdings wird hier die Cursorposition gesetzt.	variabel
0x0b	**Set Cursor Address High**	Auch hier gilt: Zuerst das untere, dann das obere Byte setzen.	variabel
0x0c	**Write Display Data**	Setzt das übertragene Zeichen an die aktuelle Cursorposition und erhöht diese danach um ein Zeichen. Um mehrere Zeichen zu setzten, muß also erst die Startposition gesetzt werden; dann können beliebig viele weitere Zeichen folgen.	Zeichen
0x0d	**Read Display Data**	Gleicher Aufbau wie Schreiben, nur gibt der Controller hier eben das Zeichen unter dem Cursor aus.	Zeichen
0x0e	**Clear Bit**	Findet nur im Graphikmodus Verwendung und löscht das angegebene Bit aus dem aktuell selektierten Byte.	0..7
0x0f	**Set Bit**	Auch dieser Befehl funktioniert nur im Graphikmodus und setzt das entsprechende Bit.	0..7

Tabelle 11 – LCD Befehlsübersicht

3. HARDWARE

3.1. AUFBAU

Um übersichtlicher und auch in Teilen leichter wiederverwendbar zu sein, besteht das Gerät nicht aus einer einzigen Platine, sondern aus mehreren Modulen, die jeweils über ein Datenkabel mit der zentralen Steuereinheit verbunden sind.

Zumindest bei dem Prototyp werden die einzelnen Module jeweils auf einer eigenen Platine aufgebaut und untereinander mit Flachbandkabeln verbunden. Im fertigen Gerät, welches dann in Serie geht, werden natürlich die Schaltpläne zu einer großen, einzelnen Platine zusammengefaßt, die dann auf Größe optimiert werden.

Um diesem Kapitel schon einmal vorweg zu greifen und eine Übersicht über den Sachverhalt zu schaffen: Hier die komplette Abbildung der einzelnen Module und deren Zusammenhänge untereinander:

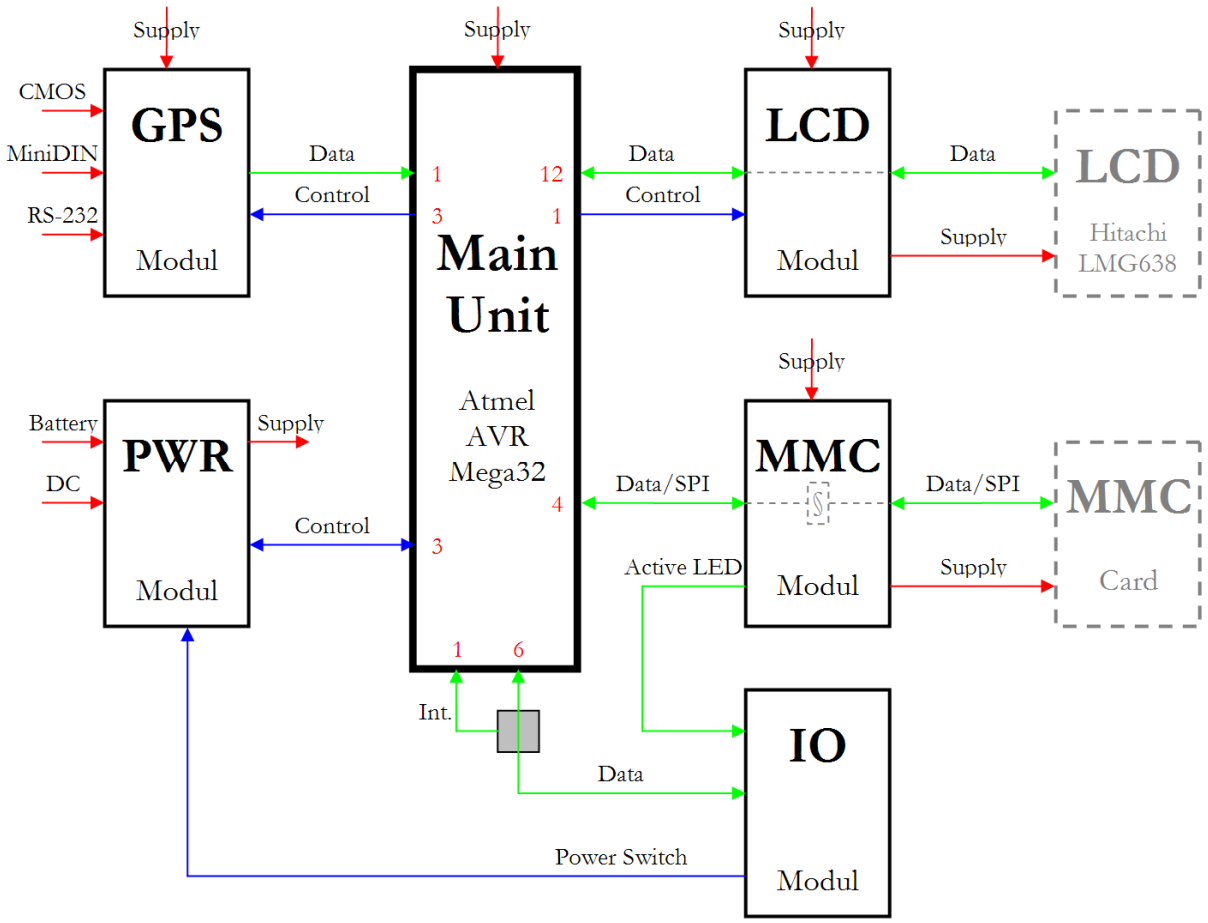


Abbildung 19 – Hardwareaufbau, komplett

Das LCD-Display und die MMC-Karten wurden im Grundlagenkapitel schon ausführlich besprochen. Die folgenden Abschnitte dieses Kapitels erläutern die restlichen fünf Module und die Main Unit genauer.

3.2. ATMEL AVR STK500

Das AVR STK500 von Atmel [STK500] ist ein Starter-Kit und Entwicklungssystem für AVR Mikroprozessoren:

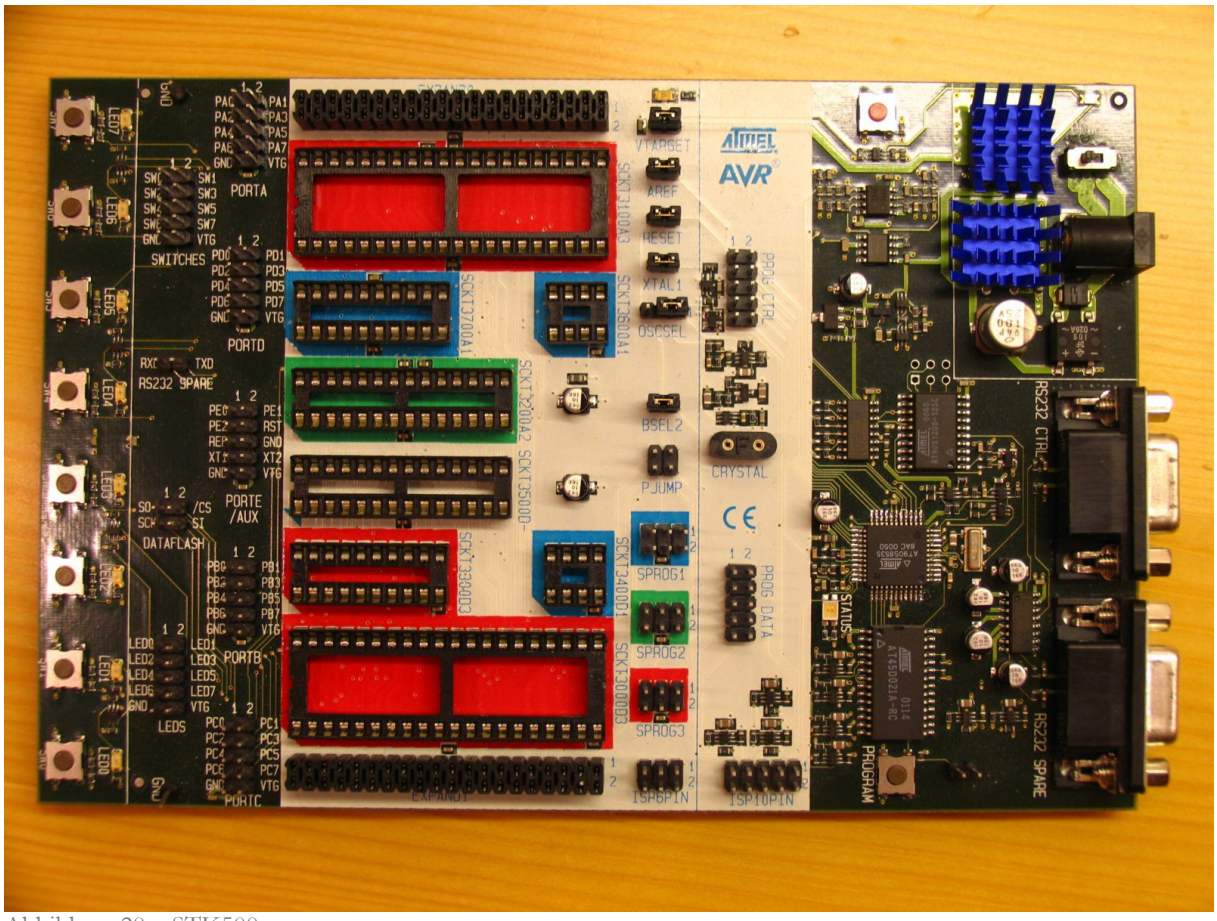


Abbildung 20 – STK500

Es besteht unter anderem aus folgenden Komponenten:

- RS-232 Interface zur Programmierung und Kommunikation mit dem PC
- Variablem Spannungsregler
- Sockel für 8-, 20-, 28- und 40-Pin AVR Controller
- ISP (Serial In System Programming) und ISP für externe Geräte
- High Voltage Programming (Seriell und Parallel)
- 8 Taster zur Eingabe
- 8 LEDs zur Ausgabe
- Alle Ports der Mikrocontroller auch als Steckverbinder
- Allgemein nutzbare RS-232 Schnittstelle
- 2 MBit DataFlash[12]

[12] DataFlash ist ein nichtflüchtiger Speicher von Atmel. Der Zugriff geschieht über SPI und ähnelt MMC-Karten.

Diese Merkmale machen das STK500 optimal für den schnellen Testaufbau der einzelnen Module.

3.2.1. TESTAUFBAU

Dieses Kapitel soll einen kurzen Eindruck über die Möglichkeiten des STK500 geben, indem einige Testaufbauten der Module gezeigt werden:

Sie werden in der Reihenfolge, wie sie gebaut wurden, gezeigt:

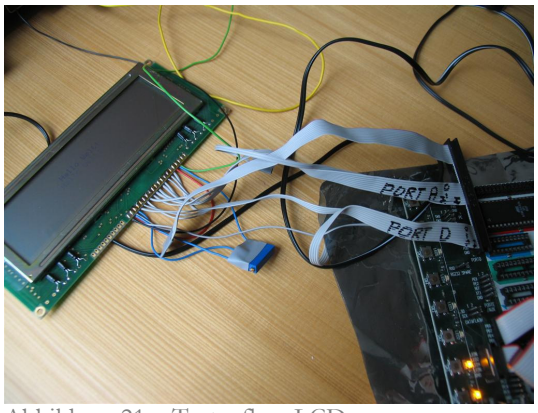


Abbildung 21 – Testaufbau LCD

Das LCD Display wird passend an ein Flachbandkabel (hier ein altes SCSI-Kabel) gelötet. Diese kann man dank Steckverbinder pro Port direkt aufstecken.

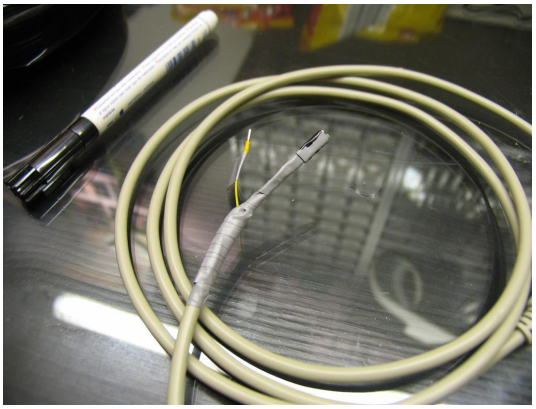


Abbildung 22 – Testaufbau GPS Kabel

Um ein Anschlußkabel für den Empfänger herzustellen, wurde ein PS2-Verlängerungskabel geteilt und die nötigen drei Pins (VDD, GND, CMOS-RX) mit passenden Steckern versehen.

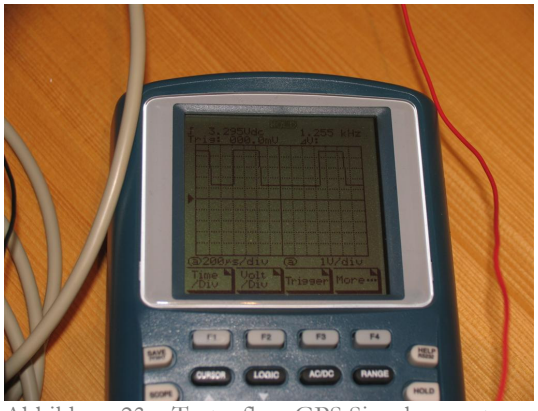


Abbildung 23 – Testaufbau GPS Signalauswertung

Das Oszilloskop zeigt das CMOS-Signal noch mal in aller Deutlichkeit an.

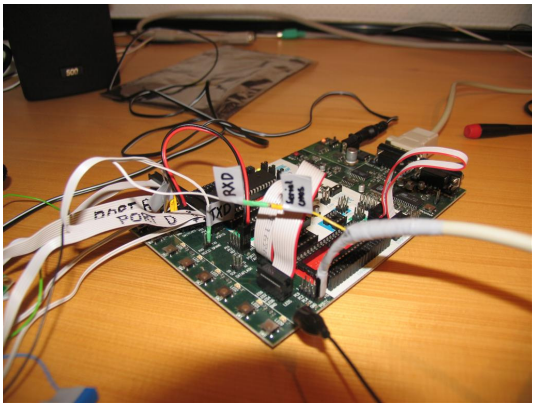


Abbildung 24 – Testaufbau GPS

Das fertige Adapterkabel für den GPS-Empfänger muß an das Flachbandkabel vom LCD angeschlossen werden, da sich der RX-Pin in einem vom LCD-Display mitgenutztem Port befindet.

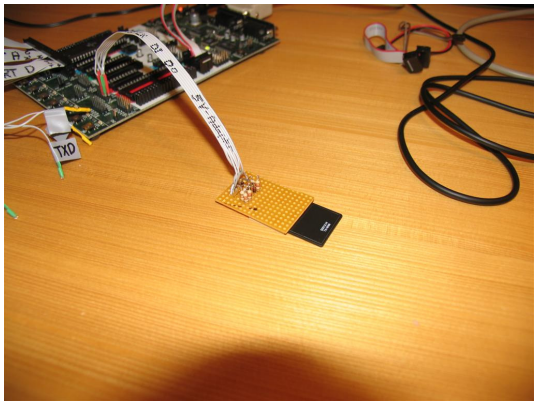


Abbildung 25 – Testaufbau MMC

Der Levelkonverter für die MMC-Karte wurde auf einer Extra-Platine aufgebaut und mit einem Flachbandkabel mit den richtigen Pins am STK500 verbunden.

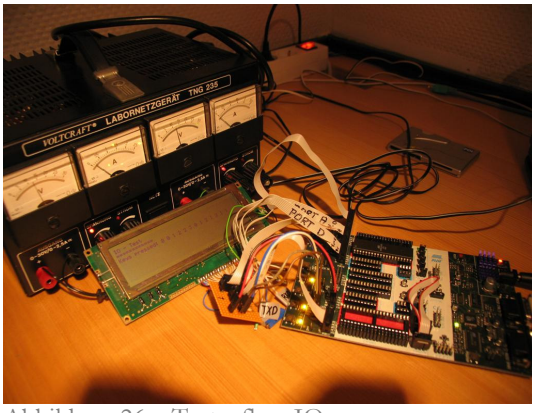


Abbildung 26 – Testaufbau IO

Da im Testaufbau die Ports knapp waren, teilt die IO-Platine einen 8-Bit-Port in zwei je 4-Bit-Ports auf, die dann mit den LED- und den Taster-Steckern auf dem STK500 verbunden werden können.

3.3. Allgemeine, verwendete Hardware

Folgender Abschnitt beschreibt kurz die verwendeten Transistorarten im Projekt.

3.3.1. BC548B

Der BC548 ist unser Standard-NPN-Transistor. Er besitzt das kleine, typische Transistorgehäuse TO-92. Seine zulässige Spannung U_{CEO} beträgt 30V und der dauerhafte Kollektor-Strom I_C 100mA.

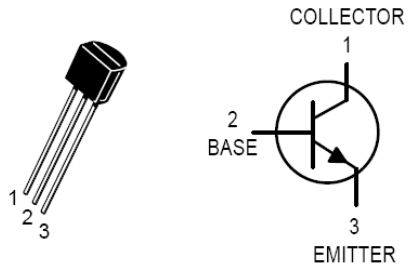


Abbildung 27 – Pinbelegung BC548B, aus [BC548]

Die minimal zugesicherte Gleichspannungsverstärkung h_{FE} beträgt beim BC548B laut Datenblatt 200. Um einen Eindruck für die reale Verstärkung dieser Serie zu bekommen, wurde eine kleine Testreihe mit folgendem, zusammengefaßten Ergebnis, angefertigt:

I_B [mA]	I_C [mA]	Gain
0.044	27	614
0.080	44	550
0.137	70	510
0.200	90	450
0.241	100	415

Tabelle 12 – Verstärkung Transistor BC548B, aus [BC548]

In den nachfolgenden Berechnungen wird zur Sicherheit nur von den garantierten 200 ausgegangen.

Datenblatt siehe [BC548].

3.3.2. BC558B

Der BC558 ist der komplementäre PNP-Transistor zu unserem BC548B. Die technischen Daten sind dieselben, wie bei der NPN-Variante. Einzig die zugesicherte Verstärkung liegt bei nur 180.

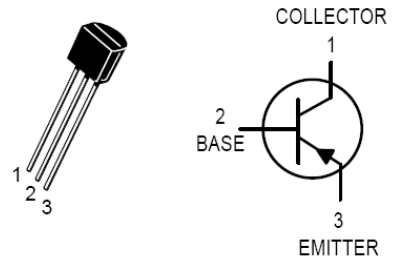


Abbildung 28 – Pinbelegung BC558B, aus [BC558]

Datenblatt siehe [BC558].

3.3.3. BD439

Der BD439 ist unser NPN-Transistor, wenn es um größere Leistungen geht. Er wird unter anderem zum Schalten der Hintergrundbeleuchtung oder des GPS-Empfängers genutzt.

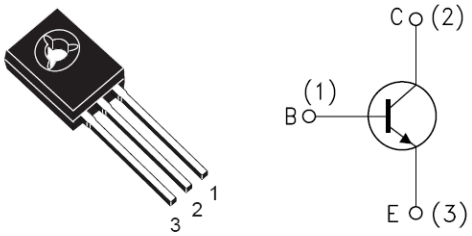


Abbildung 29 – Pinbelegung BD439, aus [BD439]

Er sitzt in einem etwas größeren SOT-32 Gehäuse und besitzt eine maximale Kollektor-Emitter-Spannung U_{CEO} von 60V. Der höchstzulässige Basisstrom I_B darf 1A und der Kollektorstrom I_C

4A nicht überschreiten. Er besitzt ohne zusätzliche Kühlung eine Wärmeverlustleistung von 100°C pro Watt. Da aber in unserem Projekt maximal nur 200mA geschaltet werden, kommt er in noch keinen kritischen Bereich und kann ohne Zusatzkühler betrieben werden.

Die typische Gleichstromverstärkung h_{FE} liegt laut Datenblatt bei 140, die minimale aber nur bei 20 bei I_C = 10mA bis 40 bei I_C = 500mA. Somit wären unverhältnismäßig hohe Basisströme notwendig. Grund genug, um auch hier eine Versuchsreihe durchzuführen:

I_B [mA]	I_C [mA]	Gain
0.14	40	280
0.46	125	274
1.33	280	210
4.30	650	150

Tabelle 13 – Verstärkung Transistor BD439

Wie man sieht, liegen die ermittelten Werte deutlich über den minimal zugesicherten. Um die Stromaufnahme nicht unnötig zu vergrößern, wird in folgenden Berechnungen von einer minimalen Verstärkung von 100 ausgegangen.

Datenblatt siehe [BD439].

3.4. MODULE

3.4.1. PWR – POWER BOARD

3.4.1.1. AUFGABE

Die Hauptaufgabe des Power-Moduls ist die Stromversorgung des Gerätes mit konstanten 5V Gleichspannung mit mindestens 500mA. Als Stromquelle dient entweder eine Batterie oder eine externe DC-Quelle[13]. Das Modul soll einen großen Eingangsspannungsbereichbesitzen besitzen, mit einer möglichst niedrigen unteren Grenzspannung.

Die Batterie ist entweder eine 9V Blockbatterie, die zur Zeit allerdings nur mit weniger als 200mAh erhältlich sind, Oder eine Kombination von 6 x 1.2V-NiMH-Akkus im Verbund. Diese werden zur Zeit schon mit relativ hohen 2300mAh angeboten. Die minimale Betriebsspannung sollte somit möglichst weit unter 7V gehen, eventuell sogar auf nur 6V, damit eine Schlußentladespannung von je 1V pro Zelle erreicht werden kann.

Der Prozessor muß die Stromversorgung bei Bedarf vollständig trennen können, um das Gerät komplett auszuschalten. Die dann aufgenommene Leistung soll minimal sein und die Batterie nicht merklich belasten. Im ausgeschalteten Zustand muß die Stromversorgung wieder hergestellt werden können, nachdem der Power-Taster, betätigt wurde.

Der Zustand dieses Tasters ist zusätzlich als IO-Signal auszugeben um ihn auch als ganz normalen Taster, vorrangig als Powerbutton, einzusetzen zu können.

[13] Netzteil, Autobatterie, Motorradbatterie, Zusatzakku, …

Letztendlich soll noch eine Ausgabe bei zu niedriger Eingangsspannung ausgegeben werden um festzustellen, daß zum Beispiel die Batterien fast entladen sind und somit eine LoBat-Warnung ausgegeben werden muß.

3.4.1.2. PINBELEGUNG DES IPWR-CONNECTORS

Der IPWR-Connector ist 10-polig und es werden 7 Leitungen benutzt:

IPWR	Typ[14]	Signal	Beschreibung
3	S	PS1	Power Switch 1, out
4	S	PS2	Power Switch 2, in
6	I	/PWRBUT	Power Button pressed
7	I	/LOBAT	Low Battery
8	O	KEEP	Keep power supply up
9		VDD	
10		GND	

Tabelle 14 – Pinbelegung IPWR-Connector

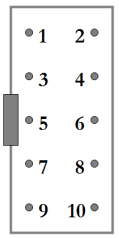


Abbildung 30 – 10-polig

Damit werden folgende Ressourcen am Mikrocontroller benötigt:

Typ	Anzahl
I	2
O	1
	3

Tabelle 15 – Ressourcenbelegung IPWR-Connector

3.4.1.3. PROBLEME UND DEREN LÖSUNGEN

BATTERIE UND EXTERNE DC-VERSORGUNG

Um ein Überladen beziehungsweise ein Laden der Batterien überhaupt zu verhindern, wird in die positive Eingangsleitung jeweils eine Standard-Diode *1N4001* geschaltet. Zusätzlich entsteht dadurch auch noch ein Verpolungsschutz. Einen Nachteil hat diese Lösung allerdings: Die minimale Eingangsspannung steigt durch die Diffusionsspannung[15] um etwa 0.7V.

KONSTANTE 5V GLEICHSPANNUNG

Es könnte ein normaler Spannungsregler, wie der *7805*, verwendet werden, der allerdings einige Nachteile besitzt: Zum einen „verheizt" er die überschüssige Spannung einfach, was vor allem für eine batteriebetriebene Applikation nicht sinnvoll ist. Zum anderen liegt die nötige Mindesteingangspannung laut Datenblatt relativ hoch bei größer oder gleich 7.0V. Auch bei einer manuellen Prüfung einzelner Exemplare konnten leider keine besseren Werte erreicht werden.

Deshalb fiel die Wahl auf die mittlerweile recht verbreiteten *Step-Down-Wandler*. Diese benutzen eine Spule als Energiespeicher. Der Hauptvorteil liegt zum einen an der niedrigen Eingangspannung, die in manchen Beschaltungsarten sogar niedriger als die Ausgangsspannung sein kann und zum anderen ist auch der Wirkungsgrad ziemlich hoch und nahezu konstant. Das bedeutet, wenn die Eingangsspannung steigt, dann sinkt die Stromaufnahme. Nur die Leistung bleibt konstant.

[14] Datenrichtung vom Controller: S: Special (connected somewhere else); I: Input; O: Output
[15] Spannungsabfall an einem PN- oder NP-Übergang

Bildlich erklärt: Wenn man einen theoretischen Step-Down-Wandler mit 100% Wirkungsgrad, bei einer Ausgangsspannung von 5V und einem Ausgangsstrom von 1A, annimmt, so werden bei 5V Eingangspannung 1A entnommen, bei 10V nur noch 0.5A und bei 20V sogar nur noch 0.25A.

Ein weiterer Vorteil ist, daß selbst bei höheren Strömen, bei welchen normale Spannungswandler schon gekühlt werden müssen, die Verlustleistung noch so gering ist, daß die Step-Down-Regler gar nicht gekühlt werden müssen.

Ein großer Nachteil ist allerdings auch nicht zu verschweigen: Es werden mehrere externe Komponenten benötigt, von denen die nötige Spule *L1* mit 300µH auch nicht gerade sehr klein ausfällt.

Für diesen Zweck wurde von *National Semiconductor* der *Step-Down Volatage Regulator* „LM2575-5.0“ ausgewählt.

Die Standardbeschaltung [LM2575] sieht folgendermaßen aus:

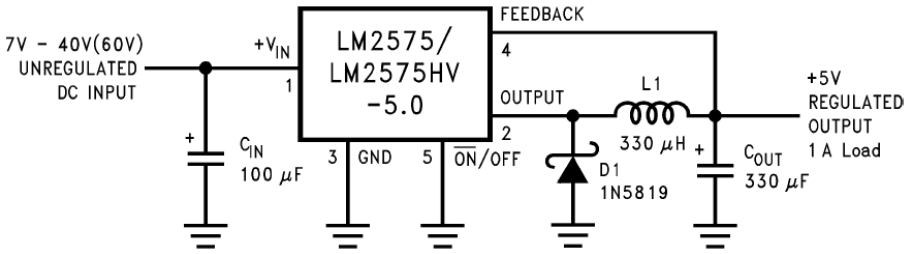


Abbildung 31 – Standardbeschaltung LM2575, aus [LM2575]

Wenn nicht die vollen 1A Ausgangsstrom genutzt werden sollen, kann die notwendige Eingangsspannung auch deutlich niedriger sein. Die im fertigen Modul gemessenen Werte gehen bei maximaler Belastung des Gerätes auf bis zu 5.5V herunter. Addiert man die Diffusionsspannung der Dioden am Eingang dazu, kommt man auf etwa 6.2V, was den geforderten 6V recht nahe kommt.

Um die negativen Spannungsspitzen der Spule nicht auf den Spannungsregler zu legen, müssen diese durch eine Diode eliminiert werden, da sonst der Regler in kürzester Zeit zerstört werden würde. Es ist äußert wichtig, daß die Diode eine schnelle Schottky Diode ist und sich physikalisch möglichst nahe am LM2575 befindet. Diese Diode *D5* besitzt eine niedrige Diffusionsspannung von nur etwa 0.2V und schaltet im Vergleich zu Gleichrichterdioden wesentlich schneller.

Zusätzlich besitzt der LM2575 einen Enable-Eingang, mit dessen Hilfe man den Wandler ein- und ausschalten kann.

Für eine genauere Beschreibung des Bauteils sei hier auf das Datenblatt [LM2575] verwiesen, das sich auch auf der Begleit-CD befindet.

Um die weiteren Schaltungsteile besser verstehen zu können oder sich einen Gesamteindruck von der Schaltung zu schaffen, sei hier noch mal auf den fertigen Schaltplan im nächsten Abschnitt hingewiesen.

D1 und *D2* übernehmen, wie schon oben erläutert, die Trennung von Batterie und externen DC-Eingang und die Verpolungssicherheit. Die ebenfalls oben schon beschriebene Beschaltung des LM2575 ist auch auf Anhieb im Schaltbild wieder zu erkennen.

KEEP + PWRSWITCH

Der Step-Down-Regler besitzt einen invertierten Enable-Eingang. Das bedeutet, er ist eingeschaltet, wenn der Eingang auf Masse liegt und ausgeschaltet, wenn er HIGH ist, was im Ruhezustand durch *R6* auch der Fall ist. Um das Gerät nun einzuschalten muß *T1* leiten, was dadurch zu erreichen ist, daß entweder *D3* oder *D4* auf HIGH gesetzt wird. Auf PS1, also die eine Seite des Power-Tasters, wird über einen Vorwiderstand die volle Eingangsspannung gelegt. Wird der Taster nun gedrückt, so kommt das Signal über PS2 und *D4* und schaltet damit den Spannungswandler ein.

Damit beim Loslassen des Tasters die Stromversorgung nicht wieder ausgeschaltet wird, muß der Mikrocontroller sofort beim Booten das KEEP-Signal setzen. Dadurch erhält sich der Prozessor selbst am Leben und durch die beiden Dioden stören sich auch bei weiterem Drücken der Powertaste das PS2- und das KEEP-Signal nicht. Um das Gerät auszuschalten reicht es nun aus das KEEP-Signal wieder auf 0 zu setzen. Allerdings darf die Powertaste zu dieser Zeit nicht mehr gedrückt sein, was aber in einer einfachen Schleife abzufangen ist.

PWRBUT

Um den Power-Taster auch als normalen Eingang zu nutzen und seinen Zustand abfragen zu können, muß das Signal in eine 0V/5V Logik umgesetzt werden. Es wird einfach der PS2 Eingang benutzt, der allerdings die volle Betriebsspannung und nicht nur die maximalen 5V aufweist. Deshalb wird ein Spannungsteiler, *R10/R9,* eingebaut. Da bei dem GPS-Logger gedrückte Taster generell auf Masse schalten sollen, invertiert der Transistor *T2* das Signal. Das endgültige Signal wird letztendlich auf /PWRBUT ausgegeben.

LOBAT

Die letzte offene Anforderung ist eine Warnung bei zu niedriger Eingangsspannung. Hierzu findet eine einfache Komparatorschaltung Verwendung. Der invertierende Eingang des Operationsverstärkers *IC2a* wird, über den Spindeltrimmer *R3,* auf einen festgelegten Wert zwischen 0V und 5V eingestellt. Mit diesem ist die genaue Schwelle der LOBAT-Warnung festzulegen. Der nichtinvertierende Eingang ist über einen Spannungsteiler direkt mit der Eingangsspannung verbunden. Der Ausgang des OP-Verstärkers entspricht somit schon dem /LOBAT Signal.

3.4.1.4. SCHALTPLAN

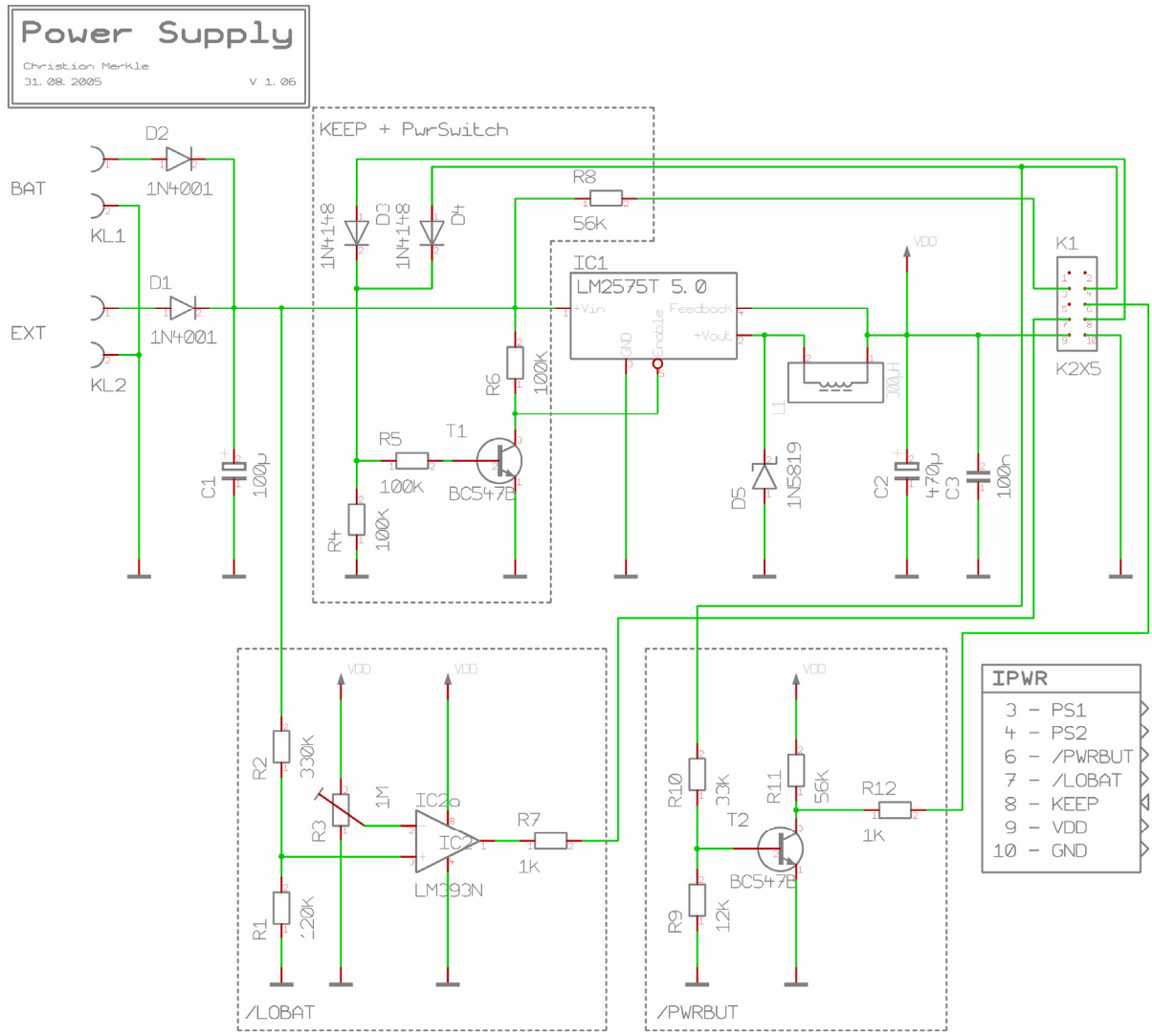


Abbildung 32 – Schaltplan PWR

3.4.1.5. DIMENSIONIERUNG DER BAUTEILE

Die Bauteile für den Step-Down-Regler werden direkt im Datenblatt vorgegeben, bzw. dort berechnet.

Die anderen Werte der Widerstände sind eigentlich, bis auf wenige Ausnahmen, alle unkritisch, da diese nur zu Schaltzwecken dienen. Sie sollten möglichst groß sein, damit wenig Strom fließt, aber dennoch eine sichere Schaltung gewährleisten.

Einzig und allein die Spannungsteiler *R2/R1* und *R10/R9* haben ein Verhältnis von 0.27. Um dieses Verhältnis zu erreichen ist eine Kombination von 330kΩ und 120kΩ zu verwenden. Damit ist eine maximale Eingangsspannung von 5V / 0.27 = 18V erlaubt – mit leicht zulässigem Spannungsüberschuß auch ohne Risiko bis 19V. Dann liegt eine maximale Spannung von 19V / 450kΩ * 120kΩ = 5.06V am Komparator und am Transistor *T2* an.

Da der Spannungsteiler für den /PWRBUT nur bei gedrücktem Powertaster, also sehr selten, unter Spannung steht, ist hier die Stromaufnahme weniger relevant. Ganz im Gegensatz zum Spannungsteiler beim Komparator für /LOBAT, der ständig unter Spannung steht – als einziges

Bauteil selbst auch bei ausgeschaltetem Gerät. Hier sind die Werte möglichst hoch zu wählen, so daß der Operationsverstärker noch zuverlässig funktioniert, die Batterien aber nur so wenig wie möglich belast werden.

3.4.1.6. STROMAUFNAHME

Die maximale Stromaufnahme beträgt bei 9V Eingangsspannung:

KEEP + PwrSwitch:
R4, *R5*, *R6*:

$$3 \cdot \left(\frac{9V}{100k\Omega} \right) \longrightarrow 0.27mA$$

/LOBAT:
(*R2*+*R1*); *R3:*

$$\frac{9V}{450k\Omega} + \frac{5V}{1M\Omega} \longrightarrow 0.25mA$$

/PWRBUT:
(*R10*+*R9*); (*R10*+*T2*); (*R11*):

$$\frac{9V}{45k\Omega} + \frac{9V}{33k\Omega} + \frac{5V}{56k\Omega} \longrightarrow 0.56mA$$

Stromaufnahme Operationsverstärker *IC2*:
Maximal 1mA

Stromaufnahme Step-Down-Regler *IC1*:
Maximal 10mA

Gesamtaufnahme: < 12mA

Der Wirkungsgrad des Spannungswandlers liegt, je nach Eingangsspannung und entnommener Leistung, bei 0.70 bis 0.83.

GEMESSENE WERTE

Stromentnahme Ausgang [mA]	Stromentnahme bei U_{IN}=7.2V [mA]	Stromentnahme bei U_{IN}=9.0V [mA]	Stromentnahme bei U_{IN}=12V [mA]	Stromentnahme bei U_{IN}=16V [mA]
Shutdown, Aus	0.076	0.076	0.076	0.076
0.0	8.9	7.4	6.6	6.3
5.0	17	13	10	9.0
50	64	51	37	28

Tabelle 16 – PWR-Modul, Gemessene Stromwerte

Graphisch dargestellt erkennt man gut den Zusammenhang:

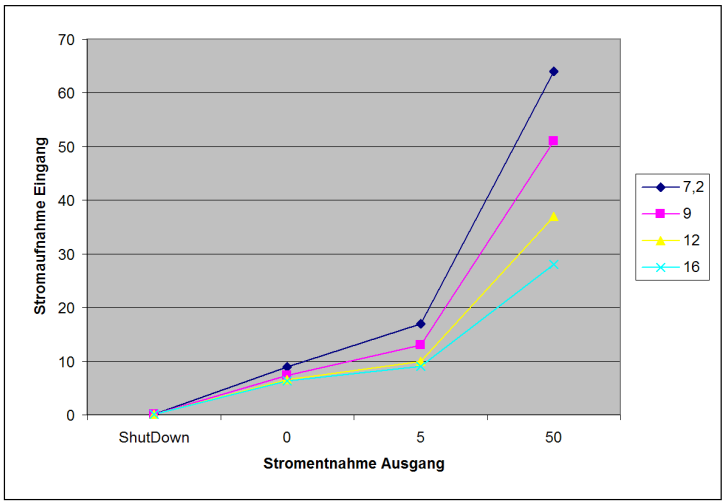


Abbildung 33 – PWR-Modul, Stromentnahme graphisch

Damit ergeben sich folgende Wirkungsgrade bei 50mA Stromentnahme am Ausgang:

Berechnung	U_{IN}=7.2V	U_{IN}=9.0V	U_{IN}=12V	U_{IN}=16V
Stromaufnahme bei 50mA	64mA	51mA	37mA	28mA
Abzug Leerlaufstrom	55mA	44mA	30mA	22mA
Faktor U_{IN} auf U_{OUT}	1.44	1.80	2.40	3.2
Normalisierung	79mA	79mA	72mA	70mA
Wirkungsgrad η	0.63	0.63	0.69	0.71

FAZIT

Die Standby-Stromaufnahme von maximal 9mA liegt den berechneten 12mA sehr nahe. Der Wirkungsgrad ist leider jedoch deutlich schlechter ausgefallen als erwartet.

3.4.1.7. FOTOS VOM AUFBAU UND DER FERTIGEN PLATINE

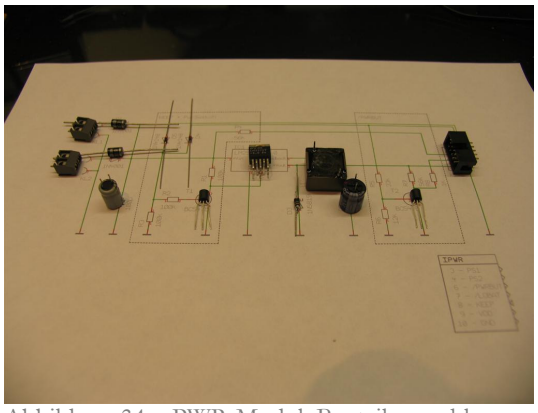


Abbildung 34 – PWR-Modul, Bauteilauswahl

Alle notwendigen Bauteile, bis auf die Widerstände, sind bereitgelegt.

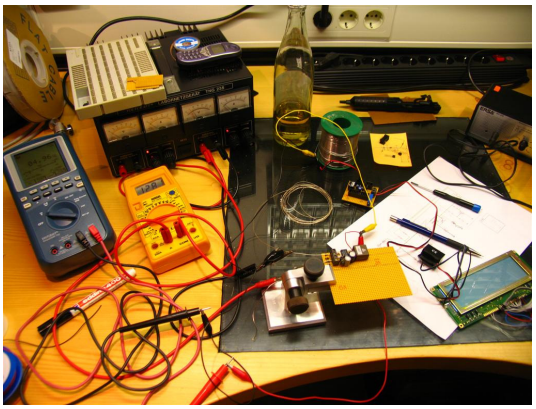


Abbildung 35 – PWR-Modul, Testläufe

Nach dem ersten Teilaufbau wird ein Test gefahren, ob sich die Schaltung auch wie erwünscht verhält und ob die Powerbutton- und KEEP-Kombination funktioniert.

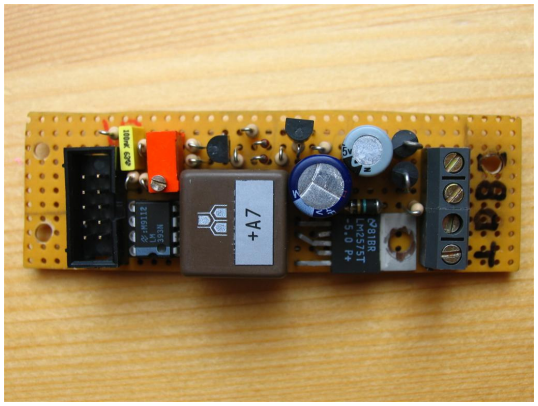


Abbildung 36 – PWR-Modul, Vorderansicht

Ansicht von oben: Von links nach rechts: 10-poliger IPWR-Connector, LOBAT-Schaltung, KEEP-Schaltung, PWRBUT-Schaltung, Spannungswandler, Eingangsbuchsen

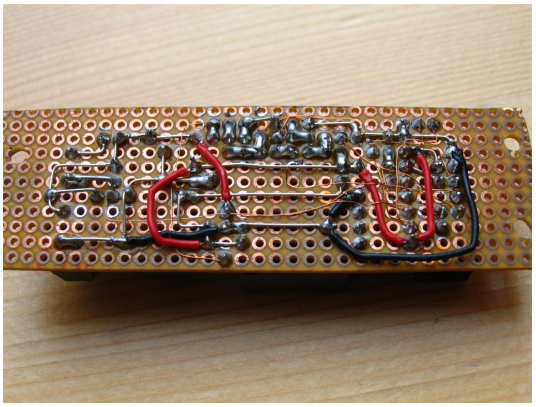


Abbildung 37 – PWR-Modul, Rückseite

Ansicht von unten: Es wurde darauf geachtet, daß alle stromführenden Leitungen entweder mit Silberdraht oder mit isoliertem Draht gelötet wurden. Nur die Datenleitungen sind aus dünnem Draht.

3.4.2. LCD – LCD Board

3.4.2.1. Aufgabe

Das LCD-Modul soll den Mikrocontroller möglichst einfach mit dem Controller des Displays verbinden und alle notwendigen Beschaltungen und Rahmenbedingungen sicherstellen.

Alle Daten- und Steuerleitungen sollen direkt durchgeleitet werden.

Das LCD-Display benötigt mehrere Betriebsspannungen: +5V, 0V, -13V und eine variable Spannung von -13V bis +5V zum Einstellen des Kontrasts. Diese müssen aus der einzigen 5V Versorgungsspannung erzeugt werden.

Zudem soll ein EL-Inverter über eine Steuerleitung geschaltet werden können, der die EL-Folie[16] versorgt und somit die Hintergrundbeleuchtung ein- und ausschaltet.

3.4.2.2. Pinbelegung des ILCD-Connectors

Der ILCD-Connector ist 16-polig und es werden alle 16 Leitungen benutzt:

ILCD	Typ[17]	Signal	Beschreibung
1	I/O	D0	Data Bit 0
…	I/O	…	…
8	I/O	D7	Data Bit 7
9	O	RS	Register Select
10	O	RW	Read / Write
11	O	E	Enable
12	O	/CS	Chip Select
13	O	/RST	Reset
14	O	/BG	Enable LCD Background
15		VDD	
16		GND	

Tabelle 17 – Pinbelegung ILCD-Connector

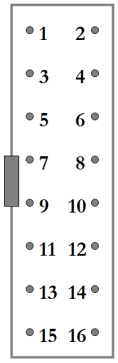


Abbildung 38 – 16-polig

Damit werden folgende Ressourcen am Mikrocontroller benötigt:

Typ	Anzahl
I/O	8
O	6
	14

Tabelle 18 – Ressourcenbelegung ILCD-Connector

[16] Es existieren mehrere Möglichkeiten ein LCD-Display zu beleuchten: Bei Displays in dieser Größe ist es üblich LEDs als Hintergrundbeleuchtung einzusetzen oder sogenannte EL-Folien, die durch eine Hochspannung von etwa 100V RMC Wechselspannung mit ca. 400 Hz angeregt werden. Genauere Angaben sind aus den jeweiligen Datenblättern zu entnehmen.

[17] Datenrichtung vom Controller: I/O: Input and Output; O: Output

3.4.2.3. Pinbelegung des LCD-Connectors

Der LCD-Connector wird im Gegensatz zum ILCD direkt mit dem Display verbunden. Er ist 20-polig und alle Leitungen werden richtig numeriert mit den Anschlüssen A1 – A20 des LCDs verbunden:

LCD Conn.	LCD	Typ[18]	Name	Beschreibung
1	A1	Sup	GND	Supply Ground
2	A2	Sup	+5V	Supply Logic +5V
3	A3	Sup	VO	Supply LCD Drive (var.)
4	A4	O	RS	Register Select
5	A5	O	RW	Read / Write
6	A6	O	E	Enable
7	A7	I/O	D0	Data Bit 0
…	…	I/O	…	…
14	A14	I/O	D7	Data Bit 7
15	A15	O	/CS	Chip Select
16	A16	O	/RST	Reset
17	A17	Sup	-12V	Supply LCD (-12V)
18	A18		-nc-	not connected
19	A19		-nc-	not connected
20	A20		-nc-	not connected

Tabelle 19 – Pinbelegung LCD-Connector

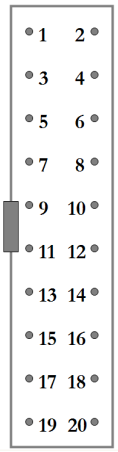


Abbildung 39 – 20-polig

3.4.2.4. Probleme und deren Lösungen

Notwendige Spannungen

Die erste offensichtliche Hürde ist das Erzeugen der notwendigen Spannungen. Dazu wird ein DC/DC-Wandler für 5V/12V eingesetzt. Dieser arbeitet nach dem Prinzip unseres Step-Down-Reglers, jetzt jedoch als Step-Up-Regler und erzeugt aus einer 5V Eingangsspannung eine 12V Ausgangsspannung. Da beide Ausgänge potentialfrei sind, kann der positive Ausgang auf Masse gelegt werden und damit eine negative Spannung von -12V erzeugt werden. Die variable Spannung für den Kontrast stellt ein Spindeltrimmer zur Verfügung, der auch von außen erreichbar ist, damit der Benutzer den Kontrast einstellen kann.

Um etwas vorwegzunehmen: Wenn die Hintergrundbeleuchtung aktiv ist, ist der DC/DC-Wandler nicht schnell genug, sich den Stromschwankungen anzupassen und somit entsteht eine recht wellige Spannung. Wenn diese direkt auf das Display geht, ändert sich der Kontrast mit der Frequenz ständig und das Bild flimmert. Um dem entgegen zu wirken muß die Spannung, zumindest die für die Kontrastversorgung, geglättet werden. Da das LCD auf der negativen Eingangseite nur sehr kleine Ströme benötigt, kann die Glättung einfach durch einen Widerstand und einen Kondensator geschehen.

Hintergrundbeleuchtung

Das LCD-Display besitzt zur Hintergrundbeleuchtung eine EL-Folie, die mit Hochspannung angeregt wird. Laut Spezifikation sollten es etwa 100V RMS bei 400Hz Wechselspannung sein. Dafür eignen sich sogenannte EL-Inverter. Diese wandeln eine bestimmte Eingangsspannung – meist 12V – in die benötigte Wechselspannung um.

[18] Datenrichtung vom Controller bzw. vom Modul: Sup: Supply; I/O: Input and Output; O: Output

SCHALTEN DER HINTERGRUNDBELEUCHTUNG

Eine weitere trickreiche Aufgabe ist es, die Hintergrundbeleuchtung zu schalten. Der EL-Inverter benötigt eine Eingangsspannung von 12V. Da wir aber nur eine Spannung von -12V zur Verfügung haben, muß diese umgepolt werden, sprich: Aus 0V werden +12V und aus -12V werden 0V.

Das Problem liegt jetzt aber darin, daß das negative -12V Signal durch eine positive Spannung von LOW (0V) und HIGH (+5V) geschaltet werden muß – aus Sicht des EL-Inverters also durch LOW (+12V) und HIGH (+17V). Der Schaltpunkt sollte etwa in der Mitte, also bei 14.5V liegen. Dazu wird ein PNP-Transistor auf die +12V gelegt. Dieser würde nun bei Spannungen größer als 11.3V (12V-0.7V) sperren und bei Werten kleiner als 11.3V leiten. Um diesen Punkt auf die geforderten 14.5V zu erhöhen, verwendet man davor einen zusätzlichen Spannungsteiler mit dem Verhältnis von 0.78.

Um nun die große Leistung des EL-Inverters schalten zu können, muß noch ein zusätzlicher NPN-Transistor verwendet werden.

3.4.2.5. SCHALTPLAN

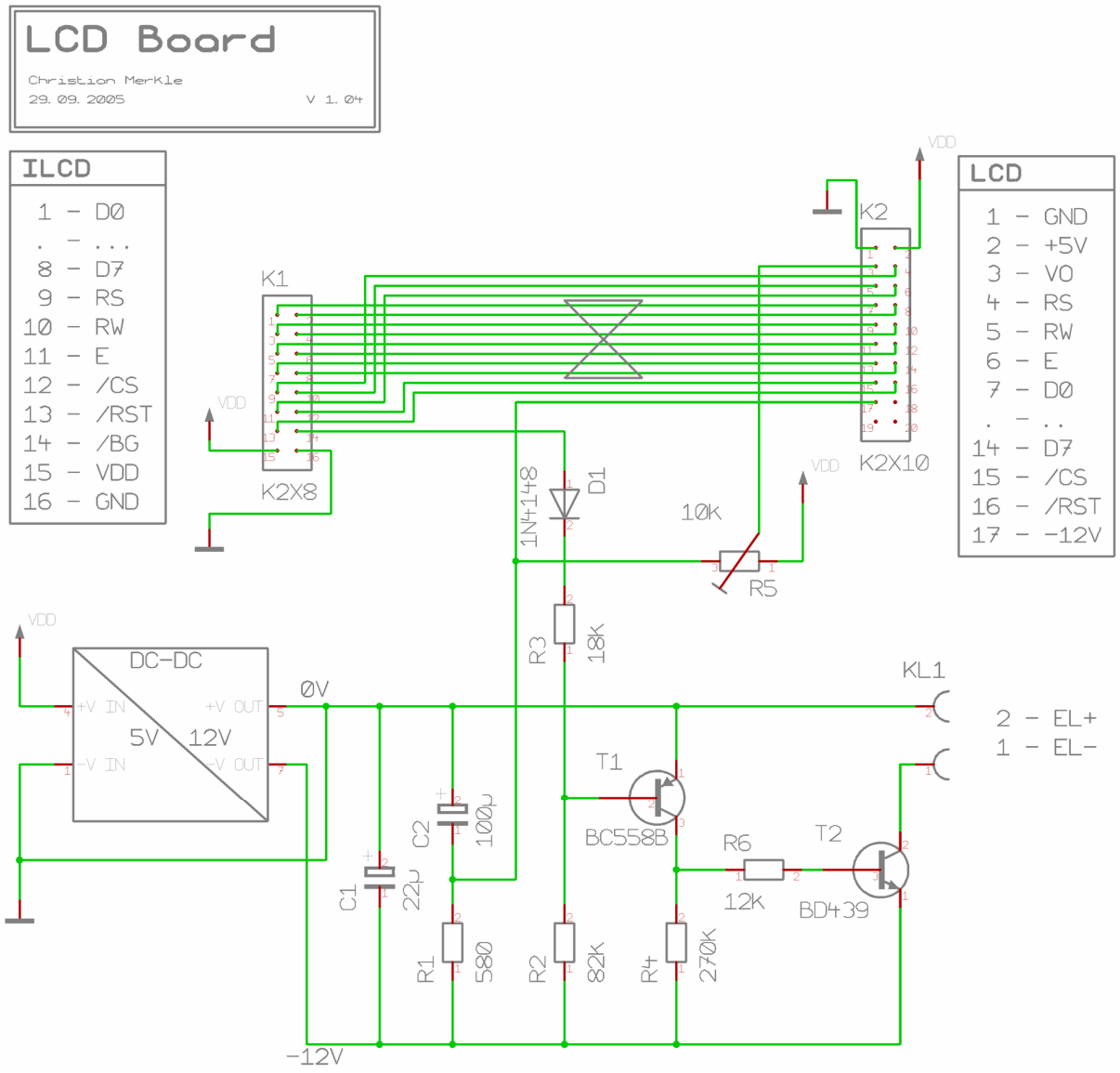


Abbildung 40 – Schaltplan LCD

3.4.2.6. DIMENSIONIERUNG DER BAUTEILE

Die Werte der zur Stabilisierung benötigten Kondensatoren sind unkritisch.

R1 sollte so hoch wie möglich gewählt werden, um eine möglichst gute Spannungsglättung zu erreichen, muß aber noch klein genug sein, um den erforderlichen Strom zu liefern. Die Stromaufnahme des negativen Eingangs des Displays ist im Vergleich zu dem (laut Datenblatt [LCD]) notwendigen Potentiometer-Wertes von 10kΩ vernachlässigbar. Die vollen 17V würden am Poti einen Strom von 1.7mA fließen lassen. Damit ist der Spannungsabfall bei *R1*, mit fast einem Volt, schon recht hoch. Aber die Spannung ist damit nahezu vollständig geglättet und das Display flimmert nicht mehr.

Der EL-Inverter nimmt gemessene 45mA auf. Zur Sicherheit werden 100mA angenommen. Da der Verstärkungsfaktor des verwendeten Leistungstransistors BD439 mindestens 100 beträgt, wird für den Basisstrom I_B 1mA angenommen, was bei 12V einem Widerstand von 12kΩ für *R6* entspricht.

R4 ist wieder unkritisch und sollte nur möglichst groß gewählt werden.

Um das Verhältnis von 0.78 für den Spannungsteiler herzustellen, wird für *R2* 82kΩ und für R3 18kΩ verwendet.

Der Basisstrom des PNP-Transistors muß bei der Verstärkung von 200 und einem minimalen Kollektorstrom von kapp über 1mA für den BD439 und den Pull-Down-Widerstand *R4* größer als 5µA sein. Der Basiswiderstand *R3* erreicht diesen Wert spielend.

Der Spindeltrimmer *R5* sollte, wie schon erwähnt, laut Datenblatt 10kΩ betragen.

3.4.2.7. STROMAUFNAHME

Der Stromverbrauch setzt sich zu einem Teil aus dem 5V Verbrauch des LCD-Displays zusammen. Laut Hersteller besitzt es eine maximale Leistungsaufnahme von 250mW, also 50mA.

Auf der anderen, der negativen, Seite benötigt der EL-Inverter maximal 50mA und die Versorgung des Displays weniger als 2mA. Der DC/DC Wandler hat einen recht schlechten Wirkungsgrad von nur 70% und eine statische Aufnahme von etwa 10mA.

Bei eingeschalteter Hintergrundbeleuchtung:

$$\frac{52mA \cdot \frac{12V}{5V}}{0.7} + 10mA \longrightarrow 189mA$$

Und im ausgeschaltetem Zustand:

$$\frac{2mA \cdot \frac{12V}{5V}}{0.7} + 10mA \longrightarrow 17mA$$

Der Gesamtverbrauch beläuft sich also mit eingeschaltetem Hintergrund auf 239mA und ohne Beleuchtung auf 67mA.

GEMESSENE WERTE

Status	Stromaufnahme
Hintergrundbeleuchtung AUS	36mA
Hintergrundbeleuchtung AN	128mA

Tabelle 20 – LCD-Modul, Gemessene Stromwerte

FAZIT

Die reale Stromaufnahme bleibt mit den gemessenen Werten um fast die Hälfte unter den berechneten, was sehr erfreulich ist.

3.4.2.8. FOTOS VOM AUFBAU UND DER FERTIGEN PLATINE

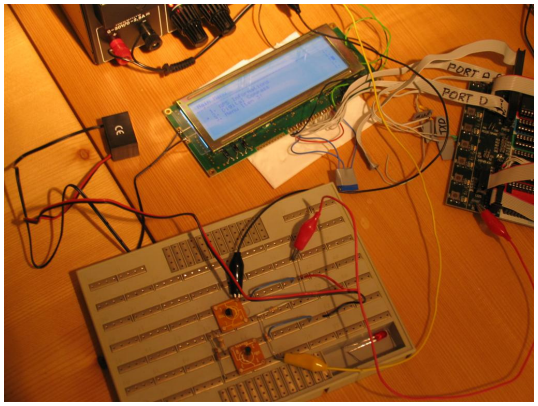


Abbildung 41 – LCD-Modul, Testläufe

Während der Schaltungsentwicklung werden einige Tests zum Schalten der Hintergrundbeleuchtung gemacht.

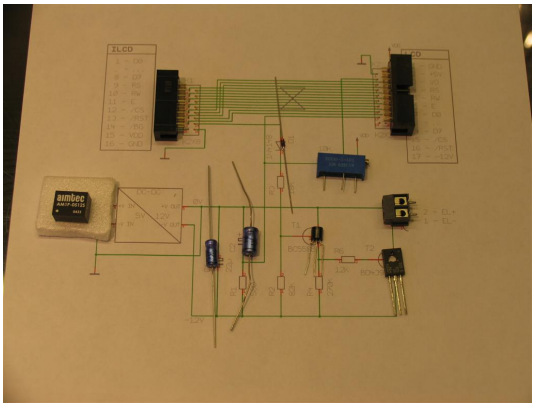


Abbildung 42 – LCD-Modul, Bauteilauswahl

Wieder sind alle notwendigen Bauteile, bis auf die Widerstände, bereitgelegt.

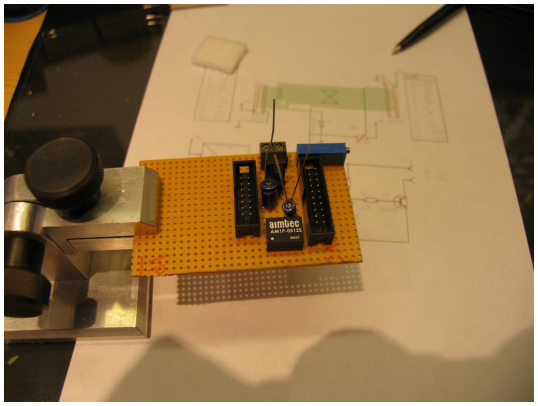


Abbildung 43 – LCD-Modul, Layout

Ein erstes Probelayout ist erstellt.

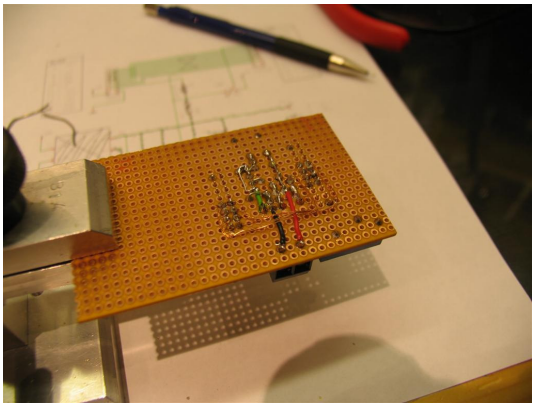


Abbildung 44 – LCD-Modul, Lötfortschritt

Pin für Pin werden die Leitungen gelegt.

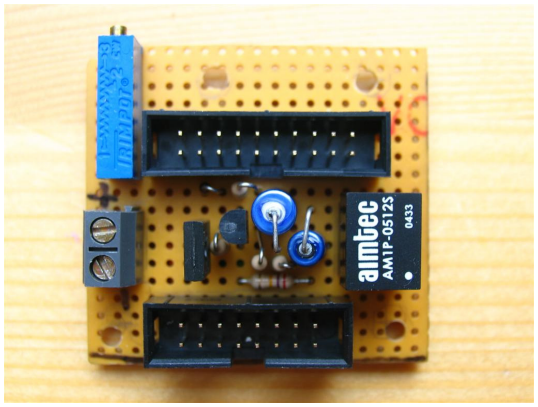


Abbildung 45 – LCD-Modul, Vorderansicht

Draufsicht: Oben: 20-poliger LCD-Connector. Unten: 16-poliger ILCD-Connector. Von rechts nach links: DC/DC-Wandler, geglättete LCD-Stromversorgung, Schaltung für Hintergrundbeleuchtung, Transistor für EL-Inverter, Spindeltrimmer zur Kontrasteinstellung.

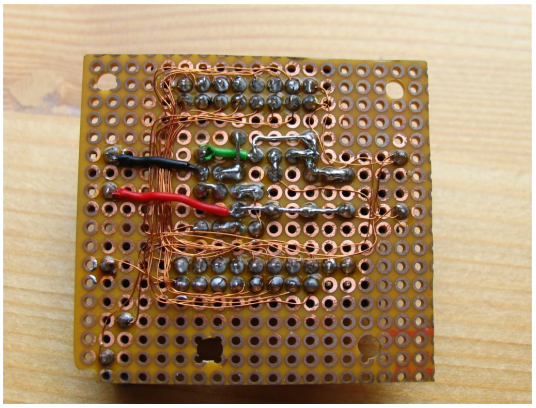


Abbildung 46 – LCD-Modul, Rückseite

Ansicht von unten: Wie auch bei allen anderen Modulen sind hier die stromführenden Leitungen wieder mit größerem Durchschnitt verlegt.

3.4.3. GPS – GPS Board

3.4.3.1. Aufgabe

Die Aufgabe dieses Moduls ist die Entgegennahme der GPS-Daten, sprich die Versorgung des Empfängers und die Quellauswahl der Daten.

Das Board soll zwei Verbinder in die Außenwelt besitzen: Der erste ist ein 6-poliger Mini-DIN Stecker für den Holux GPS-Empfänger. Er besteht aus drei Teilen: Der Stromversorgung des Empfängers, einer seriellen CMOS- und einer seriellen RS-232 Schnittstelle.

Der zweite Stecker ist eine normale 9-polige serielle RS-232 Schnittstelle, um auch beliebig andere GPS-Geräte an den Datenlogger anzuschließen.

Dieses Modul soll einen der drei Eingangskanäle an den beiden physikalischen Steckern per Software auswählen und auf den einen gemeinsamen Receive-Kanal für den Hauptprozessor abbilden.

Des weiteren muß der GPS-Empfänger über eine Datenleitung ein- und ausgeschaltet werden können.

3.4.3.2. PINBELEGUNG DES IGPS-CONNECTORS

Der IGPS-Connector ist 10-polig und es werden 7 Leitungen benutzt:

ILCD	Typ[19]	Signal	Beschreibung
3	O	CH0	Channel Select 0
4	O	CH1	Channel Select 1
5	IF	RC	Serial Receive
6	OF	TR	Serial Transmit
8	O	ON	Power GPS On
9		VDD	
10		GND	

Tabelle 21 – Pinbelegung IGPS-Connector

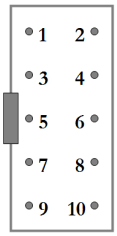


Abbildung 47 – 10-polig

Damit benötigen wir folgende Ressourcen am Mikrocontroller:

Typ	Anzahl
O	3
UART	2
	5

Tabelle 22 – Ressourcenbelegung IGPS-Connector

3.4.3.3. PINBELEGUNG DES GPS-CONNECTORS

Der GPS-Connector ist der 6-polige Mini-DIN Stecker für den Holux-Empfänger.

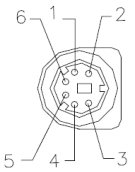


Abbildung 48 - Pinbelegung Mini-DIN 6-polig

GPS	Typ[20]	Signal	Pegel	Beschreibung	DIN-Stecker Pin
1	I	RS-R	RS-232	Serial RS-232 Receive	6
2	O	RS-T	RS-232	Serial RS-232 Transmit	1
3	I	CM-R	CMOS	Serial CMOS Receive	5
4	O	CM-T	CMOS	Serial CMOS Transmit	3
5	Sup	VDD		Supply +5V	2
6	Sup	GND		Supply Ground	4

Tabelle 23 – Pinbelegung GPS-Connector

3.4.3.4. PINBELEGUNG DER SERIELLEN RS-232 SCHNITTSTELLE

Hier werden nur die Receive- und Transmit-Leitungen beschaltet. Die restlichen Signale, wie RTS, CTS usw., sind entsprechend zu verbinden.

In diesem Fall sind wir das DTE und der GPS-Empfänger ist das DCE. DTE bedeutet *Data Terminal Equipment* und ist normalerweise der Computer. DCE bedeutet *Data Circuit-Terminating Equipment* oder *Data Communications Equipment* und ist in der PC-Welt meistens ein Modem. Diese Festlegung ist wichtig, wenn es um die Beschaltung der Pins geht.

[19] Datenrichtung vom Controller: O: Output; IF: Input on fixed pin; OF: Output on fixed pin
[20] Datenrichtung vom Modul: I: Input; O: Output; Sup: Supply

Damit sieht die DTE Pinbelegung der 9-poligen seriellen Schnittstelle so aus:

DTE Pin	Signal	Name	Richtung	Bedeutung für Modul
1	DCD	Data Carrier Detected	I	
2	RxD	Receive Data	I	TxD vom GPS-Empfänger
3	TxD	Transmit Data	O	RxD vom GPS-Empfänger
4	DTR	Data Terminal Ready	O	
5	GND	Ground		Masse
6	DSR	Dataset Ready	I	
7	RTS	Request To Send	O	
8	CTS	Clear To Send	I	
9	RI	Ring Indicator	I	

Tabelle 24 – Serielle Schnittstelle, 9-polig, Pinbelegung

Abschließend noch das Pin-Layout dieser Schnittstelle:

Abbildung 49 - Serielle Schnittstelle, 9-polig

3.4.3.5. Probleme und deren Lösungen

Schalten des GPS-Empfängers

Das Schalten des Empfängers gestaltet sich verhältnismäßig einfach. Der laut Datenblatt [GM210] von Holux maximale Stromverbrauch liegt bei 170 mA. Deshalb ist für *T1* der Leistungstransistor BD439, anstelle des Universellen, einzusetzen.

Receive Signal – Eingangsquellenauswahl

Um nun eine der drei Eingangsquellen auszuwählen, benötigt man einen Multiplexer. Der aus der 4000er CMOS-Reihe stammende 4539 vollbringt hier gute Dienste. Dafür wird der MC14539 von Motorola [4539] eingesetzt. Das ist ein *Dual 4-Channel Data Selector/Multiplexer*, wobei allerdings nur einer der beiden Multiplexer verwendet wird. Er sitzt in einem 16-poligem PDIP Gehäuse:

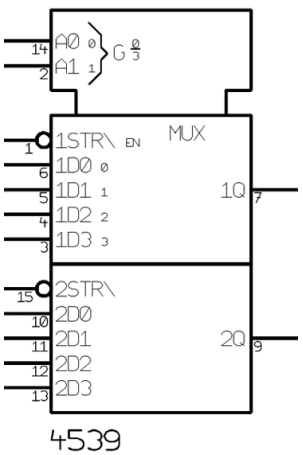
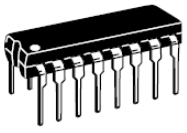


Abbildung 50 – MC14539 Blockschaltbild



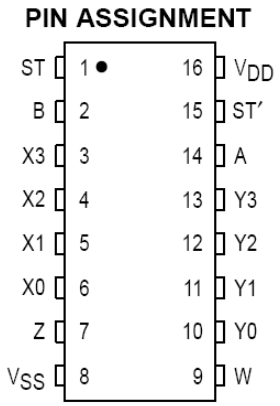


Abbildung 51 – MC14539 Pinbelegung, aus [4539]

Mit A0 und A1 kann binär zwischen den 4 möglichen Eingangssignalen (D0 – D3) ausgewählt werden. Diese routet das Bauteil dann auf den Ausgang Q. Mit dem /Strobe Eingang kann der Ausgang manuell auf 0 gesetzt werden. Dieses Feature wird hier aber nicht verwendet.

Da wir nur 3 Eingänge abzubilden haben, wird der Eingang 0 auch auf 0 gesetzt. Somit soll folgende Abbildung realisiert werden:

Nr.	Eingang
0	NULL
1	Mini-DIN – CMOS
2	Mini-DIN – RS-232
3	externe RS-232 Schnittstelle

Tabelle 25 – GPS Eingang: Quellenauswahl

Zum Festlegen eines sicheren Ausgangszustandes ziehen die beiden Widerstände *R1* und *R2* die Adressierungsleitungen auf 0.

Der Mini-DIN – CMOS Eingang ist, wie der Name schon sagt, bereits auf CMOS Pegel und kann somit mit nur einem Sicherheitswiderstand direkt an den Multiplexer am Eingang 1 angeschlossen werden.

Etwas mehr Mühe hat man mit den beiden RS-232 Eingängen, die laut Spezifikation einen Pegel von -12V bis 12V besitzen. Dafür muß ein Levelkonverter eingesetzt werden. Hierzu eignet sich der MAX232 hervorragend (siehe [MAX232]):

Er stellt jeweils zwei unabhängige RS-232 zu CMOS und CMOS zu RS-232 Konverter zur Verfügung. Zur Erzeugung der erforderlichen +/- 12V besitzt er intern die komplette, notwendige Hardware und benötigt nur 4 externe 1µF Kondensatoren:

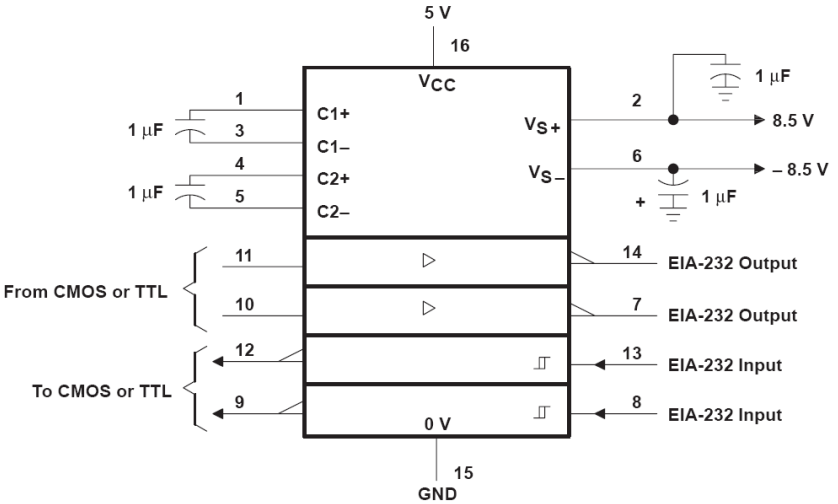


Abbildung 52 – MAX232 Blockschaltbild, aus [MAX232]

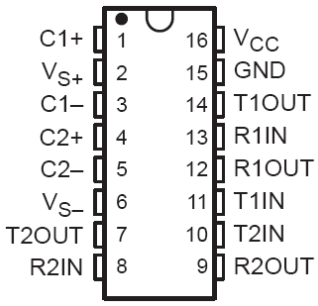


Abbildung 53 – Pinbelegung MAX232, aus [MAX232]

Die beiden RS-232 Eingänge werden also über die Sicherheitswiderstände *R14 / R13* an die RS-232 Seite des Level-Konverters angeschlossen. Das konvertierte CMOS Signal wird wieder direkt mit den Eingängen 2 und 3 des Multiplexers verbunden. Somit kann man aus allen drei Quellen das Signal auswählen.

TRANSMIT SIGNAL – SIGNALAUSGANG

Das Transmit Signal der seriellen Schnittstelle wird direkt auf alle drei Ausgänge geschalt, da dadurch keine Nachteile entstehen. An dem CMOS Ausgang kann dies direkt geschehen, an den beiden RS-232 Ausgängen aber nicht, da die Signale erst auf den richtigen Pegel von +/- 12V gebracht werden müssen. Die beiden CMOS Eingänge des MAX232 Level-Konverters sind dafür noch frei.

3.4.3.6. SCHALTPLAN

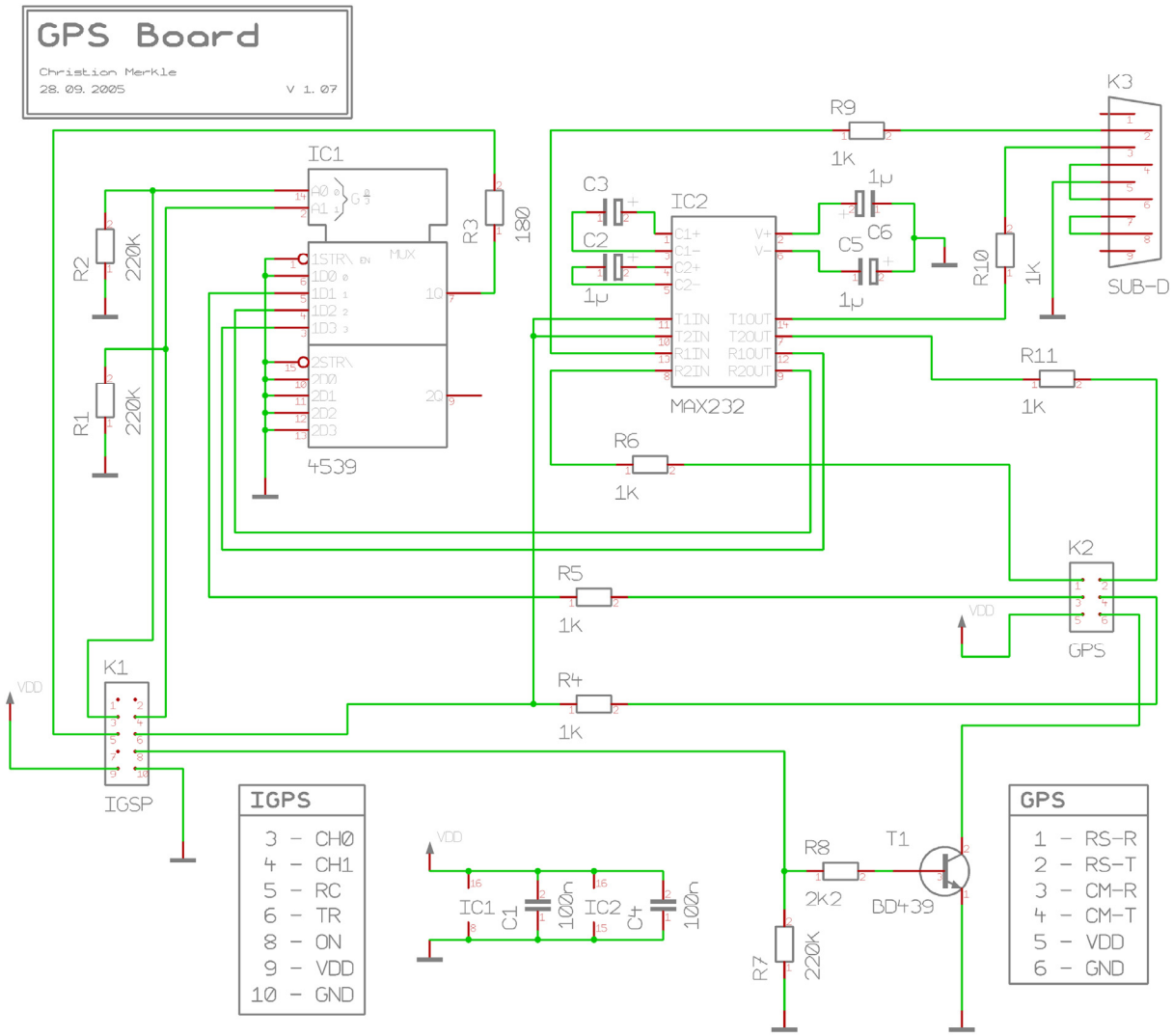


Abbildung 54 – Schaltplan GPS

3.4.3.7. DIMENSIONIERUNG DER BAUTEILE

Die Dimensionierung der meisten Bauteile ist in diesem Modul recht einfach:

Die Pull-Down Widerstände *R1*, *R2* und *R10* sind möglichst groß zu wählen. Die restlichen Sicherheitswiderstände müssen nur so groß gewählt werden, daß keine Zerstörung bei versehentlich falsch angesteckten Signalen von außen eintreten kann. Bei 5V und 1kΩ beträgt der maximale Strom noch recht unkritische 5mA.

Die vier Kondensatoren für den Level-Konverter sind, wie vom Datenblatt verlangt, vier 1µF Elektrolytkondensatoren.

Bei der Schaltstufe bei *T1* ist eine kleine Stromberechnung durchzuführen:

Um auch andere, kompatible Empfänger verwenden zu können, wird hier ein maximaler Strom von 200mA angenommen. Der Transistor besitzt eine Verstärkung von 100. Somit sollte der Basisstrom etwas mehr als 2mA betragen. 5V / 2mA = 2.5kΩ, der nächst kleinere Wert für *R11* ist demnach 2.2kΩ.

3.4.3.8. STROMAUFNAHME

Die maximale Stromaufnahme des Multiplexers ist vernachlässigbar und liegt laut Datenblatt im niedrigen µA-Bereich, ganz im Gegensatz zum MAX232, der sich erst mit doch recht beachtlichen 10mA zufrieden geben soll.

Hinzu kommt noch der GPS-Empfänger, der eingeschaltet maximal 170mA aufnehmen soll.

Zusammen also höchstens 180mA im eingeschalteten und 10mA im ausgeschalteten Zustand.

GEMESSENE WERTE

Status	Stromaufnahme
GPS AUS	10.8mA
GPS AN	88.0mA

Tabelle 26 – GPS-Modul, Gemessene Stromwerte

FAZIT

Die Stromaufnahme mit ausgeschaltetem GPS-Empfänger ist sogar noch höher als berechnet. Wenigsten bleibt die – wesentlich öfters auftretende – Situation, daß der Empfänger aktiv ist, deutlich hinter den Berechnungen zurück.

3.4.3.9. Fotos vom Aufbau und der fertigen Platine

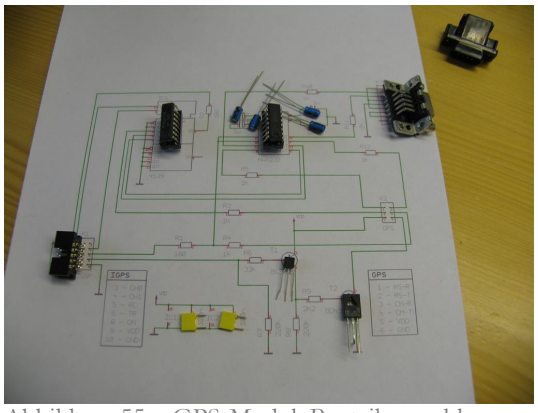


Abbildung 55 – GPS-Modul, Bauteilauswahl

Auch hier werden wieder zuerst alle notwendigen Teile zurechtgelegt und verschiedene Layouts getestet.

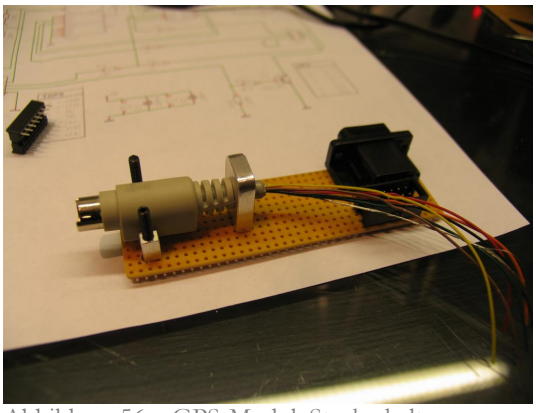


Abbildung 56 – GPS-Modul, Steckerhalter

Da im Gegensatz zu den reichlich angebotenen Buchsen kein 6-poliger Mini-DIN Stecker zum Einbau aufgetrieben werden konnte, wird kurzerhand eine eigene Halterung dafür gefräst.

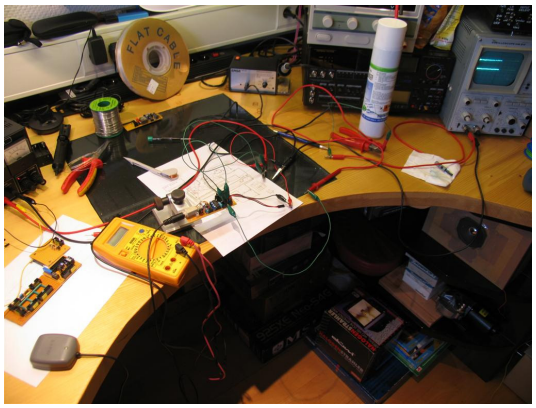


Abbildung 57 – GPS-Modul, Testläufe

Nach dem Zusammenbau wird der Multiplexer und der Levelkonverter getestet. Rechts im Bild sieht man auf dem Oszilloskop den gewünschten CMOS-Pegel Ausschlag vom RS-232 Eingang.

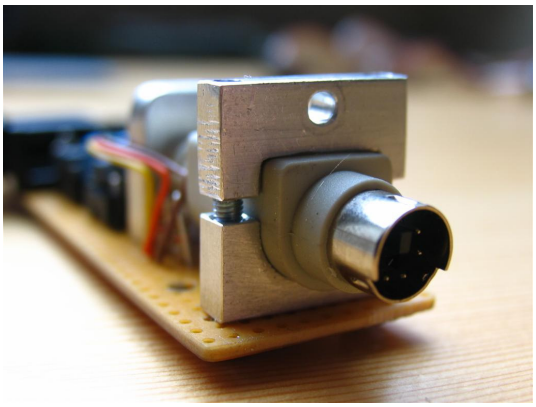


Abbildung 58 – GPS-Modul, Steckerhalter 2

Die Steckerhalterung besitzt oben ein zusätzliches Loch, um später fest mit dem Gehäuse verbunden zu werden. Dadurch können, auch beim relativ gewaltsamen Abziehen des Steckers, keine Schäden auftreten.

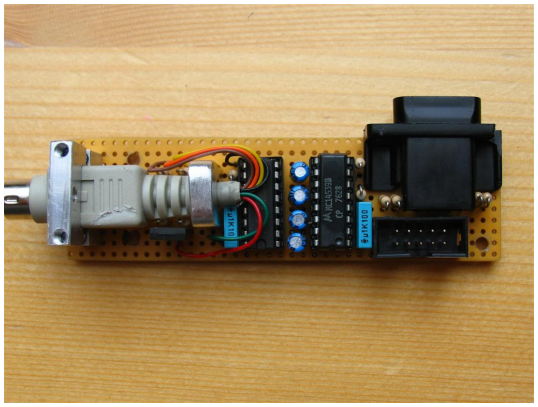


Abbildung 59 – GPS-Modul, Vorderansicht
Ansicht von oben: Von links nach rechts: Holux Stecker, Transistor zum Schalten des GPS-Empfängers, MAX232 Levelkonverter, Multiplexer, serielle Schnittstelle und 10-poliger IGPS-Connector

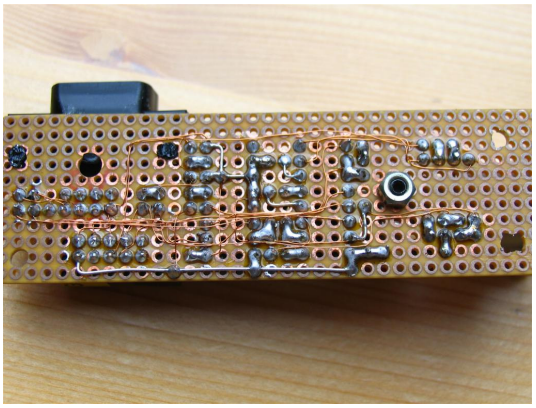


Abbildung 60 – GPS-Modul, Rückseite
Die Unterseite.

3.4.4. IO – IO Board

3.4.4.1. Aufgabe

Der Aufbau des IO-Moduls ist ebenso einfach, wie dessen Aufgabe: Diese Platine befindet sich nicht, wie die restlichen, unten im inneren des Gerätes, sondern auf der Rückseite der Frontplatte. Sie soll die richtig plazierten fünf Taster und drei LEDs aufnehmen und deren Signale an den Mikrocontroller weiterleiten.

Der erste Taster ist der Powerbutton. Dieser muß seine beiden Anschlußpunkte getrennt und direkt an das Power-Modul weiterleiten. Die restlichen Taster gehen an den Controller und haben folgende Aufgaben: Auf, Ab, Menü/OK und Abbrechen.

Die drei LEDs haben folgende Funktionen: Power-LED (grün), MMC-Zugriff (rot) und Aufnahme/Record (ebenfalls rot). Sie sollen direkt an 5V betrieben werden können.

Aus Platzgründen steht nur ein 10-poliger Connector zur Verfügung.

Ein Problem ergibt sich allerdings schon bei der Aufgabenstellung: Da die drei LEDs und vier Taster jeweils eine Signalleitung belegen und der Powerbutton gleich zwei Leitungen belegt, benötigt man hier schon insgesamt neun Leitungen. Deshalb kann nicht, wie bei allen anderen Modulen, die komplette Versorgungsspannung mit VDD und GND angeschlossen werden sondern nur ein Teil davon, nämlich GND. Das Modul ist so zu konstruieren, daß die +5V auf dem Modul nicht nötig sind.

3.4.4.2. PINBELEGUNG DES IIO-CONNECTORS

Der IIO-Connector ist 10-polig und es werden alle 10 Leitungen benutzt:

ILCD	Typ[21]	Signal	Beschreibung
1	S	PS1	Power Switch Side 1
2	S	PS2	Power Switch Side 2
3	O	LR	LED Recode
4	S	LM	LED MMC Access
5	O	LP	LED Power
6	I	SE	Switch Escape
7	I	SM	Switch Menu/OK
8	I	SU	Switch Up
9	I	SD	Switch Down
10		GND	

Tabelle 27 – Pinbelegung IIO-Connector

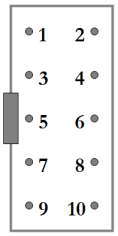


Abbildung 61 – 10-polig

Damit belegt dieses Modul folgende Ressourcen am Mikrocontroller:

Typ	Anzahl
I	4
O	2
	6

Tabelle 28 – Ressourcenbelegung IIO-Connector

3.4.4.3. PROBLEME UND DEREN LÖSUNGEN

POWERBUTTON

Hier gibt es erfreulicherweise gar kein Problem: Die beiden Anschlüsse werden einfach direkt mit Pin 1 und 2 verbunden.

TASTER – EINGABE

Die restlichen Taster werden im Active-Low-Modus angeschlossen. Das bedeutet, daß wenn die Taster gedrückt sind, sie die Leitung auf Masse ziehen. Ein zusätzlicher Widerstand in den Leitungen soll nur bei versehentlich falsch geschaltetem Ausgang am Prozessor schlimmeres verhindern. Wenn der Taster nicht gedrückt ist, ist das Signal im sogenannten *floating*-Zusatand. Das bedeutet, es liegt weder auf LOW, noch auf HIGH. Da auf dem Modul keine +5V vorhanden sind, muß das Signal irgendwo anders durch einen Pull-Up-Widerstand auf den Standardzustand HIGH gelegt werden. Dies könnte entweder extern auf der Main Unit geschehen oder, da dies die AVR Controller schon anbieten, intern direkt im Mikrocontroller durch einen per Software schaltbaren Pull-Up-Widerstand.

LEDS – AUSGABE

Technisch sinnvoller wäre es, die Anode der LED fest mit der Versorgungsspannung zu verbinden und die Leuchtdiode über die Kathode mit 0 einzuschalten, da die AVR-Mikrocontroller mehr Strom im *Sink*-Betrieb aufnehmen können, als sie im *Source*-Betrieb abgeben. Da wir aber keine +5V zur Verfügung haben steuern wir die LEDs einfach im nicht invertierenden Betrieb an und legen die Kathode direkt auf Masse.

[21] Datenrichtung vom Controller: S: Special (connected somewhere else); I: Input; O: Output

3.4.4.4. SCHALTPLAN

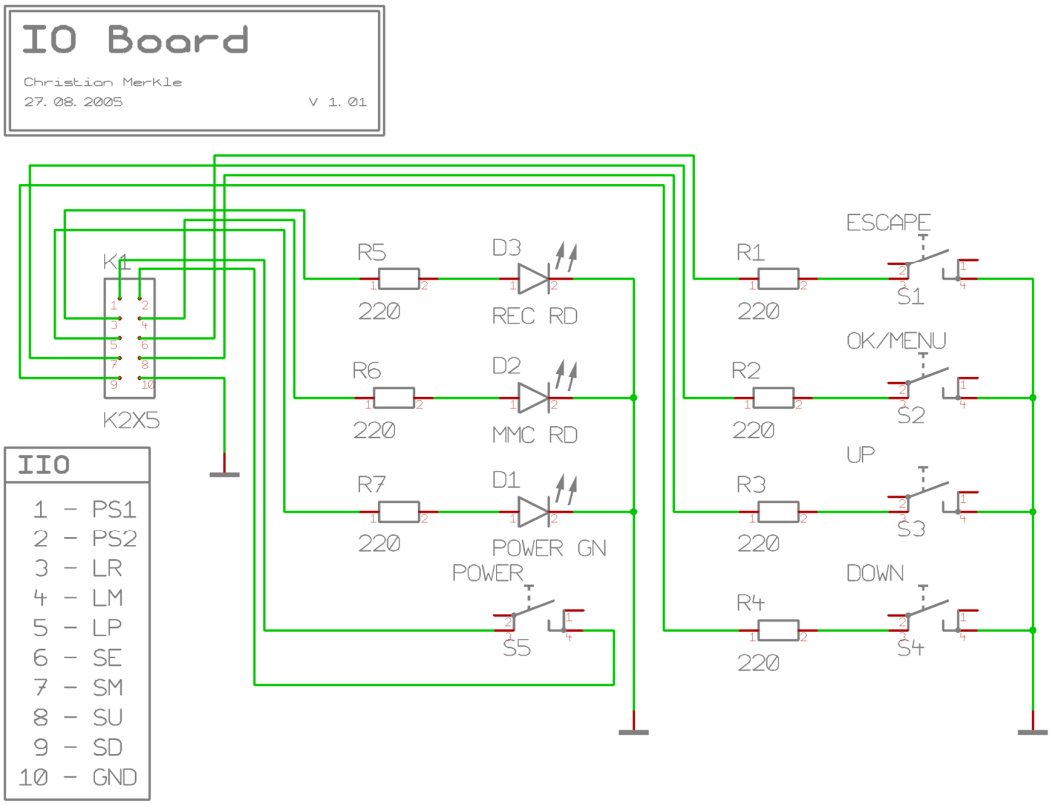


Abbildung 62 – Schaltplan IO

3.4.4.5. DIMENSIONIERUNG DER BAUTEILE

Die sieben Widerstände sollen den Strom durch die LEDs oder – bei falscher Belegung – auch den durch die Taster begrenzen. Bei 5V und 220Ω würden maximal 23mA fließen. Laut Datenblatt [AVRM32] von Atmel besitzen die IO-Ausgänge im Source-Betrieb bei etwa 20mA nur noch maximal 4.6V. Das entspricht dann nur noch knapp 21mA pro LED.

3.4.4.6. STROMAUFNAHME

Die maximale Stromaufnahme liegt an, wenn alle drei LEDs gleichzeitig leuchten, also 3 * 21mA = 63mA.

GEMESSENE WERTE

Status	Stromaufnahme
Alle LEDs an	20.2mA

Tabelle 29 – IO-Modul, Gemessene Stromwerte

3.4.4.7. FOTOS VOM AUFBAU UND DER FERTIGEN PLATINE

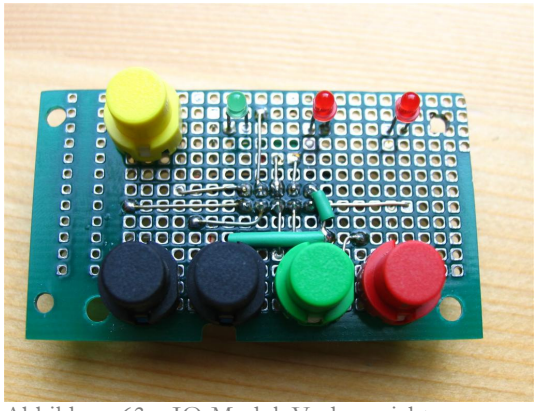


Abbildung 63 – IO-Modul, Voderansicht

Von links nach rechts: Oben: Power-Switch, Power-LED, MMC-LED, Record-LED. Unten: Taster: Down, Up, Menü/OK, Escape.

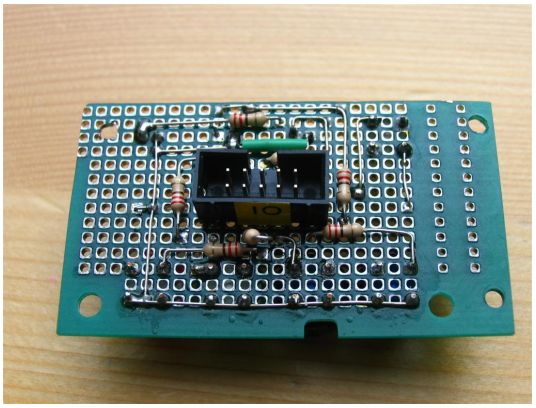


Abbildung 64 – IO-Modul, Rückseite

Von unten sind die Widerstände und der 10-polige IIO-Connector sichtbar.

3.4.5. MMC – MMC BOARD

3.4.5.1. AUFGABE

Das MMC-Modul ist neben dem gerade beschrieben IO-Modul das zweit einfachste und auch das letzte verwendete Modul.

Es soll lediglich die MMC Fassung und damit die Speicherkarte aufnehmen und die Karte mit 3.6V versorgen und einen Level-Konverter bereitstellen, um die 5V des Prozessors auf die benötigten 3.6V für die MMC-Karte und umgekehrt, abzubilden.

Zusätzlich dazu soll eine Ausgangsleitung vorhanden sein, die direkt an die MMC-LED des IO-Moduls angeschlossen wird und jeden Zugriff auf die MMC-Karte anzeigt. Um die LED direkt ansteuern zu können, muß der Ausgangsstrom des Ausgangs mehr als 21mA für die Leuchtdiode liefern können.

3.4.5.2. PINBELEGUNG DES IMMC-CONNECTORS

Der IMMC-Connector ist 10-polig und es werden 7 Leitungen benutzt:

ILCD	Typ[22]	Signal	Beschreibung
4	S	ACT	MMC Access Active
5	I	DO	MMC Data Out
6	O	DI	MMC Data In
7	O	CK	Clock
8	O	/CS	Chip Select
9		VDD	
10		GND	

Tabelle 30 – Pinbelegung IMMC-Connector

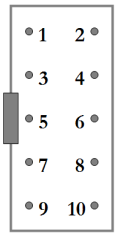


Abbildung 65 – 10-polig

Damit werden folgende Ressourcen am Mikrocontroller benötigt:

Typ	Anzahl
I	1
O	3
	4

Tabelle 31 – Ressourcenbelegung IMMC-Connector

3.4.5.3. PROBLEME UND DEREN LÖSUNGEN

LEVELKONVERTIERUNG 5V -> 3.6V

Um die aufwendige Arbeit mit Level-Konvertern zu umgehen, werden hier nur einfache Spannungsteiler eingesetzt. In einschlägigen Artikeln[23] wird berichtet, daß diese Vorgehensweise vorzüglich funktioniert.

Um 5V auf 3.6V herunterzusetzen, ist ein Teilungsverhältnis von 0.72 nötig, was die Spannungsteiler *R5/R4*, *R7/R6* und *R9/R8* übernehmen.

LEVELKONVERTIERUNG 3.6V -> 5V

Die andere Richtung kann man sich eigentlich komplett sparen, da der Mikrocontroller die 3.6V schon als HIGH erkennt. *R2* wird lediglich zur Sicherheit verwendet, um Schäden an der Hardware zu vermeiden. Der AVR-Controller erkennt ein Signal ab 0.6 * VDD als HIGH, was bei 5V immerhin 3V sind. Andererseits geben die MMC-Karten einen HIGH-Pegel von mehr als 3V aus.

STROMVERSORGUNG DER MMC-KARTE MIT 3.6V

Um sich einen extra Spannungsregler für die Erzeugung der 3.6V zu sparen, bedient man sich oft eines einfachen Tricks: Man nutzt die Diffusionsspannung in Dioden aus: Es werden die zwei Dioden *D1* und *D2* in Reihe geschaltet. An jeder fällt nun eine Spannung von etwa 0.7V ab, zusammen also 1.4V. Dies ergibt eine Versorgungsspannung von 3.6V.

Um die Spannung konstant auf diesem Pegel zu halten und keine Spitzen beim Einstecken der Karte zu erzeugen, soll mit Hilfe von *R10* ein ständiger Strom durch die Dioden fließen.

[22] Datenrichtung vom Controller: S: Special (connected somewhere else); I: Input; O: Output
[23] Homepage: www.ulrichradig.de

Der Kondensator *C1* stabilisiert diese Spannung zusätzlich ein wenig und glättet die Stromspitzen der Speicherkarte ein wenig.

In dem in Serie gehenden Produkt ist zuvor noch einmal zu prüfen, ob es nicht besser ist, einen richtigen Spannungswandler und gegebenenfalls auch richtige Level-Konverter, einzusetzen. Für den Prototyp reicht unsere Lösung aber auf jeden Fall aus.

ACTIVE SIGNAL

Um anzuzeigen, daß gerade ein Zugriff auf die MMC-Karte stattfindet, verwendet man das /CS Signal. Ist dieses LOW, so findet ein Zugriff statt. Um nun die Zugriffs-LED ansteuern zu können, muß ein PNP-, anstelle eines NPN-Transistors verwendet werden. Er wird direkt auf das /CS Signal vor dem Spannungsteiler geschaltet.

3.4.5.4. SCHALTPLAN

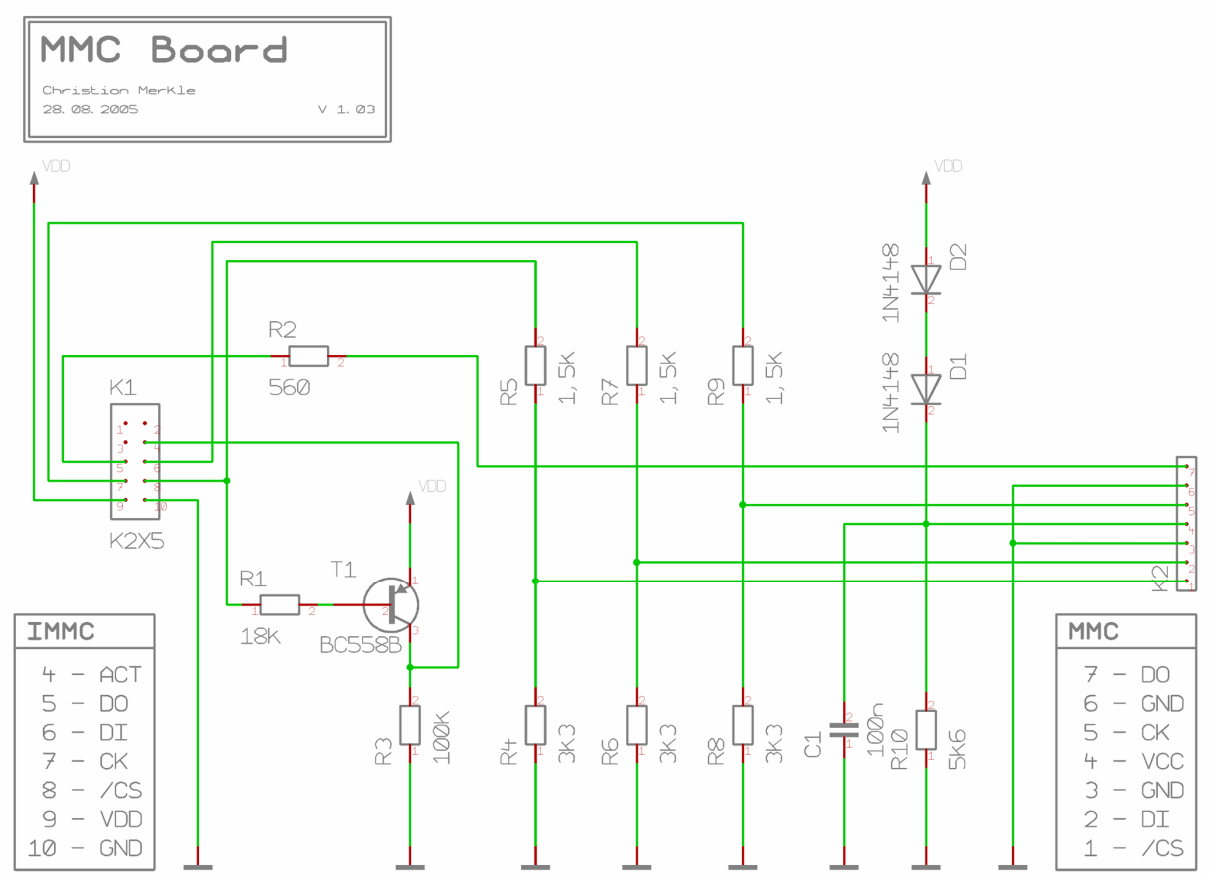


Abbildung 66 – Schaltplan MMC

3.4.5.5. DIMENSIONIERUNG DER BAUTEILE

Die Spannungsteiler sollen einen Faktor von etwas weniger als 0.72 für die exakten 3.6V besitzen: Die Werte 1.5kΩ und 3.3kΩ erledigen diese Aufgabe mit 0.69 hervorragend und geben damit 3.45V aus.

Um die 21mA der LED mit dem Transistor *T1* direkt schalten zu können, muß ein Basisstrom von mindestens 21mA / 180 = 0.12mA fließen. Bei 18kΩ gibt es also mehr als eine doppelte Sicherheit mit 0.28mA.

3.4.5.6. STROMAUFNAHME

Wenn alle drei Signalausgänge auf HIGH sind, dann fließen

$$3 \cdot \left(\frac{5V}{4.8k\Omega} \right) \longrightarrow 3.1mA$$

Die Spannungsversorgung über die zwei Dioden nimmt ebenfalls noch auf:
5V / 5.6kΩ = 0.9mA

$$\frac{5V}{5.6k\Omega} \longrightarrow 0.9mA$$

Die 0.28mA für den Transistor sind vernachlässigbar klein und der Strom für die LED selbst wurde schon beim IO-Modul mitberechnet.

Die Stromaufnahme der MMC Karten ist sehr unterschiedlich. Es wird hier einfach der maximale Wert aus dem Hitachi-Datenblatt [MMC] verwendet. Ohne Clocksignal wird hier weniger als 150µA angegeben. Bei Volllast mit 20MHz sollen immerhin maximal 80mA benötigt werden, typisch jedoch beim Schreibzugriff nur 35mA. Bei unserem Projekt werden die Karten, auf Grund des Systemtaktes von 8 MHz, über den SPI-Bus – mit einer Frequenz von 0.5 * f_{OSC} = 4MHz – deutlich weniger aufnehmen. Es wird eine lineare Abhängigkeit angenommen und damit bleiben von den 80mA nur noch 16mA übrig. Zur Sicherheit werden maximale 20mA angenommen.

Somit bleibt die maximale Stromaufnahme des MMC-Moduls unter 25mA, ohne Zugriff bei etwa 5mA.

GEMESSENE WERTE

Status	Stromaufnahme
Ohne MMC Karte	0.93mA
Mit MMC Karte	1.96mA

Tabelle 32 – MMC-Modul, Gemessene Stromwerte

FAZIT

Leider konnte die Stromaufnahme beim Zugriff nicht gemessen werden. Die anderen Werte sind aber sehr niedrig.

3.4.5.7. FOTOS VOM AUFBAU UND DER FERTIGEN PLATINE

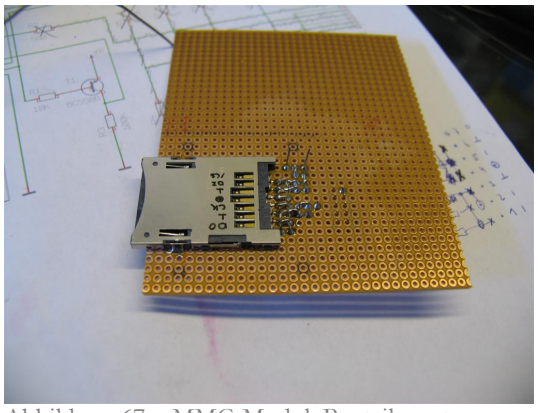


Abbildung 67 – MMC-Modul, Bauteilmontage

Die Bauteile werden aufgelötet. Als MMC-Fassung wird eine qualitativ hochwertige von Molux verwendet. Diese rastet nach einem Druck hörbar ein und gibt die Karte nach einem erneuten Druck wieder aus.

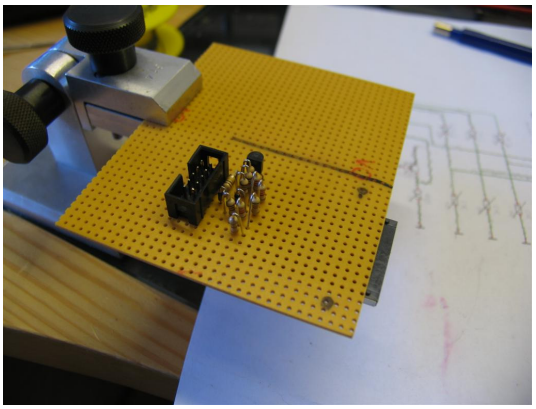


Abbildung 68 – MMC-Modul, Bauteilmontage 2

Die Oberseite während der Montage.

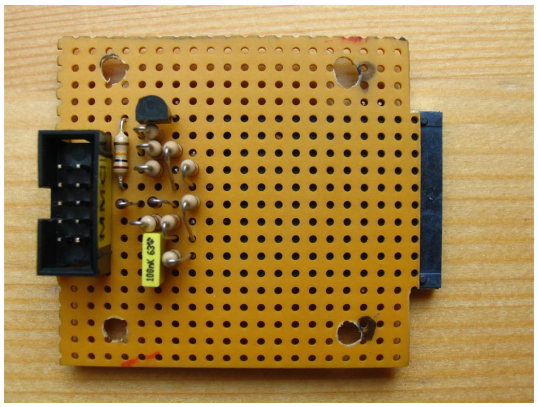


Abbildung 69 – MMC-Modul, Vorderansicht

Ansicht von oben: Von links nach rechts: 10-poliger IMMC-Connector, LED-Active-Schaltung, Levelkonverter–Schaltungen.

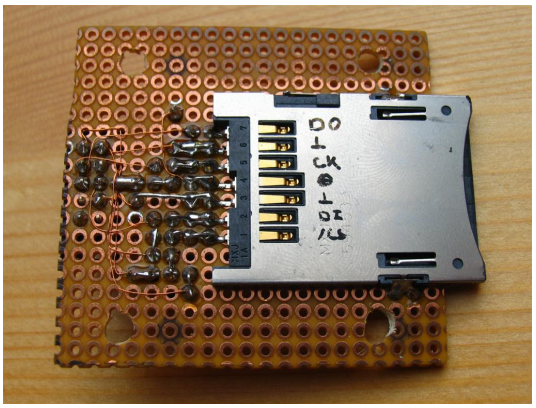


Abbildung 70 – MMC-Modul, Rückseite

Ansicht von unten: Hier sieht man Molex MMC-Fassung gut.

3.4.6. MU – MAIN UNIT

3.4.6.1. AUFGABE

Die Aufgabe der Main Unit ist ähnlich der eines Mainboards im PC.

Auf der Main Unit soll der AVR-Mikrocontroller mit allen notwendigen Komponenten, wie Spannungsversorgung, Takterzeugung und Resetschaltung, aufgenommen werden.

Ferner ist eine 6-polige ISP-Schnittstelle zu implementieren, damit der Controller auch im System neu programmiert werden kann.

Dann sollen alle Steckverbinder der anderen Module möglichst effektiv an die einzelnen Ports angeschlossen und – gegebenenfalls untereinander – richtig verbunden werden.

Zuletzt soll noch eine Schaltung für das IO-Modul entwickelt werden, um aus den 4 Taster-Signalen ein einzelnes Interrupt-Signal zu erzeugen, damit der Prozessor bei einem Tastendruck aus einem eventuellen Schlafzustand aufwachen kann.

3.4.6.2. PROBLEME UND DEREN LÖSUNGEN

STROMVERSORGUNG DES MIKROCONTROLLERS

Das gesamte System benötigt eine konstante Gleichspannung von 5V. Das für diese Aufgabe zuständige Power-Modul liefert diesen Strom, der alle anderen Module und die Main Unit nach einem Systemstart versorgt.

Der Prozessor benötigt eine stabilisierte Spannung, die nochmals mit einem Kondensator entkoppelt ist. Die Versorgungsspannung kann direkt an die Pins 10 (VCC) und 11 (GND) angeschlossen werden.

Zusätzlich dazu besitzt der Controller einen separaten Spannungseingang für den AD-Wandler. Dieser sollte zusätzlich noch einmal gefiltert werden, wenn man den Wandler einsetzt. Dadurch tritt weniger Rauschen auf. Falls der Wandler jedoch – wie bei uns – nicht verwendet wird, so können die analogen Versorgungsanschlüsse auch direkt ohne Filter beschaltet werden. Deshalb werden bei uns die Pins 30 (AVCC) und 31 (AGND) parallel zu den anderen digitalen Versorgungsanschlüssen geschaltet.

TAKTERZEUGUNG

Der Prozessor soll mit einem Systemtakt von 8 MHz laufen. Dazu wird nur ein kleiner Quarz-Oszillator zwischen die dafür vorgesehenen Pins 12 und 13 (XTAL1/2) gebracht und dessen zwei Anschlüsse über einen Kondensator mit 18pF auf Masse gelegt.

RESETSCHALTUNG

Wenn kein ISP verwendet wird, kann die /RESET-Leitung direkt auf VCC gelegt werden. Falls, wie in diesem Projekt, das Programmieren im Zielsystem jedoch verwendet wird, muß das

Programmiergerät das RESET-Signal von außen auf LOW setzen können, ohne einen Kurzschluß zu erzeugen. Deshalb soll laut Datenblatt ein Widerstand *R1* von 4.7kΩ verwendet werden.

Um die Resetleitung etwas fehlerunanfälliger zu gestalten, wird noch ein kleiner zusätzlicher Kondensator nach Masse geschaltet.

ISP

Der 6-polige ISP-Connector ist von Atmel wie folgt definiert:

ISP	Signal	Beschreibung
1	MISO	SPI Master In Slave Out
2	VTG	Supply +5V
3	SCK	SPI Serial Clock
4	MOSI	SPI Master Out Slave In
5	RST	Reset
6	GND	Supply Ground

Tabelle 33 – Pinbelegung IMMC-Connector

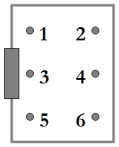


Abbildung 71 – 6-polig

VTG wird allerdings nicht direkt auf unsere Versorgungsspannung VDD gelegt, da bei nicht exakt gleichen Spannungswerten hier ein Strom aus oder in den Programmer fließen und damit eventuell Bauteile beschädigen würde. Zur Sicherheit bauen wir hierfür einen 10kΩ Widerstand *R2* ein.

Das Reset-Signal wird direkt am Reset-Pin des AVRs angeschlossen. So wie die drei restlichen Signale des SPI-Busses an die dafür vorgesehenen Pins von Port B.

INTERRUPTLEITUNG FÜR DIE TASTER

Da die Signale bei Tastendruck auf Masse gelegt werden und sonst durch den Pull-Up-Widerstand im Controller auf VDD sind, kann hier mit Dioden eine einfache Logikschaltung aufgebaut werden:

Von 5V werden über einen 10kΩ Widerstand *R3* vier Dioden *D1…4* angeschlossen und diese mit je einem Tastersignal verbunden. Die Interruptleitung wird nun direkt zwischen den Dioden und dem Widerstand abgegriffen. Ohne Tasterdruck leitet keine Diode und die Interruptleitung liegt über den Widerstand auf HIGH. Sobald aber ein Taster betätigt wird, leitet dessen Diode und zieht die Interruptleitung auf LOW.

3.4.6.3. PINBELEGUNGEN UND VERTEILUNGEN

Bei der Software ist eine möglichst einfache Konfiguration der Module an beliebige Ports und Pins nötig, damit die jeweiligen Module auch einzeln anderweitig eingesetzt werden können.

Die einzelnen Module werden laut folgender Tabelle mit dem Controller verbunden:

Modul	Pin	Signal	Typ	AVR Port / Bit	Bemerkung
IPWR	3	PS1	S		Connect to IIO PS1
	4	PS2	S		Connect to IIO PS2
	6	/PWRBUT	I	D2	
	7	/LOBAT	I	C6	
	8	KEEP	O	B0	
	9	VDD			
	10	GND			
ILCD	1	D0	I/O	A0	
	…	…	I/O	…	
	8	D7	I/O	A7	
	9	RS	O	D4	
	10	RW	O	D5	
	11	E	O	D6	
	12	/CS	O	D7	
	13	/RST	S		Connect to VDD, no reset
	14	/BG	O	B2	
	15	VDD			
	16	GND			
IMMC	4	ACT	S		Connect to IIO LM
	5	DO	I	B6	
	6	DI	O	B5	
	7	CK	O	B7	
	8	/CS	O	B4	
	9	VDD			
	10	GND			
IIO	1	PS1	S		Connect to IPWR PS1
	2	PS2	S		Connect to IPWR PS2
	3	LR	O	C5	
	4	LM	S		Connect to IMMC ACT
	5	LP	O	C4	
	6	SE	I	C3	
	7	SM	I	C2	
	8	SU	I	C1	
	9	SD	I	C0	
	10	GND			
IGPS	3	CH0	O	D1	
	4	CH1	O	C7	
	5	RC	I	D0	
	6	TR	S		Connect to GND, no transmit
	8	ON	O	B1	
	9	VDD			
	10	GND			

Tabelle 34 – Pinbelegung Controller, Sortiert nach Module

Es folgt eine Gesamtübersicht der Ports des Mikrocontrollers:

Port	Modul	Signal	Typ
A0	LCD	D0	I/O
…	LCD	…	I/O
A7	LCD	D7	I/O
B0	PWR	KEEP	O
B1	GPS	ON	O
B2	LCD	/BG	O
B3			
B4	MMC	/CS	O
B5	MMC	DI	O
B6	MMC	DO	I
B7	MMC	CK	O
C0	IO	SD	I
C1	IO	SU	I
C2	IO	SM	I
C3	IO	SE	I
C4	IO	LP	O
C5	IO	LR	O
C6	PWR	/LOBAT	I
C7	GPS	CH1	O
D0	GPS	RC	I
D1	GPS	CH0	O
D2	PWR	/PWRBUT	I, Int.
D3		/IO_INT	I, Int.
D4	LCD	RS	O
D5	LCD	RW	O
D6	LCD	E	O
D7	LCD	/CS	O

Tabelle 35 – Pinbelegung Controller, Sortiert nach Pins

3.4.6.4. SCHALTPLAN

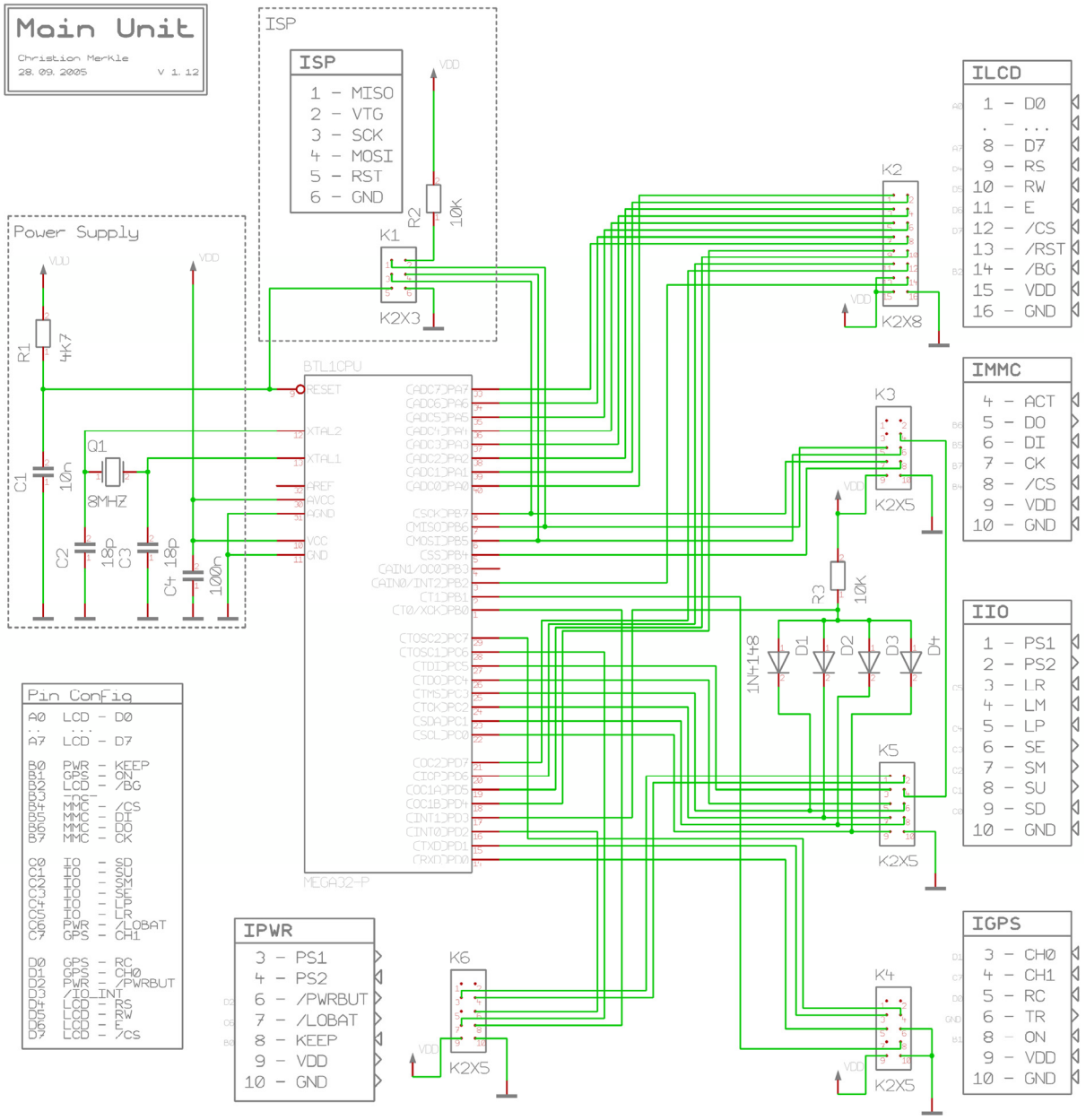


Abbildung 72 – Schaltplan Main Unit

3.4.6.5. STROMAUFNAHME

Neben den maximalen 0.5mA von *R10* nimmt auf der Main Unit nur noch der Mikrocontroller selbst Strom auf. Im aktiven Zustand bei 5V und 8 MHz laut Datenblatt [AVRM32] maximal 15mA und im Idle-Zustand noch 8mA.

Der maximale Stromverbrauch der Main Unit wird somit auf 16mA festgelegt.

GEMESSENE WERTE

Status	Stromaufnahme
Aktiv	13.8mA

Tabelle 36 – Main Unit, Gemessene Stromwerte

3.4.6.6. Fotos vom Aufbau und der fertigen Platine

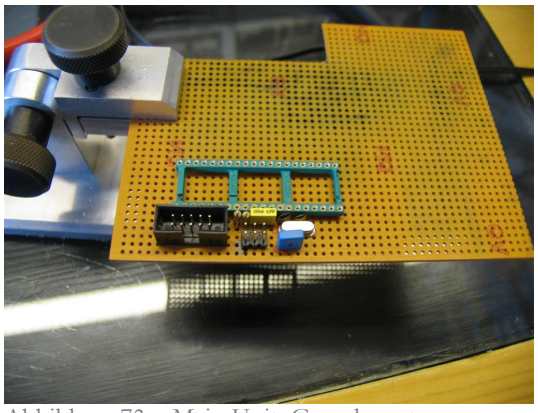


Abbildung 73 – Main Unit, Grundmontage

Zuerst wird eine Grundkonfiguration aus Stromversorgung, Takterzeugung, Reset-Schaltung und ISP-Schnittstelle aufgebaut, um diese Funktionen zu testen.

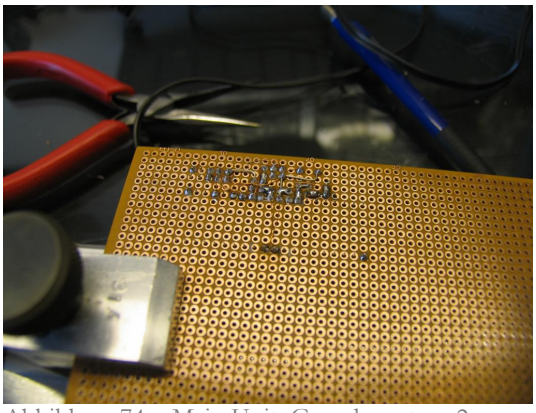


Abbildung 74 – Main Unit, Grundmontage 2

Auf der Unterseite werden die Teile Stück für Stück miteinander verbunden und bei dieser Aktion der IMMC-Connector gleich mit eingebaut, da er sich mit dem ISP-Connector die SPI-Pins des Controllers teilt.

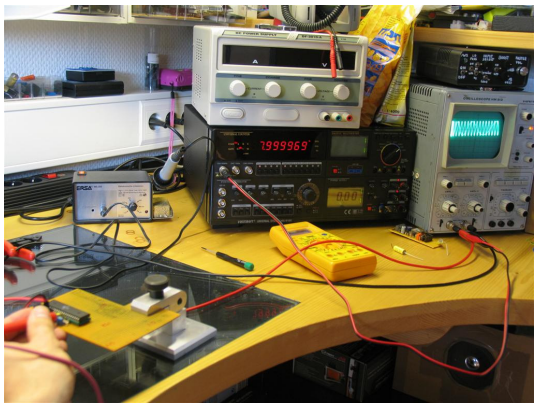


Abbildung 75 – Main Unit, Oszillatortest

Nach dem Aufbau wird zuerst die Takterzeugung getestet. Laut Frequenzzähler auf fünf Stellen genau 8 MHz. So soll es sein.

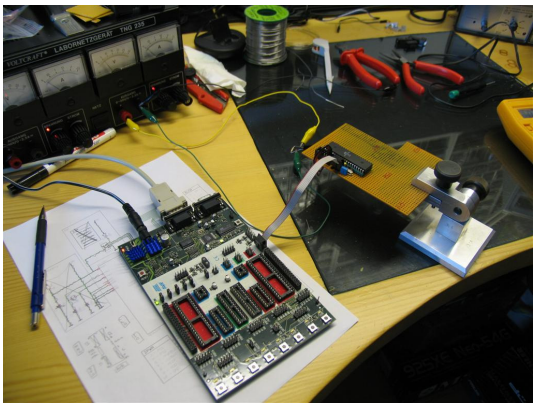


Abbildung 76 – Main Unit, ISP-Test

Danach wird die ISP-Funktionalität getestet und der Controller schon einmal probeweise im Zielsystem programmiert.

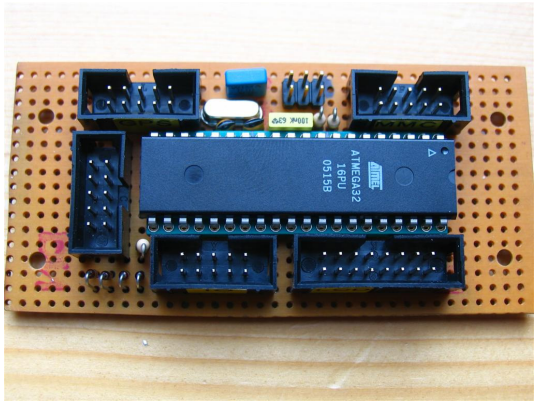


Abbildung 77 – Main Unit, Vorderansicht

Hier ein Foto nach der restlichen Fertigstellung – noch mit dem alten Mega162 zum Testen. Die einzelnen Connectors wurden von einem Beschriftungsgerät mit ihrem Namen beschriftet, um späteren Verwechslungen vorzubeugen.

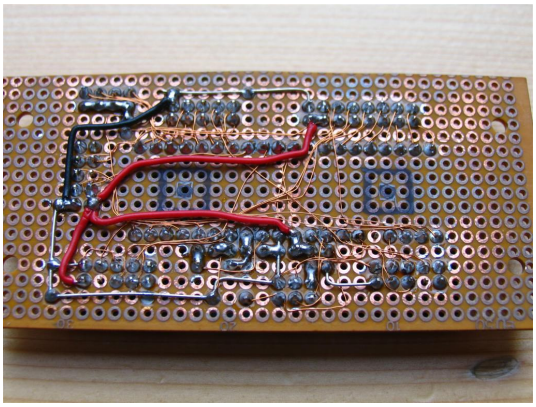


Abbildung 78 – Main Unit, Rückseite

Ansicht von unten: Auch hier gilt wie immer: Alle stromführenden Leitungen mit einem größerem Durchschnitt. Immerhin könnten hier maximal 0.6A fließen.

3.5. STROMVERBRAUCH

3.5.1. THEORETISCH BERECHNETER

Folgende Tabelle zeigt die berechneten maximalen Werte mit diversen Abhängigkeiten an:

Modul	Minimal [mA]	Maximal [mA]	Differenz [mA]	Typ
PWR	12	12		
LCD	67	239	153	Hintergrundbeleuchtung an
GPS	10	180	170	GPS-Empfänger an
IO	0	63	63	Alle LEDs an
MMC	5	25	20	MMC Zugriff
MU	9	16	7	Aktive CPU

Tabelle 37 – Maximale Stromaufnahme, berechnet

Wenn man den Wirkungsgrad des Spannungsreglers von schlechtestens 0.78 mit berücksichtigt, so ergeben sich aus den zusammengerechneten Werten einige Szenarien:

Szenario	Gesamte Stromaufnahme Module [mA]	Stromentnahme Batterie [mA]
Schlafzustand, alles deaktiviert	93	119
Normalbetrieb, GPS an	270	346
Normalbetrieb, GPS und Beleuchtung an	442	566
Normalbetrieb, GPS und Bel. an, MMC	471	604
Absolutes Maximum	525	673

Tabelle 38 – Maximale Stromaufnahme, berechnet, Szenarien

3.5.2. REAL GEMESSENER

Folgende Tabelle zeigt die real gemessenen Werte mit diversen Abhängigkeiten an:

Modul	Minimal [mA]	Maximal [mA]	Differenz [mA]	Typ
PWR	8.9	8.9		
LCD	36	128	92	Hintergrundbeleuchtung an
GPS	11	88	77	GPS-Empfänger an
IO	0.0	21	21	Alle LEDs an
MMC	0.93	2.0	1.1	MMC-Zugriff
MU		13.8		

Tabelle 39 – Maximale Stromaufnahme, gemessen

Hieraus ergeben sich ebenfalls einige Szenarien:

Szenario	Gesamte Stromaufnahme Module [mA]
Schlafzustand, alles deaktiviert	57
Normalbetrieb, GPS an	134
Normalbetrieb, GPS aus, Beleuchtung an	183
Normalbetrieb, GPS und Beleuchtung an	239
Absolutes Maximum	262

Tabelle 40 – Maximale Stromaufnahme, gemessen, Szenarien

Zum Abschluß die Tabelle der real gemessenen Werte auf der Eingangsseite:

Aufbau	Stromentnahme bei U_{IN}=7.2V [mA]	Stromentnahme bei U_{IN}=9.0V [mA]	Stromentnahme bei U_{IN}=12V [mA]	Stromentnahme bei U_{IN}=16V [mA]
GPS+LCDBG: Aus	79	63	47	36
LCDBG: An GPS: Aus	174	135	99	74
LCDBG+GPS: An	248	192	140	103

Tabelle 41 – Gemessene Eingangsstromwerte, Szenarien

Und das Ganze noch einmal graphisch:

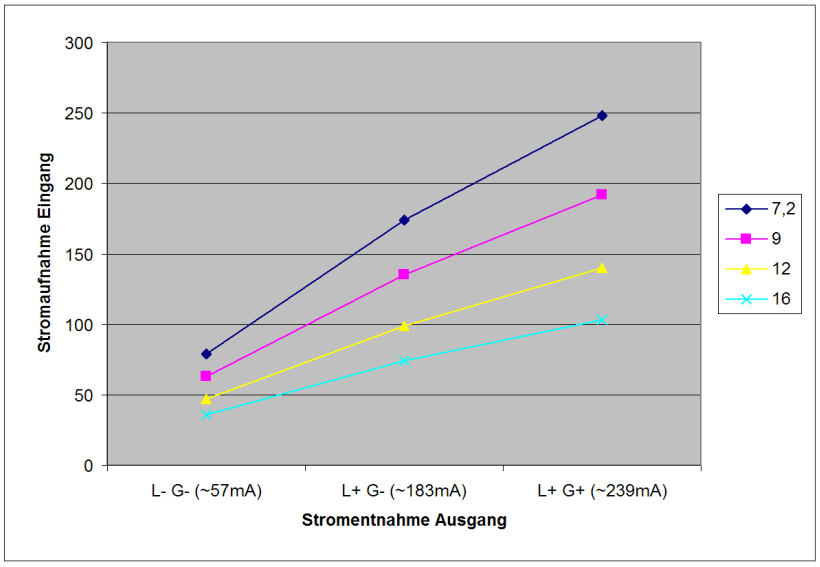


Abbildung 79 – Gemessene Eingangsstromwerte, graphisch

3.6. GEHÄUSE

Ein weiterer, nicht zu unterschätzender, Teil meiner Diplomarbeit sollte die rein mechanische Komponente der Metallbearbeitung sein. Obwohl es nur ein Prototyp ist, ging es darum, in diesem Bereich Erfahrungen zu sammeln und den Projektabschnitt erst genau zu planen und anschließend die Vorstellungen auf den verfügbaren Maschinen und Materialien umzusetzen. Deshalb wird hier bewußt ein etwas anspruchsvolleres Designergehäuse aus Aluminium und Edelstahl erstellt.

3.6.1. VERFÜGBARE MASCHINEN

Hier eine kleine Übersicht der zur Verfügung stehenden Maschinen:

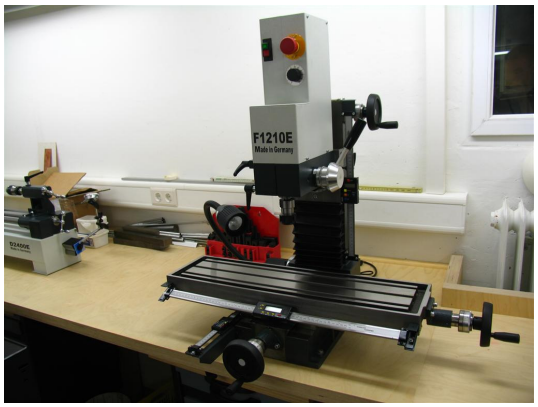


Abbildung 80 – Maschinen, Fräsmaschine

Die Fräsmaschine F1210E von Wabeco hatte am meisten zu tun: Ausfräsen und Bearbeiten der Frontplatte, der Seitenteile, der Eckpfosten und der Bodenplatte. Es gibt keinen Teil des Gehäuses, an dem sie nicht maßgeblich mitgewirkt hat.

Abbildung 81 – Maschinen, Drehmaschine

Die Drehmaschine D2400E, ebenfalls von Wabeco, bearbeitete in diesem Projekt nur die vier Eckpfosten aus Edelstahl.

Abbildung 82 – Maschinen, Bandsäge

Die Bandsäge von Optimum brachte alle Teile auf die benötigte Grundlänge.

Abbildung 83 – Maschinen, Schraubstock

Der Schraubstock rechts im Bild verrichtete sowohl die gröberen, als auch filigranere Arbeiten, wie das Einspannen der Modul-Platinen, um diese bearbeiten zu können oder auf die richtige Länge zu bekommen.

3.6.2. PLANUNG DES GEHÄUSES

Die Basis soll eine 10mm dicke Bodenplatte bilden, in der die Platinen dann eingelassen werden. Die Seitenteile bestehen aus Alu-Flachmaterial mit einer Breite von 40mm und einer Stärke von 5mm. Die Ecken werden aus 10mm dickem Edelstahl-Rundmaterial hergestellt. Für die Frontplatte steht ein 3mm dickes, eloxiertes Aluminiumblech zur Verfügung.

3.6.2.1. BODENPLATTE

Zuerst wird die Bodenplatte anhand des Platzbedarfes der einzelnen Module berechnet. Jedes Modul wird mit drei bis vier Schrauben befestigt. Der fertige Plan sie so aus:

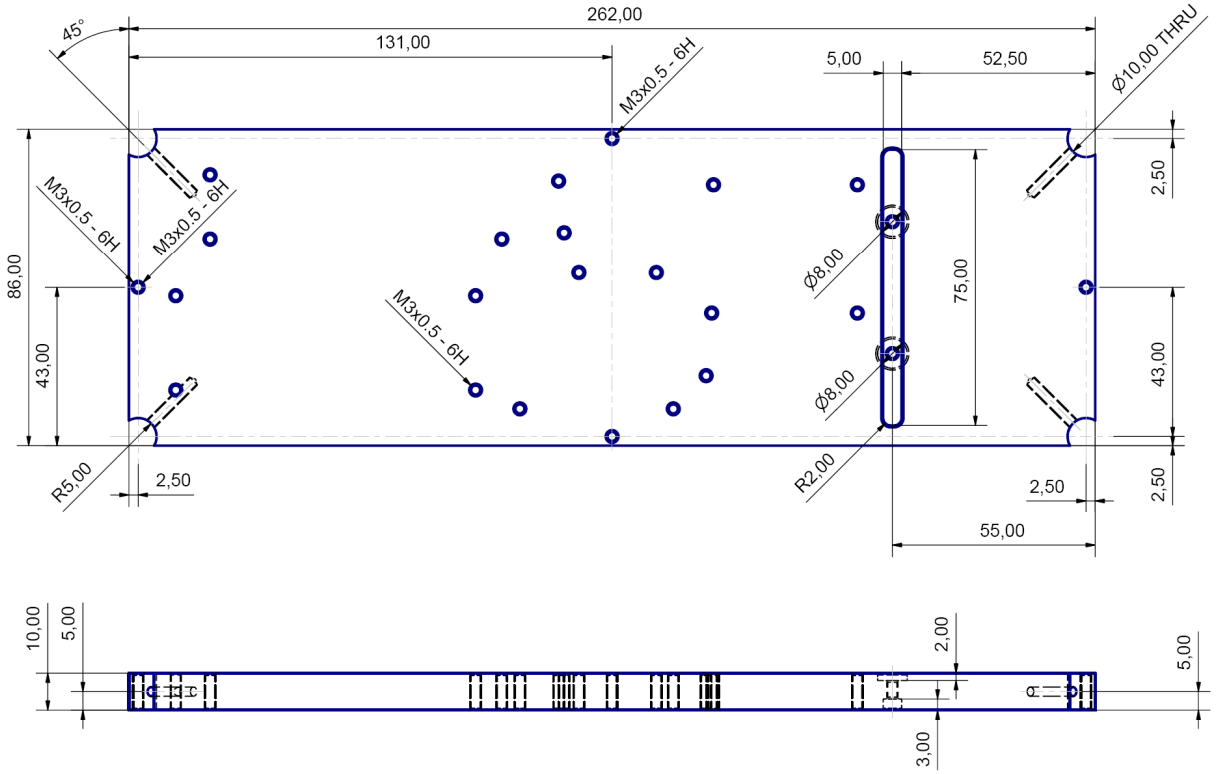


Abbildung 84 – Gehäuse, Plan: Bodenplatte

Die große Anzahl an Gewindebohrungen mag anfangs vielleicht verwundern, beim Zusammenbau erkennt man dann jedoch deren Aufgabe. An den vier Ecken erkennt man den vorgesehen Platz für die Edelstahlpfosten. Diese werden später im Winkel von 45° von der Seite angeschraubt.

In diesem Projekt sollen aus ästhetischen Gründen nur Zylinderkopf-Inbusschrauben - ebenfalls aus Edelstahl – verwendet werden.

Die vier zentrierten Bohrungen in der Mitte jeder Seite fixieren später zusätzlich die Seitenteile. In die Aussparung auf der rechten Seite kommt später eine kleine Trennwand, welche die Platinenseite von der Batteriekammer abtrennt.

3.6.2.2. ECKPFOSTEN

Ohne Umwege gleich zum Plan der Ecken:

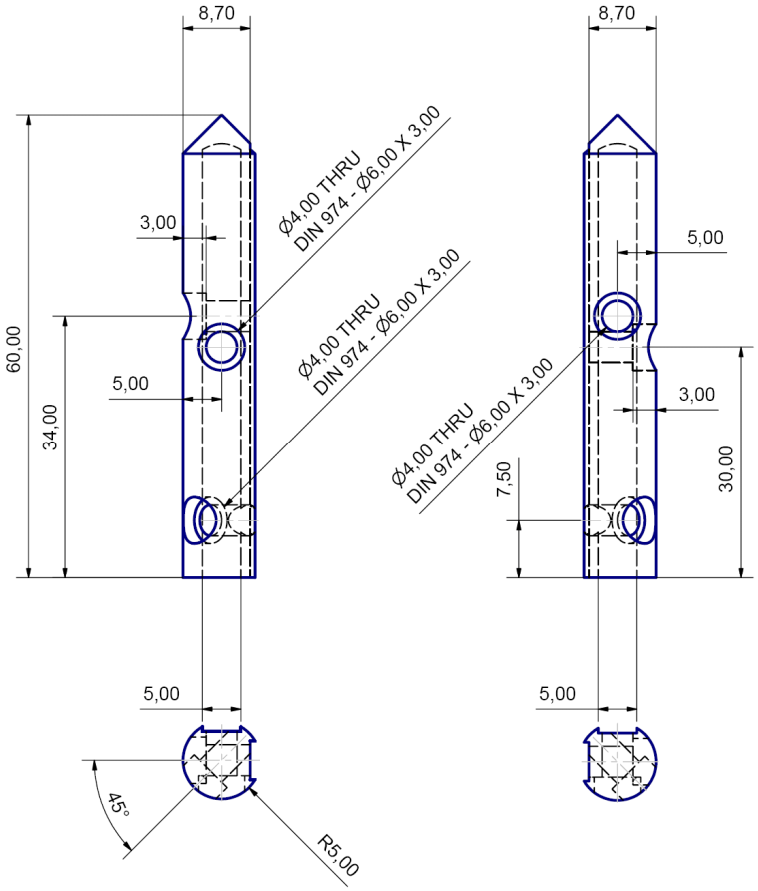


Abbildung 85 – Gehäuse, Plan: Eckpfosten

Oben werden die Pfosten im Winkel von 45° abgedreht und laufen somit konisch zu. Mit der Bodenplatte verschraubt man den Pfosten mit dem untersten Loch. Von den beiden oberen Löchern ist das untere davon für die lange Seite zuständig und das obere für die kurze Seite. Dieser Aufbau macht es notwendig, daß zwei verschiedene Typen gefertigt werden müssen – jeweils mit vertauschten oberen Löchern. Der obige Plan zeigt nur einen der beiden Typen.

3.6.2.3. SEITENTEILE

Die Seitenteile machen auf den ersten Blick einen recht umfangreichen Eindruck, sind aber vom Schwierigkeitsgrad her die einfachsten, einzig das Batteriefach wird ein wenig Geschick und Geduld verlangen:

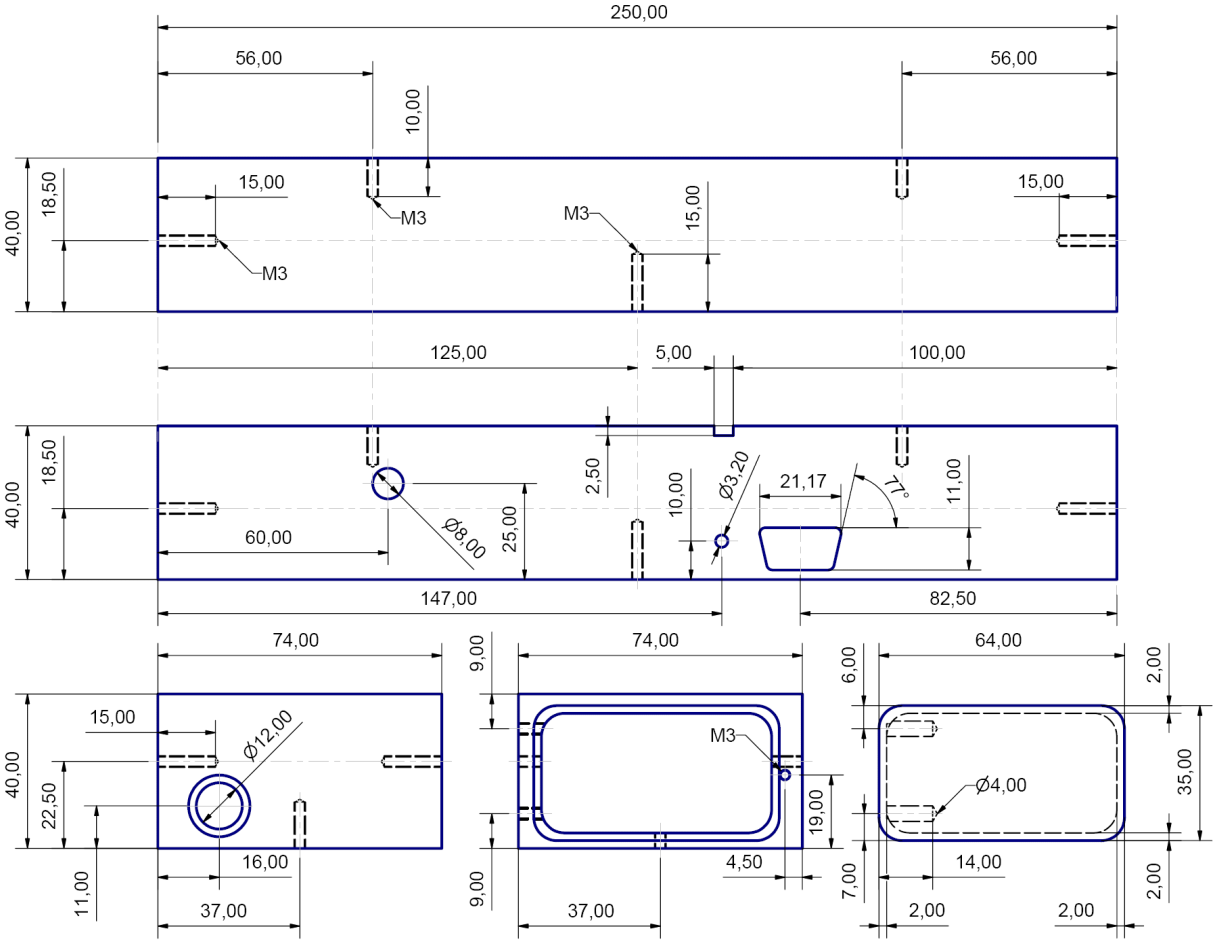


Abbildung 86 – Gehäuse, Plan: Seitenteile

Die Teile von oben nach unten: Vorderseite, Hinterseite, links außen und die letzten beiden Teile für rechts außen.

Die Vorderseite besitzt außer den Befestigungsgewinden keine weiteren Elemente. In den beiden oberen Löchern ist die Frontplatte festzuschrauben. Links und rechts die Eckpfosten und unten logischerweise die Bodenplatte.

Auf der Rückseite kommen neben den eben erwähnten Gewinden noch einige Zusatzelemente vor: In das linke 8mm Loch kommt später die DC-Buchse. Durch das kleine 3.2mm Loch kann der Benutzer mit einem dünnen Schraubendreher den Kontrast nachstellen. Gleich daneben befindet sich die Aussparung für die serielle Schnittstelle.

An der linken Seite kommt neben den Befestigungsschrauben nur der Mini-DIN-Stecker für den GPS-Empfänger zum Einsatz.

Das Batteriefach ist etwas trickreicher: Zusätzlich zu den drei Befestigungslöchern kommen auf der linken Seite noch zwei weitere Gewinde hinzu. Dort werden anschließend zwei Madenschrauben eingeschraubt und ermöglichen das Einfädeln des Batteriedeckels. In der rechten Bohrung wird später eine kurze Edelstahlschraube mit Beilagscheibe den Deckel

verschließen können. Der Batteriedeckel ist das genaue Gegenstück zu der Seitenwand und besitzt für die beiden Madenschrauben jeweils eine 4mm dicke Bohrung.

3.6.2.4. FRONTPLATTE

Die Frontplatte soll auf jeder Seite zwei Millimeter über die Seitenwände überstehen:

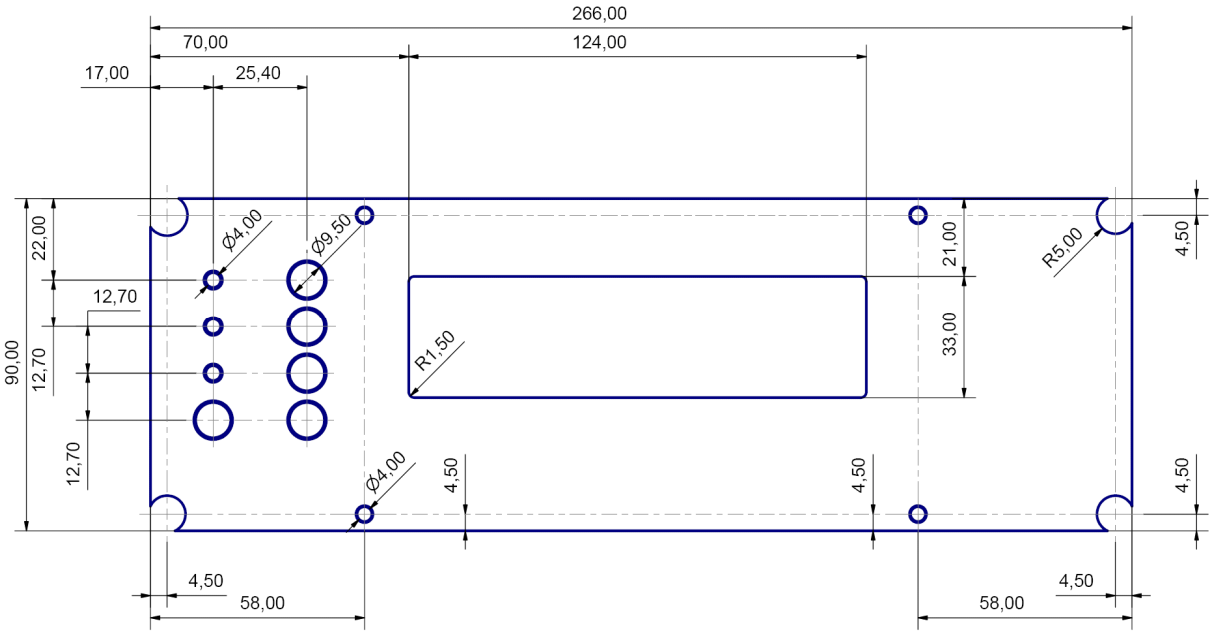


Abbildung 87 – Gehäuse, Plan: Frontplatte

Befestig wird die Platte mit den vier Löchern auf der langen Seite.

Links kommen die drei LEDs und die fünf Taster hindurch und in die Mitte das LCD-Display.

Der nächste Abschnitt bewegt sich weg von den Plänen, hin zu deren praktischen Umsetzung.

3.6.3. Herstellung der Gehäuseteile

3.6.3.1. Bodenplatte

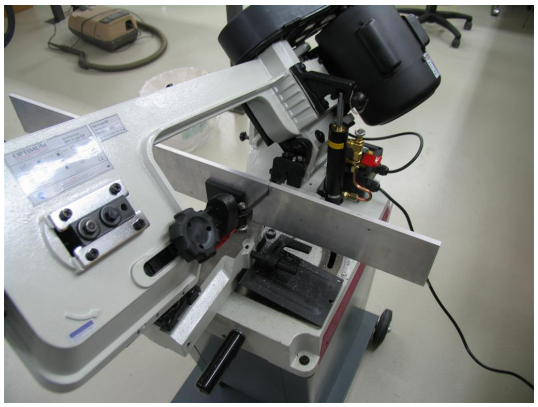


Abbildung 88 – Gehäuse, Bodenplatte 1

Zuerst habe ich die Bodenplatte erstellt. Hier sieht man, wie die Bandsäge die richtige Ziellänge vom Rohmaterial absägt.

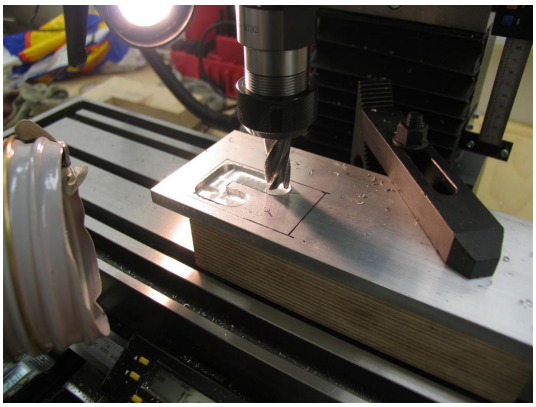


Abbildung 89 – Gehäuse, Bodenplatte 2

Da die Platinen später direkt auf die Platte geschraubt werden sollen, müssen jetzt die Vertiefungen gefräst werden. Um eine sichere Tiefe zu erreichen, habe ich 5mm, also genau die Hälfte, abgenommen. Zum Anzeichnen die Module auflegen und mit einem Stift markieren.

Abbildung 90 – Gehäuse, Bodenplatte 3

Zuerst sind die Konturen mit einem groben Schruppfräser abzutragen, anschließend die Ecken und Konturen noch einmal mit einem dünnen Fräser nachzufahren. Links unten soll die Main Unit sitzen, darüber das GPS-Modul. Rechts daneben das LCD-Modul und zum Schluß die MMC-Platine mit der Aussparung ganz nach außen.

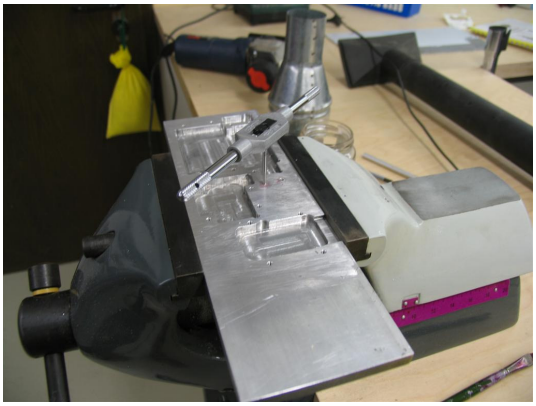


Abbildung 91 – Gehäuse, Bodenplatte 4

Wenn alle Platinen beim Auflegen passen, die Bohrungen und Gewinde zum Befestigen der Module bohren. Da es nur die Bodenplatte ist, wurden keine Sacklöcher, sondern Durchgangslöcher erstellt.

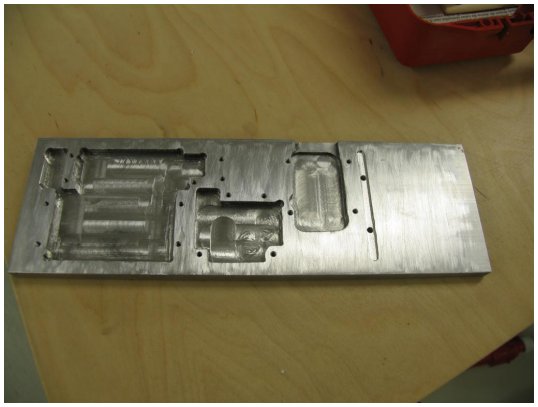


Abbildung 92 – Gehäuse, Bodenplatte 5

Nachdem alle Löcher gebohrt worden sind, die Kanten noch leicht entgraten. Zur Sicherheit habe ich noch eine durchsichtige, selbstklebende Folie in alle Vertiefungen geklebt, damit später keine Kurzschlüsse auftreten können.

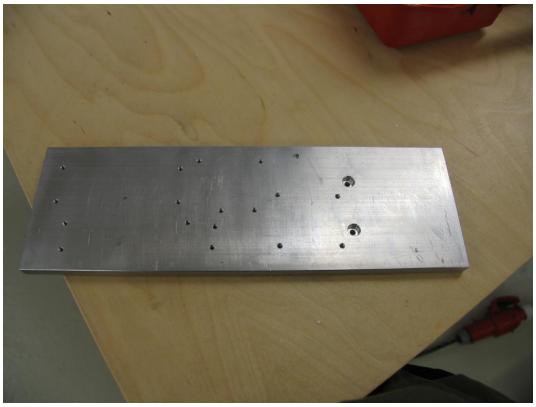


Abbildung 93 – Gehäuse, Bodenplatte 6

Hier die Ansicht von unten: Es müssen noch die Ecken für die Eckpfeiler aufgefräst werden. Die zwei größeren versenkten Löcher dienen zum Festschrauben der Trennwand für das Batteriefach.

3.6.3.2. ECKPFOSTEN

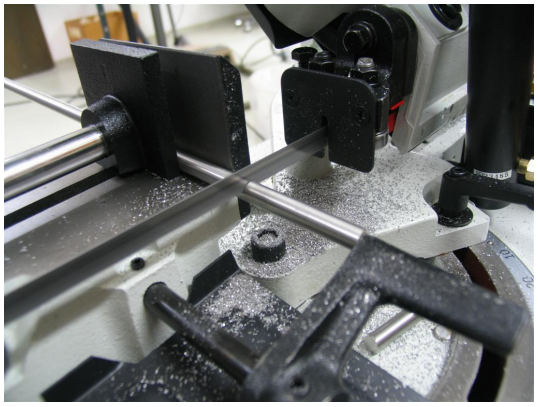


Abbildung 94 – Gehäuse, Eckpfosten 1

Der Beginn einer der vier Eckpfosten an der Bandsäge.

Abbildung 95 – Gehäuse, Eckpfosten 2

Auf der Drehbank wurde die Oberfläche behandelt und der Kegel oben gedreht.

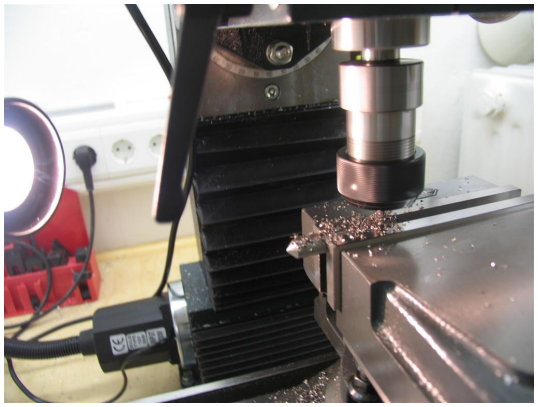


Abbildung 96 – Gehäuse, Eckpfosten 3

Um später die Seitenteile spaltlos einzusetzen, müssen zwei Nuten mit der Breite der Seitenteile, komplett durch den Edelstahlpfosten gefräst werden.

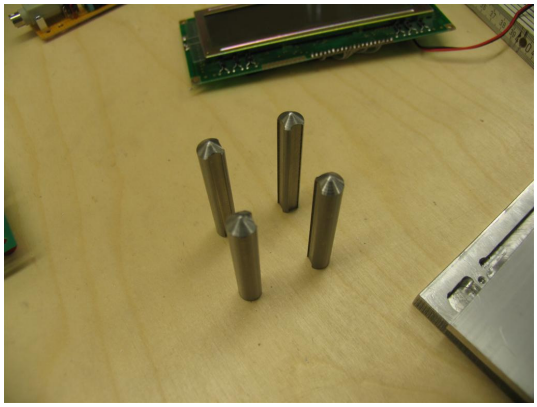


Abbildung 97 – Gehäuse, Eckpfosten 4

Nach der Fertigstellung aller Nuten hier die vier Pfosten.

Abbildung 98 – Gehäuse, Eckpfosten 5

Um das Gehäuse stabil zu befestigen, werden die Pfosten an beide Seitenteile und an die Bodenplatte geschraubt. Hier ein fertiger Eckpfosten.

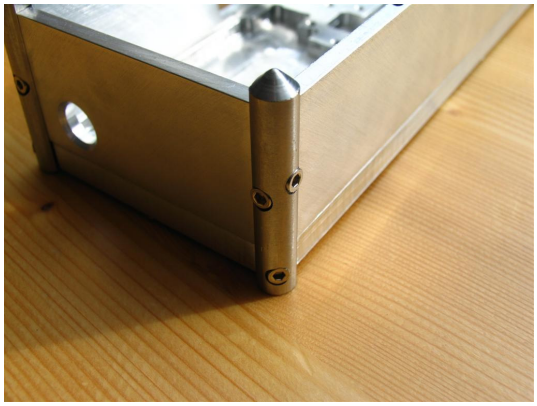


Abbildung 99 – Gehäuse, Eckpfosten 6

Ein Vorgeschmack auf die kommenden Arbeiten: Das ist ein Pfosten im fertig eingebauten Zustand mit V2A-Innensechskantschrauben.

3.6.3.3. SEITENTEILE

Abbildung 100 – Gehäuse, Seitenteile 1

Zuerst sind alle Pfosten an der Bodenplatte zu befestigen. Dadurch können die Abstände noch einmal genau überprüft werden und die Seitenteile danach angefertigt werden. Im Bild sieht man auch, daß bei der Bodenplatte die Ecken jetzt schon für die Pfosten ausgefräst sind.

Abbildung 101 – Gehäuse, Seitenteile 2

Es sind alle Seitenteile auf die richtige Länge gebracht und auch die passenden Gewinde schon gebohrt.

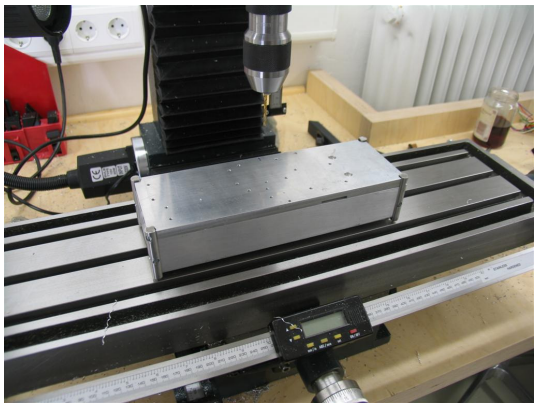


Abbildung 102 – Gehäuse, Seitenteile 3

Um die Stabilität noch einmal zu erhöhen, sind im nächsten Schritt die vier Seitenteile mit Flachkopfschrauben an der Bodenplatte zu befestigen. Da die Bodenplatte nicht direkt auf dem Untergrund aufliegt sondern nur die vier Pfosten, und diese jeweils nur im Winkel von 45° festgeschraubt sind, treten hier bei harten Stößen sicher hohe Scherkräfte auf. Die neuen Schrauben müßten diese gut auffangen.

Abbildung 103 – Gehäuse, Seitenteile 4

Danach geht es an die Bearbeitung der einzelnen Seitenteile. Hier die schwierigste Aufgabe, das Batteriefach. Die beiden Teile sind so bearbeitet, daß sie exakt ineinander passen. Durch die zwei Madenschrauben links kann man die Blende einfach reinklappen. Auf der rechten Seite muß noch ein Gewinde gebohrt werden, damit man die eingeklappte Blende auch befestigen kann.

Abbildung 104 – Gehäuse, Seitenteile 5

Hier alle fertigen Seitenteile auf einmal: Ganz oben das Batteriefach, darunter die spätere linke Geräteseite mit dem Anschluß des GPS-Empfängers an den Mini-DIN Stecker. Darunter die Frontplatte ohne jegliche Einbauteile. Dann die Rückplatte mit der Aussparung für die serielle Schnittstelle, daneben eine Bohrung, um mit einem kleinen ⟶

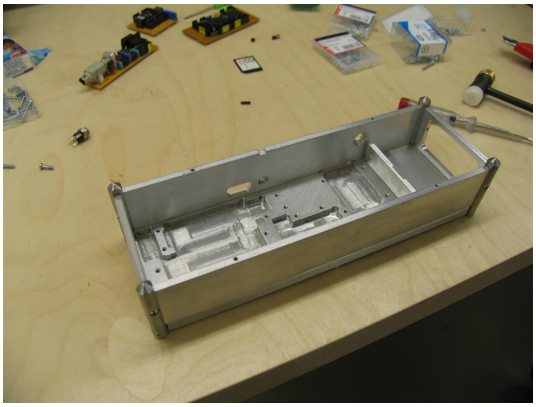


Abbildung 105 – Gehäuse, Seitenteile 6

Schraubendreher den LCD-Kontrast einzustellen. Auf der anderen Seite kommt später die DC-Eingangsbuchse zur Stromversorgung hinein. Ganz unten liegt die Trennwand zwischen Platinen und dem Batteriefach. Auf dem rechten Bild sieht man das Gehäuse soweit zusammengebaut.

3.6.3.4. FRONTPLATTE

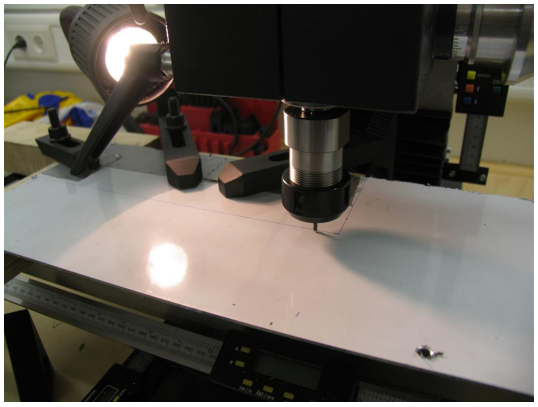


Abbildung 106 – Gehäuse, Frontplatte 1

Im Gegensatz zum restlichen Gehäuse, das bis auf die Eckpfosten aus normalem Aluminium besteht, ist die Frontplatte aus eloxiertem Aluminium, welches aber noch mit einer Schutzfolie auf beiden Seiten bedeckt ist. Zuerst wird die Grundform aus dem großen Rohstück gefräst.

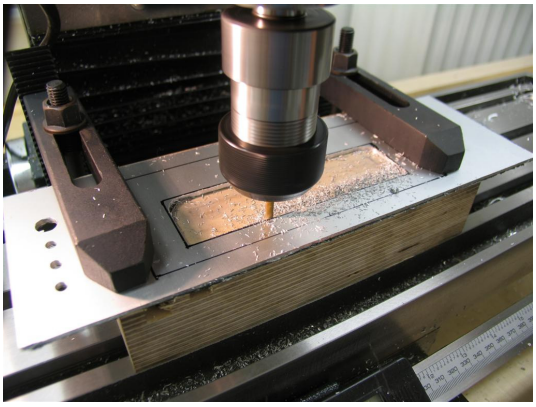


Abbildung 107 – Gehäuse, Frontplatte 2

Anschließend werden alle Durchlässe für die späteren Bedienelemente gefertigt. Links kommen die LEDs hinein, rechts daneben verdeckt die Spannpratze die Löcher für die Tasten. Das Bild zeigt das Fräsen der Aussparung für das LCD-Display.

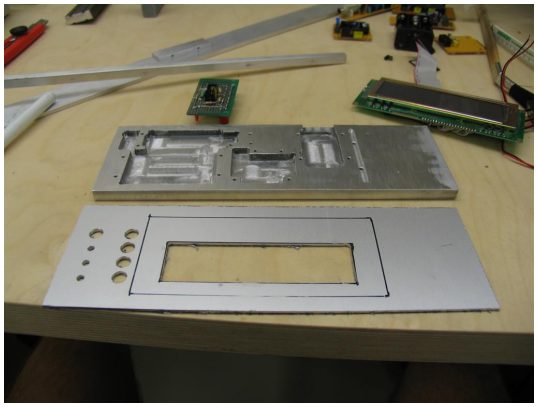


Abbildung 108 – Gehäuse, Frontplatte 3

Hier sieht man die mit den Aussparungen fertige Frontplatte. Zur Sicherheit legt man noch einmal das IO-Modul und das LCD-Display auf und prüft, ob von vorne alles paßt, sichtbar ist und ob sich die Taster auch ohne Klemmen drücken lassen.

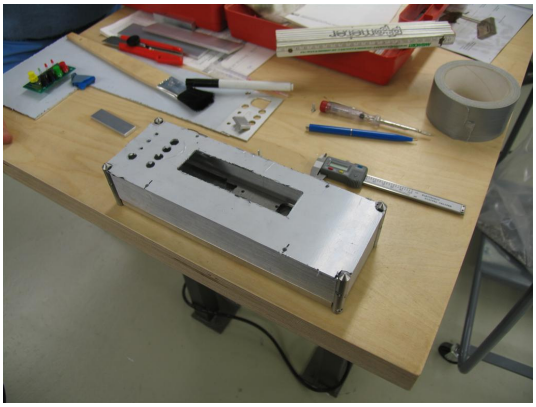


Abbildung 109 – Gehäuse, Frontplatte 4

Anschließend die Löcher für die Eckpfosten ausfräsen und probehalber auf den Unterbau auflegen.

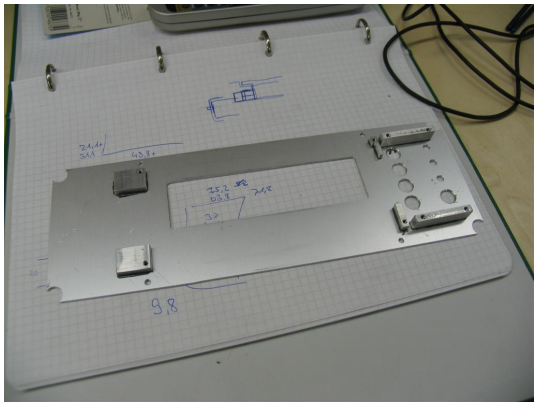


Abbildung 110 – Gehäuse, Frontplatte 5

Nachdem alles paßt, klebt man mit einem speziellen 2-Komponenten-Metallkleber die angefertigten Halterungen für das IO-Modul und das LCD-Display. Es ist unbedingt darauf zu achten, daß die Bohrungen an den richtigen Positionen sind, da ein nachträgliches Ablösen und Neuanbringen extrem aufwendig ist. Um eine höhere Endfestigkeit zu erreichen, habe ich die Platte bei ca. 80°C für etwa eine Stunde aushärten lassen.

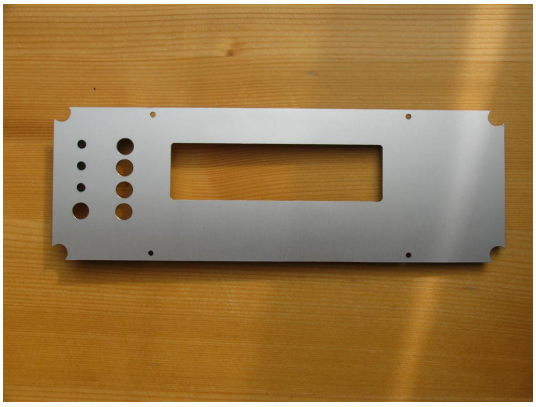


Abbildung 111 – Gehäuse, Frontplatte 6

Wenn der Kleber fest ist, kann die Platte fertiggestellt, alle Kanten noch einmal leicht entgratet und die Schutzfolie abgezogen werden.

3.6.4. Einbau der Module und der Zusammenbau

Nachdem die einzelnen Gehäuseteile nun gefertigt und aufgebaut sind, stehen jetzt der Einbau der einzelnen Hardwaremodule und deren Verbindungen untereinander an:

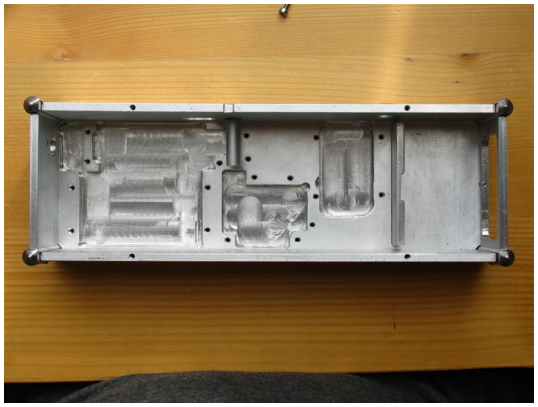


Abbildung 112 – Gehäusezusammenbau 1

Hier der fertig zusammengebaute Unterbau des Gehäuses. Die einzelnen Gewindelöcher sind gut zu erkennen.

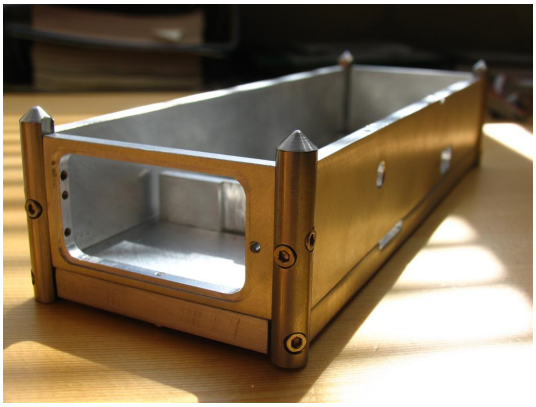


Abbildung 113 – Gehäusezusammenbau 2

Die Seitenansicht auf das Batteriefach: Hier passen die 6x1.5V AA-Batterien perfekt hinein.

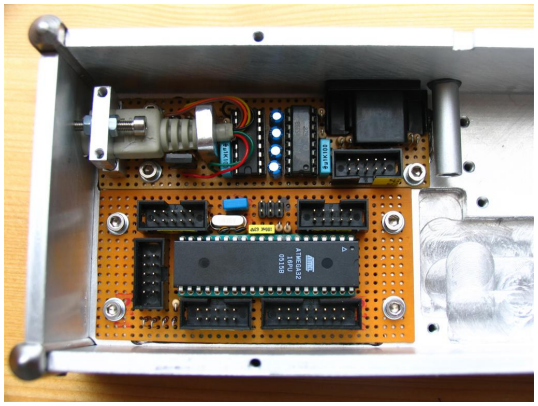


Abbildung 114 – Gehäusezusammenbau 3

Zuerst wird die GPS-Platine oben eingesetzt und festgeschraubt, danach unten die Main-Unit.

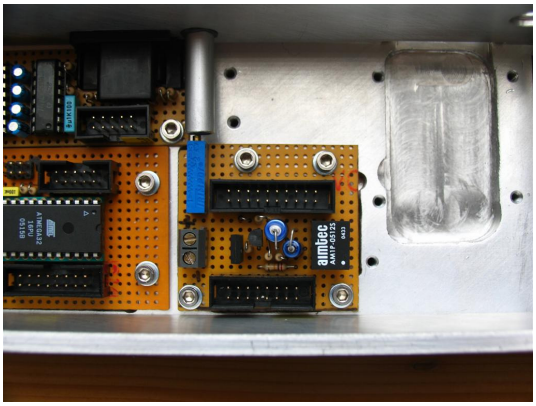


Abbildung 115 – Gehäusezusammenbau 4

Anschließend ist das LCD-Modul an der Reihe: Durch das Rohr oben kann man bei Bedarf den Kontrast nachregeln.

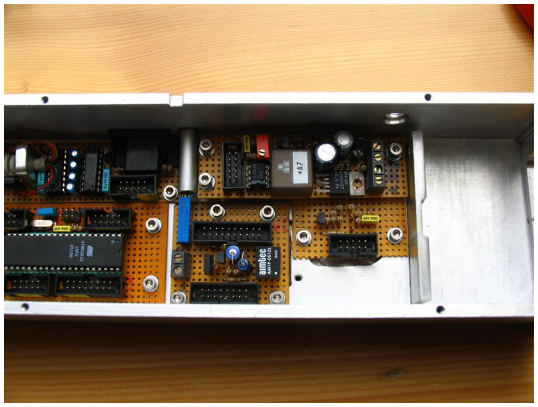


Abbildung 116 – Gehäusezusammenbau 5

Jetzt befestigt man das MMC-Modul. Quer darüber kommt das Powermodul, das mit ein paar Millimetern Abstand zum darunter liegenden Modul montiert wird.

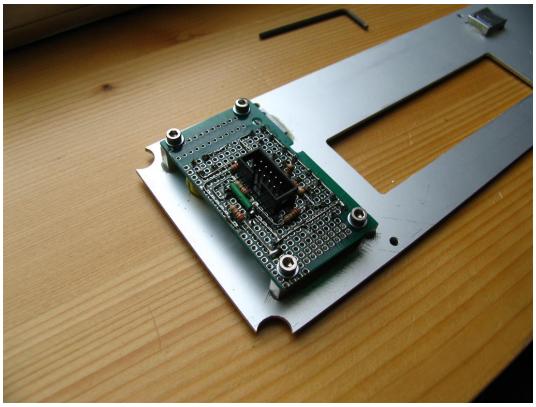


Abbildung 117 – Gehäusezusammenbau 6

Auf die Frontplatte kommt zuerst die IO-Platine. Wichtig ist, daß sich die Schalter nicht verkanten. Außerdem müssen die LEDs in die Fassungen eingepaßt werden.

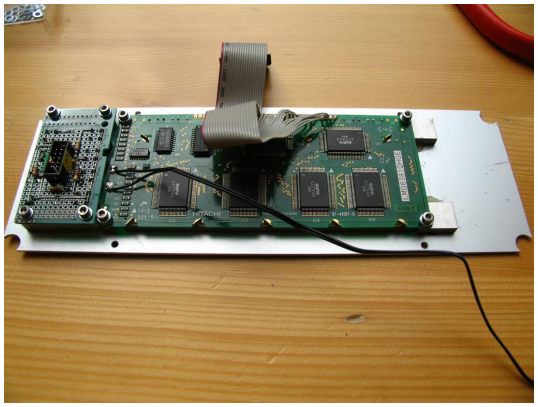


Abbildung 118 – Gehäusezusammenbau 7

Das Display kommt direkt neben die IO-Platine. Das ist das einzige Modul, auf dem die Anschlußleitungen fest verlötet sind.

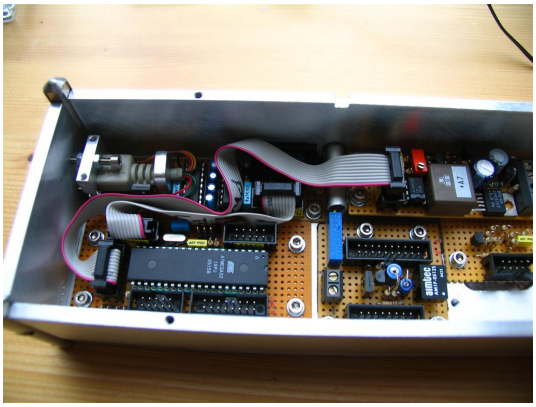


Abbildung 119 – Gehäusezusammenbau 8

Wenn alle Platinen verschraubt sind, fängt die Verkabelung untereinander an. Hier ist möglichst gutes Verlegen gefragt, denn der Platz ist sehr knapp.

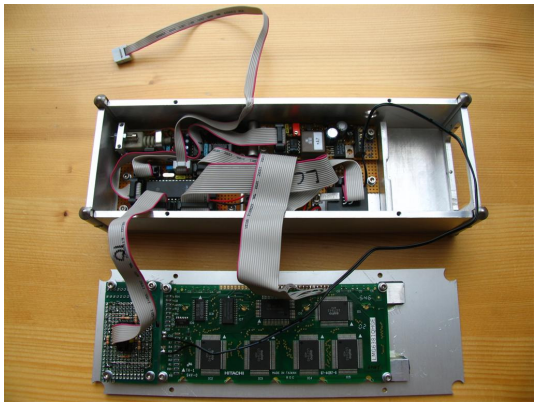


Abbildung 120 – Gehäusezusammenbau 9

Hier sind alle Verbindungsleitungen bereits installiert. Oben schaut der ISP-Stecker zum Programmieren heraus. In der finalen Version wird dieses Kabel entfernt.

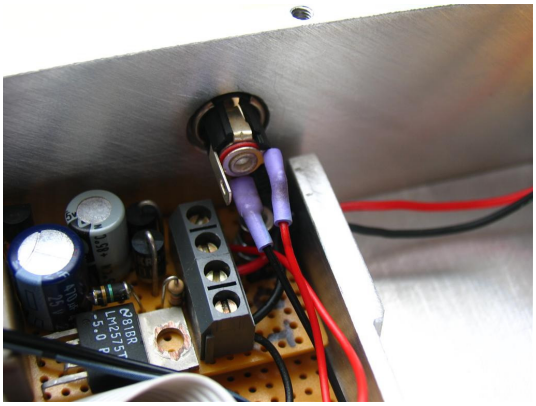


Abbildung 121 – Gehäusezusammenbau 10

Zuletzt noch den 9V-Stecker und die DC-Buchse montieren und an das Powermodul anschließen.

Abbildung 122 – Gehäusezusammenbau 11

Das fertige Gerät von hinten …

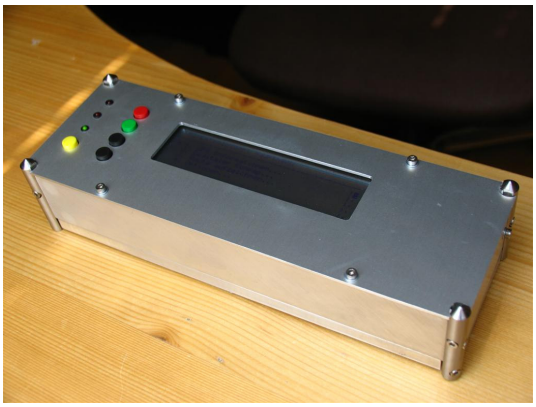


Abbildung 123 – Gehäusezusammenbau 12

… und von vorne.

4. AVR SOFTWARE

4.1. VERWENDETE SOFTWARE

Im folgenden Abschnitt soll kurz die in dieser Diplomarbeit verwendete Software beschrieben werden.

4.1.1. IDE - DIE PROGRAMMIERUMGEBUNG

Zur Verwaltung und dem Editieren des Quelltextes wird das freie Programm *Programmers Notepad*[24], eine IDE-Umgebung, eingesetzt. Es unterstützt eine Vielzahl von Programmiersprachen und beherrscht *Syntax-Highlighting*. Zusätzlich können Tastenkürzeln Befehlsabläufe oder Makros zugewiesen werden. Damit kann das Kompilieren, Bereinigen und Überspielen der Software in das Zielsystem automatisiert werden. Selbst die Ausgaben des Compilers werden in der IDE in ein Ausgabefenster umgeleitet und bei einem Klick auf die Zeile einer entsprechenden Warnung oder Fehlermeldung durch einen Parser direkt die betreffende Quelldatei nebst fehlerhafter Zeile geöffnet und markiert.

Es folgen ein paar Screenshots zur besseren Vorstellung der Programmierumgebung:

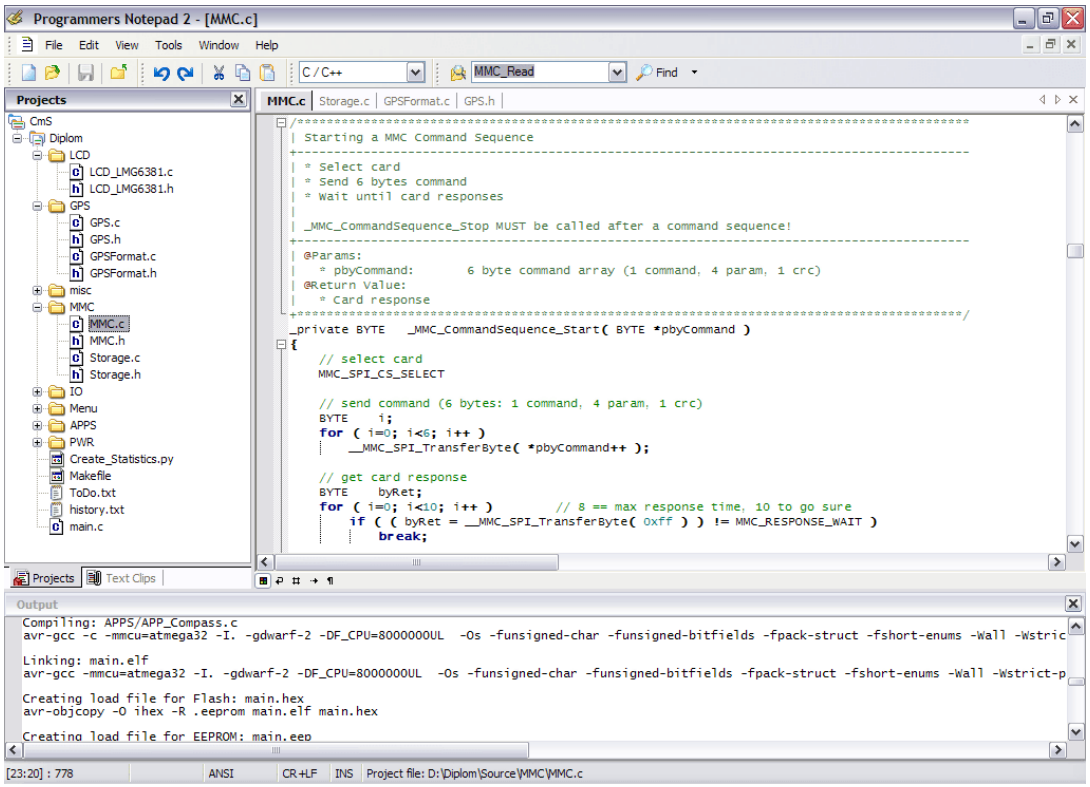


Abbildung 124 – Screenshot Programm Notepad 2

[24] Homepage: http://www.pnotepad.org/

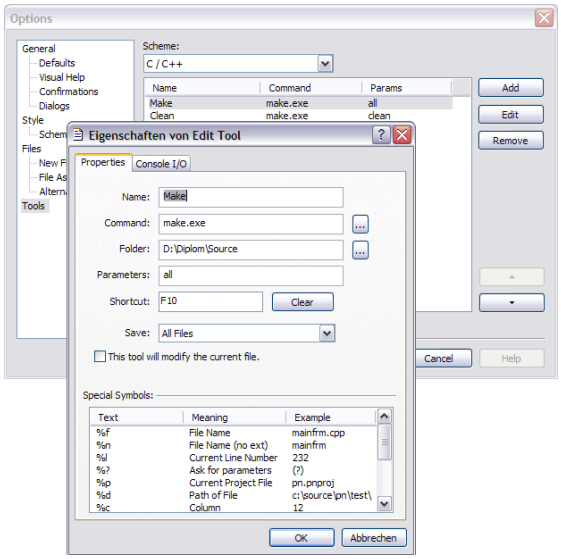


Abbildung 125 – Screenshot PN2, Tools und Makros

Hier können verschiedene Menüeinträge, die auch über Shortcuts erreichbar sind, erstellt werden. Unter anderem kann angegeben werden, welcher Befehl mit definierbaren Parametern aufgerufen wird, ob die Quelldateien vorher gespeichert werden sollen, die Ausgabe überwacht oder dafür ein neues Fenster geöffnet werden soll.

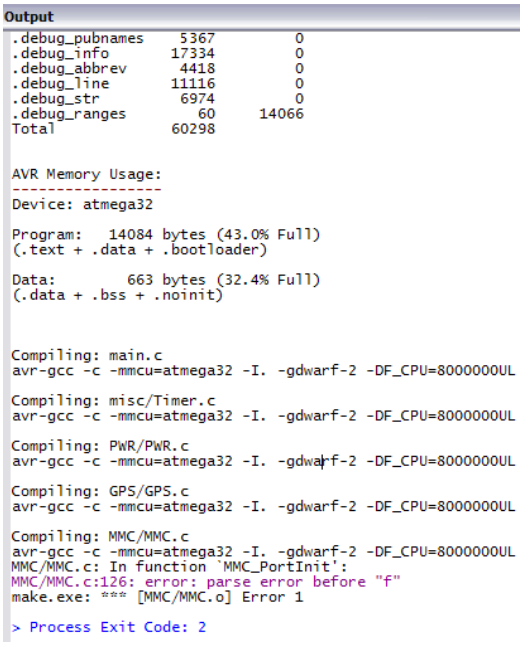


Abbildung 126 – Screenshot PN2, Output Window

Im Output-Fenster wird die Ausgabe des ausgeführten Befehls formatiert wiedergegeben. Eventuelle Fehler werden markiert, hier im Beispiel violett. Nach einem Klick darauf wird die Datei geöffnet und die entsprechende Zeile ausgewählt.

4.1.2. Compiler avr-gcc aus WinAVR

Im Projekt wird das freie Paket WinAVR[25] eingesetzt. Es besteht aus einigen Softwarepaketen, die bei Interesse in [WINAVR] genau beschrieben werden. Die wichtigsten sind:

- **GNU Binutils 2.15 + coff-avr-patch (20030831)**
  Ausführbare Tools für die ARV-Plattform (Assembler, Linker, usw.).

- **GNU Make**
  Das bekannte Make-Tool.

- **GNU Compiler Collection (GCC) 3.4.3**
  Die Compiler der Sprachen C und C++ für die AVR-Plattform.

- **avr-libc 1.2.3**
  C Standard Library für AVRs.

- **avrdude 4.4.0cvs**
  Eine Open-Source Programmiersoftware.

[25] Homepages: http://sourceforge.net/projects/winavr und http://winavr.sourceforge.net/

- **avrdude-gui 0.2.0**
  Die passende GUI dazu.

- **GNU Debugger (GDB) 6.1**
  Der Standard Debugger für AVRs.

- **Insight 6.1**
  Eine GUI für den GDB.

- **simulavr 0.1.2.1**
  Wird zusammen mit GDB zur Simulation benutzt.

- **Programmers Notepad 2.0.5.32**
  IDE, oben schon genauer beschrieben.

In diesem Projekt werden nur die ersten vier Tools aus dem Paket verwendet.

4.1.3. ISP Programmierung des Controllers

Um den Mikrocontroller zu programmieren, ist ein Programmiergerät notwendig. Dieses wird von dem PC über die serielle Schnittstelle gesteuert und muß an das Zielsystem über den ISP-Verbinder verbunden werden. Der Controller kann somit direkt im Zielsystem programmiert werden. Hierfür wird das ISP-Interface benutzt.

Hier noch einmal der Hardwareaufbau:

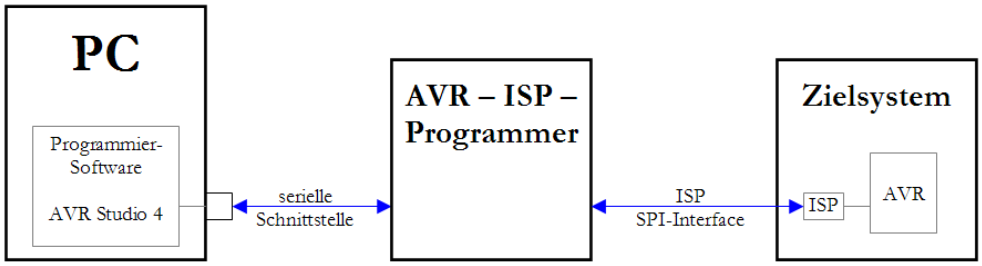


Abbildung 127 – Hardwareaufbau ISP Programmierung

Als AVR-ISP-Programmer wird das im Hardwarekapitel schon ausführlich beschriebene STK500 verwendet, das unter anderem auch das ISP beherrscht.

Um mit dem STK500 den Zielcontroller programmieren zu können, wird in diesem Projekt das *AVR-Studio* von Atmel in der Version 4.11 verwendet.

In dem Programmierdialog muß die kompilierte, binäre Datei mit der `.HEX`-Dateiendung ausgewählt werden. Nach dem Programmierstart wird der Zielcontroller gelöscht, neu programmiert, wieder ausgelesen und damit verifiziert.

4.1.4. Schaltplanentwicklung

Zum Entwickeln der Schaltpläne wird *Target 3001! V11* (http://www.ibfriedrich.com) eingesetzt. Mit diesem Programm können sowohl Schaltpläne gezeichnet, als auch die Platinen dazu

entwickelt werden. Bei bestimmten, kostenpflichtigen Versionen ist zusätzlich ein Simulator für analoge, digitale und gemischte Schaltungen vorhanden.

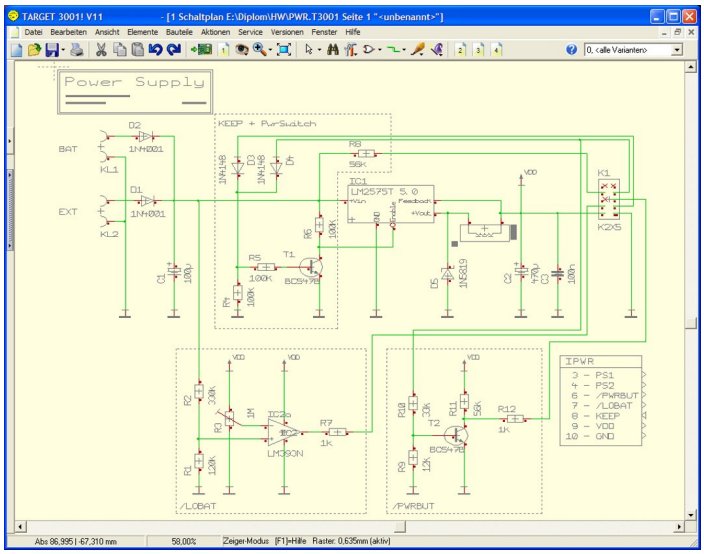


Abbildung 128 – Screenshot Target 3001! V11

4.1.5. DOKUMENTATION

Zum Erstellen dieses Dokuments wird *Microsoft Word 2003* eingesetzt. Im Gegensatz zu anderen Meinungen läuft dieses Produkt nämlich extrem stabil, ist sehr vielseitig und flexibel und bietet teilweise weitreichendere Möglichkeiten und Unterstützung für den Autor, als viele Konkurrenzprodukte, speziell im Bereich der Automatisierung und (Makro-)Programmierung. Mit einem entsprechenden Monitor lassen sich problemlos zwei Seiten und die Formatauswahl nebeneinander anzeigen und bearbeiten:

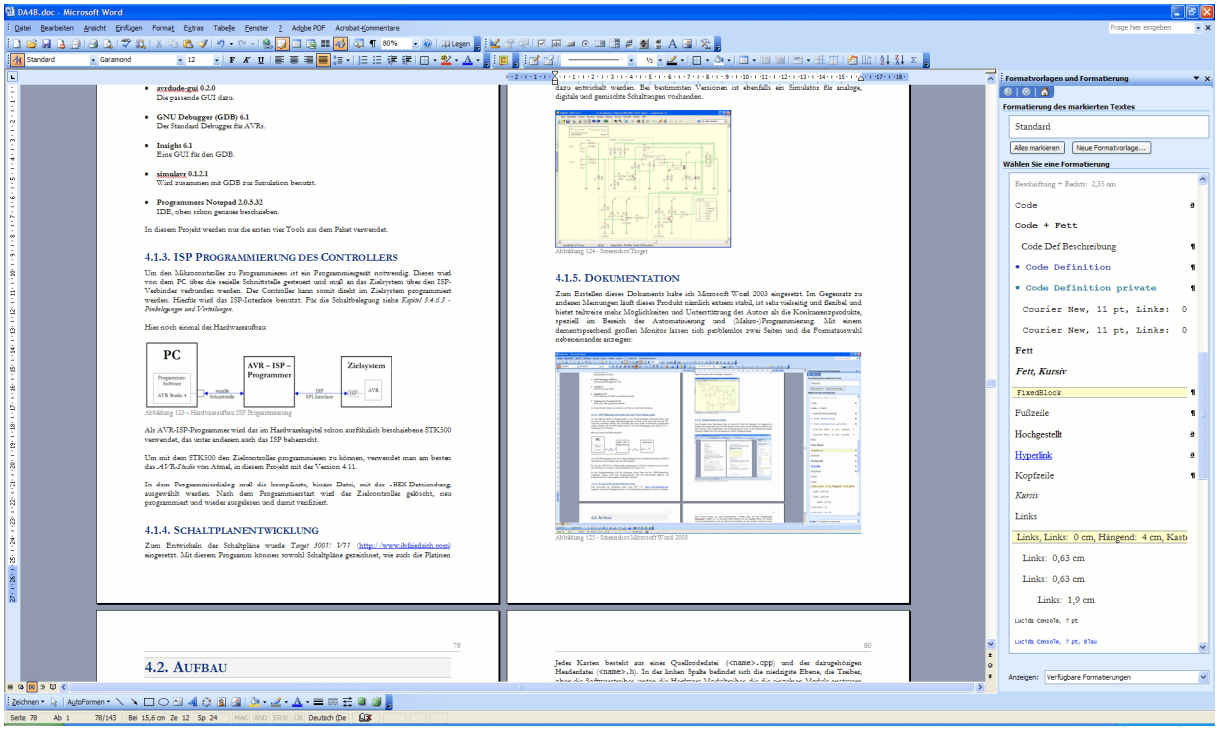


Abbildung 129 – Screenshot Microsoft Word 2003

4.1.6. GEHÄUSEPLANUNG

Um die technischen Zeichnungen zu erstellen, stand mir für einen Zeitraum das Programm *Autodesk Inventor 6* von Autodesk zur Verfügung. Dieses Produkt läßt sich an den Möglichkeiten und der übersichtlichen und intuitiven Bedienung nur noch schwer übertreffen, setzt allerdings einen dementsprechend potenten Rechner voraus.

Im Groben existieren zwei Modi: Eine Skizzenansicht zum Erstellen zweidimensionaler CAD-Zeichnungen und dann die Teileansicht in 3D. In dieser 3D-Ansicht können beliebige, weitere Ebenen definiert werden, die wiederum Skizzen darstellen. Somit kann man sehr einfach weitere Elemente dem Bauteil hinzufügen oder entfernen (Bohrungen, Fräsungen usw.).

Hier ein Screenshot beim Erstellen der Seitenteile. Oben erkennt man noch die 2D-Skizze mit den Abmessungen. Zurück in der 3D-Ansicht wurde nun Bohren ausgewählt. Die rote Vorschau visualisiert das zu erwartende Ergebnis. Im mittleren Bauteil wurden die Bohrungen schon ausgeführt:

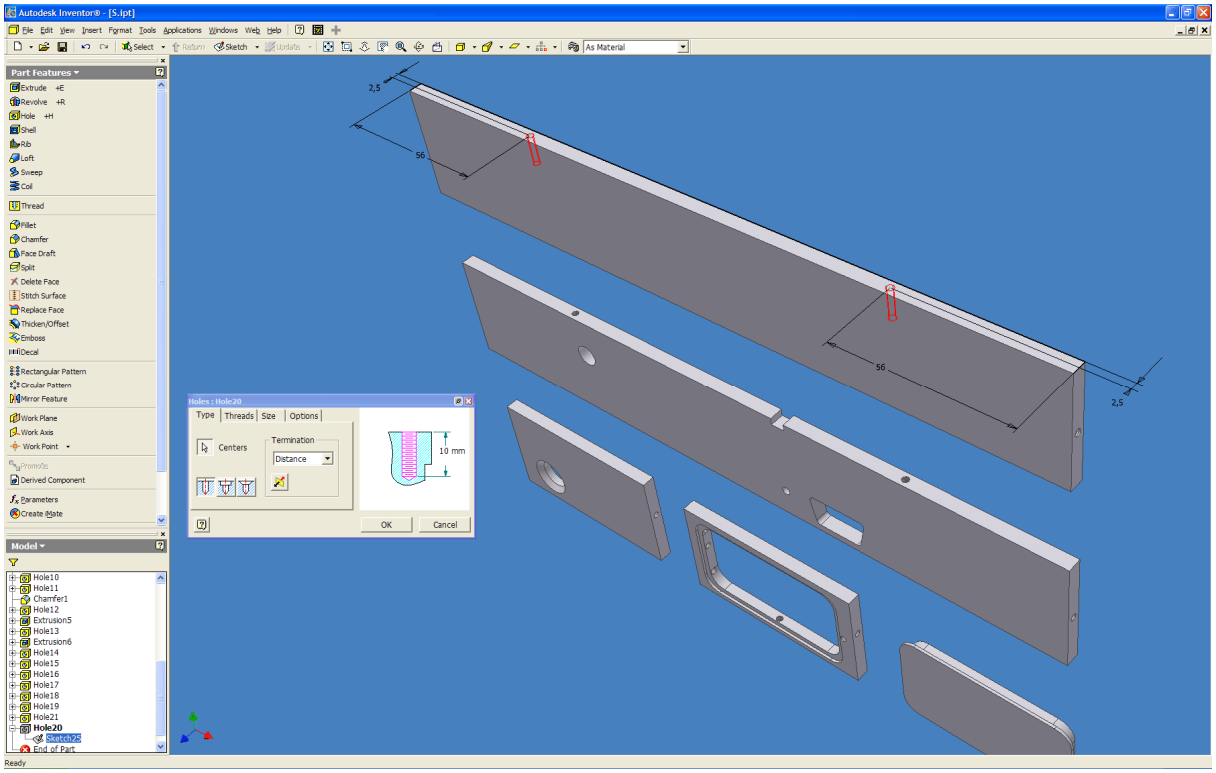


Abbildung 130 – Schreenshot Autodesk Inventor 6

Neben weiterer 3D-Aktionen kann nun jede beliebige Fläche als neue Skizze ausgewählt und somit wieder eine Zeichnung mit 2D-Formen erstellen werden. Mit dieser Arbeitsweise kann man auch sehr einfach den späteren Ablauf beim Erstellen der Teile an den Maschinen planen und eventuelle Probleme frühzeitig erkennen und beseitigen.

4.2. Aufbau

4.2.1. Projektübersicht

Um eine grobe Übersicht zu bekommen und das Projekt und dessen Abhängigkeiten in den folgenden Abschnitten besser verstehen zu können, hier erst einmal ein Diagramm mit dem kompletten Aufbau der Quellcodedateien und deren Abhängigkeiten untereinander:

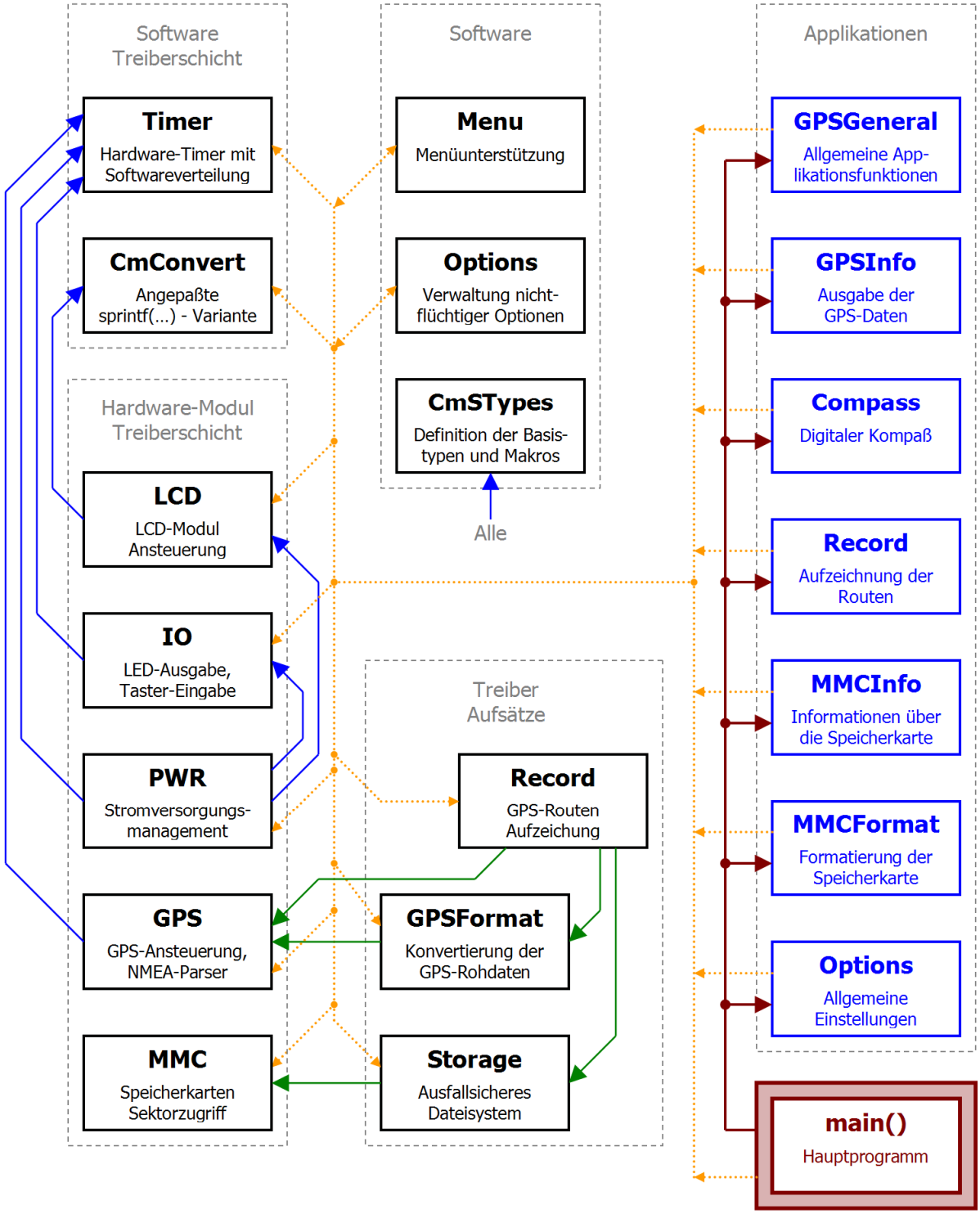


Abbildung 131 – Softwareaufbau und -abhängigkeiten

Jedes Element besteht aus einer Quellcodedatei (`<name>.cpp`) und der dazugehörigen Headerdatei (`<name>.h`). In der linken Spalte befindet sich die unterste Ebene – die Treiber – oben die Softwaretreiber und unten die Hardware-Modultreiber, die die einzelnen Module ansteuern und verwalten. Die blauen Pfeile zeigen die Abhängigkeiten unter den Hardwaremodulen an. Die Pfeilspitze zeigt immer auf den verwendeten Treiber hin. Als nächstes kommen unten mittig die Treiberaufsätze, die auf den Treibermodulen aufsetzen und dem Treiber dadurch erweitern. Die Abhängigkeiten unter diesen Aufsätzen stellen grüne Pfeile dar. Oben mittig befinden sich die allgemeinen Softwareteile. Auf der rechten Seite stehen die Applikationen des Gerätes in blau. Diese werden vom Hauptprogramm aus aufgerufen. Die orange gestrichelte Pfeilstruktur stellt einen Bus dar, um die Darstellung der Abhängigkeiten hier zu vereinfachen. Geht ein Pfeil aus dem Bus hinaus, so verwendet der Bus dieses Modul. Geht allerdings ein Pfeil in den Bus hinein, so verwendet das Modul den Bus. Eine Applikation kann also auf fast alle Treiber und Softwarebausteine zugreifen.

4.2.2. DATEILISTE

In der folgenden Abbildung sind alle verwendeten Quelldateien aufgelistet:

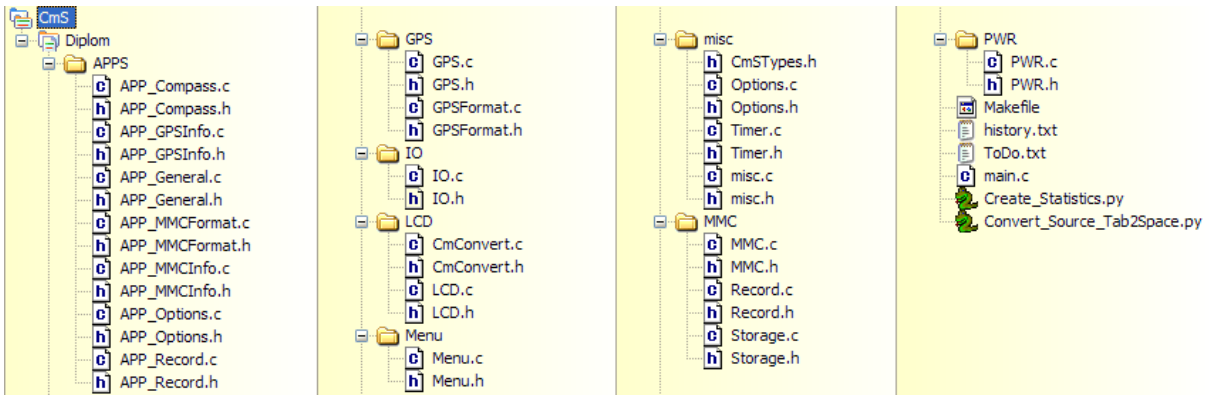


Abbildung 132 – Sourcedateienliste

4.2.3. AUFBAU DES KAPITELS

Das restliche Kapitel behandelt die einzelnen Teile Stück für Stück, beschreibt deren Funktion, die Benutzung, die aufgetretenen Probleme und alles sonst Wissenswerte zu den verschiedenen Modulen.

Jeder Abschnitt zu den Softwaredateien wird in folgende Teile untergliedert:

- **Aufgabe**
  Eine kurze Beschreibung der zu bewältigenden Aufgabe der Software.

- **Hardwarekonfiguration**
  Beschreibung der notwendigen Konfiguration der Hardware (nur bei Treibern).

- **Interface**
  Genaue Auflistung des Interfaces.

- **Aufbau des Modultreibers**
  Grobe Erklärung des Aufbaus und der Abläufe im Treiber.

- **Implementierung kritischer Stellen**
  Detaillierte Beschreibung kritischer Stellen in der Implementierung.

- **Beispielverwendung und Tests**
  Eine kleine Bespielanwendung, Ergebnisse und Tests mit dem Modul.

- **Berechnungen, Analysen und Statistiken**
  Wie der Titel schon sagt, befinden sich hier Berechnungen, Analysen und Statistiken rund um das Modul.

Zuerst aber zu den wichtigen Formatregeln der erstellten Software:

4.2.4. FORMATE UND REGELN DER SOFTWARETEILE

Die Modultreiber für die fünf Module LCD, GPS, MMC, IO und PWR und alle anderen Softwareteile haben zumindest ein grobes, einheitliches Format, an das sie sich halten müssen:

- **Funktionsnamen**
  Alle Funktionsnamen beginnen mit den Buchstabenkürzel des Treibers, gefolgt von einem Unterstrich, also zum Beispiel:

  ```
  LCD_MeineFunktion();
  ```

- **Globale, exportierte Variablennamen**
  Globale Variablen beginnen mit einem `g_`, dann die Buchstabenkürzel, wieder ein Unterstrich, der Typkurzbezeichnung und letztendlich ein aussagekräftiger Name:

  ```
  g_LCD_byMyByte
  ```

- **Defines**
  Auch hier gilt: Zuerst die Kürzel, dann meistens noch die Gruppe und anschließend der Name:

  ```
  LCD_DATA_PORT
  ```

- **Notwendige Funktionen**
  Die Treiber müssen zusätzlich folgende Funktionen implementieren:

  - **`XXX_PortInit();`**
    Diese Funktion wird direkt nach dem Prozessorstart ausgeführt und soll die Richtung und Standardwerte der einzelnen Pins setzten. Hier soll noch keine Initialisierung stattfinden sonder wirklich nur die Richtung und der Anfangszustand, evtl. mit Pull-Up-Widerstand, gesetzt werden.

    Es ist darauf zu achten, daß diese Funktion so schnell wie möglich zurückkehrt, damit alle Module möglichst bald initialisiert werden können.

  - **`XXX_Init( … );`**
    Im Gegensatz zur `PortInit`-Funktion wird hier das entsprechende Modul initialisiert und für den Empfang der weiteren Befehle vorbereitet. Da alle Module der

Reihe nach initialisiert werden, sollte auch hier nicht mehr Zeit als nötig verbracht werden.

4.2.5. CMSTYPES.H

Diese Headerdatei wird von allen Quellcodedateien eingebunden und beinhaltet die Standardtypen, sowie einige Defines und Makros für dieses Projekt.

Folgende Datentypen sind definiert:

```
//------------------------------------------------------------------------------------------
// common types
//------------------------------------------------------------------------------------------
typedef unsigned char           BYTE;
typedef unsigned short          WORD;
typedef unsigned long           DWORD;
typedef unsigned long long      QWORD;

typedef signed      char        s8;
typedef unsigned    char        u8;
typedef signed      short       s16;
typedef unsigned    short       u16;
typedef signed      long        s32;
typedef unsigned    long        u32;
//------------------------------------------------------------------------------------------
```

Listing 1 – CmSTypes, common types

Weiter ist eine Boolean-Unterstützung implementiert:

```
//------------------------------------------------------------------------------------------
// bool support
//------------------------------------------------------------------------------------------
typedef     BYTE                        BOOL;

#define     FALSE                       0
#define     TRUE                        1

#define     false                       FALSE
#define     true                        TRUE

#define     ON                          TRUE
#define     OFF                         FALSE
//------------------------------------------------------------------------------------------
```

Listing 2 – CmSTypes, bool support

Zudem sind noch einige kleine Makros und Definitionen, die immer wieder benötigt werden, zusammengefaßt:

```
//------------------------------------------------------------------------------------------
// MIN / MAX
//------------------------------------------------------------------------------------------
#define     MIN(a,b)                    (((a)<(b))?(a):(b))
#define     MAX(a,b)                    (((a)<(b))?(b):(a))
//------------------------------------------------------------------------------------------


//------------------------------------------------------------------------------------------
// function modifyers (they do nothing)
//------------------------------------------------------------------------------------------
#define     _public
#define     _private
//------------------------------------------------------------------------------------------


//------------------------------------------------------------------------------------------
// optimized data type macros
//------------------------------------------------------------------------------------------
#define     LOBYTE(w)                   (*((BYTE*)&w))              // optimized asm output
#define     HIBYTE(w)                   (*((BYTE*)(&w) + 1))        // optimized asm output
//------------------------------------------------------------------------------------------


//------------------------------------------------------------------------------------------
// pointer
//------------------------------------------------------------------------------------------
#ifndef NULL
    #define     NULL                    0
#endif
//------------------------------------------------------------------------------------------
```

Listing 3 – CmSTypes, macros and defines

4.3. Software Treiberschicht

4.3.1. Timer

Der Atmel AVR Mega32 ist mit zwei 8-Bit Timern und einem 16-Bit ausgestattet. Da diese aber recht schnell zu wenig werden, muß eine Softwarelösung zur Unterstützung mehrerer Timer her. Dafür ist die Datei `Timer.c` im `misc` Verzeichnis zuständig.

Sie bietet folgende Funktionen an:

4.3.1.1. Interface

```
//-- Init -------------------------------------------------------------------------
    void Timer_Init( void );
//---------------------------------------------------------------------------------

//-- Timer Management -------------------------------------------------------------
    void Timer_SetTimer( BYTE byTimerID, WORD wElapse_milliSec, TIMER_PROC pTimerProc );
    void Timer_KillTimer( BYTE byTimerID );
//---------------------------------------------------------------------------------
```

Listing 4 – Timer, Interface

4.3.1.2. Aufbau

Diese Befehle sind schnell erklärt:

- **`void Timer_Init( void );`**
  Diese Funktion ist beim Systemstart auszuführen, belegt einen 8-Bit Timer und startet diesen.

- **`void Timer_SetTimer( BYTE byTimerID, WORD wElapse_milliSec, TIMER_PROC pTimerProc );`**
  Das ist die Hauptfunktion: Es soll eine ID übergeben werden, damit der Timer eindeutig identifiziert werden kann. Zur Wahrung der Übersichtlichkeit sollte diese ID durch ein Define in der Header-Datei – im dafür vorgesehenen Abschnitt – festgelegt werden. Diese Liste sieht im Moment so aus:

```
//-------------------------------------------------------------------------------------
// timer name defines
//-------------------------------------------------------------------------------------
#define                         TIMER_ID_KEY                    0
#define                         TIMER_ID_GPS                    1
#define                         TIMER_ID_PWR                    2
#define                         TIMER_ID_REC                    3
#define                         TIMER_ID_OPT                    4
//-------------------------------------------------------------------------------------
```

Listing 5 – Timer, Name Defines

Der zweite Parameter gibt die Zeit in Millisekunden an, mit welcher der Timer ausgelöst werden soll. Dieser Wert wird, im Gegensatz zu zum Beispiel Standard Windows- oder Linux-Timern, auch exakt eingehalten. Der Wertebereich reicht von einer Millisekunde bis hin zu etwa 25 Sekunden bei 10 MHz Taktrate. Die Abhängigkeit ist indirekt proportional.

Der letzte Parameter gibt den Pointer für die Callback-Funktion an. Der Timer-Treiber ruft nach dem Erreichen der Wiederholungszeit diese Funktion auf, die wie folgt definiert ist:

```
//-- timer callback ---------------------------------------------------------------
typedef     void (*TIMER_PROC)( void );
```

Listing 6 – Timer, Callback

WICHTIG:
Es ist unbedingt darauf zu achten, daß die Callback-Funktion möglichst schnell die Kontrolle wieder zurückgibt. Während der Abarbeitung können keine anderen Interrupts ausgeführt werden. Diese werden zwar als aufgetreten vorgemerkt und im Anschluß an das Beenden der Callback-Funktion ausgeführt, falls sie in der Zwischenzeit aber öfters aufgetreten sind, kann das im Nachhinein nicht mehr festgestellt werden. Vor allem für den GPS-Empfang ist das sehr wichtig, denn falls dort ein Zeichen vergessen wird, ist der ganze Datensatz unbrauchbar und somit auch Teile der Aufzeichnung.

Die passende Funktion zum Deaktivieren eines Timers ist:

- `void Timer_KillTimer( BYTE byTimerID );`
  Der mit der ID identifizierte Timer wird abgestellt.

4.3.1.3. IMPLEMENTIERUNG KRITISCHER STELLEN

Im Kern des Timer-Treibers steht ein Array aus User-Timer-Informationen. Darin sind alle notwendigen Daten zu den einzelnen aktiven Timern gespeichert.

Es existiert ein Hardware-Timer, der genau jede Millisekunde einmal aufgerufen wird. Dieser erniedrigt die Restdauer bis zum nächsten Auftreten eines jeden User-Timers. Wird eine Ausführungszeit erreicht, so wird der Zähler wieder auf die Anzahl von Millisekunden für diesen Timer zurückgesetzt und dessen Callback-Funktion aufgerufen.

4.3.1.4. STATISTIK

Hier eine Aufstellung des ungefähr benötigten Speichers:

Speicherart	Größe [Bytes]
Programmcode	636
RAM	30

Tabelle 42 – Timer-Treiber Speicherverbrauch

4.4. MODULTREIBER

4.4.1. LCD

4.4.1.1. AUFGABE

Die Aufgabe dieses Treibers ist es ein API für das Display zu schaffen, um es entsprechend seiner Spezifikation aus dem Grundlagenkapitel anzusteuern.

Neben den Standard-Kontroll-Funktionen, wie das Initialisieren des Displays und Platzieren und Steuern des Cursors, soll es Funktionen für einen direkten Zugriffsmodus besitzen. Darin ist das Display fest ohne Scrollmöglichkeit und es können an beliebigen Positionen Zeichen, Texte oder Zahlen ausgegeben werden.

Zusätzlich soll es einen Konsolen-Modus geben, in dem das LCD-Display wie eine Konsole verwendet werden und man zeilenweise Text ausgeben kann. Erreicht man den unteren Rand und muß umbrechen, so scrollt das Display automatisch weiter.

Nicht vergessen werden darf die Power-Management-Funktion zum Schalten der Hintergrundbeleuchtung.

4.4.1.2. HARDWAREKONFIGURATION

Das LCD-Modul benutzt einen 8-Bit Datenbus und einen 4-Bit Steuerbus, die direkt an das LCD-Display weitergeleitet werden und verarbeitet sonst nur noch ein einzelnes Signal zum Steuern der Hintergrundbeleuchtung.

Es wird davon ausgegangen, daß die 8-Datenbit auf einem kompletten Port liegen. Die vier Steuerbit müssen auf demselben Port sein, die Pins sind jedoch beliebig wählbar. Letztendlich darf die Hintergrundbeleuchtung an einem beliebigen Port/Pin angeschlossen werden.

Hier die Hardwarekonfiguration:

```
//------------------------------------------------------------------------------------------
// ### hardware config
//------------------------------------------------------------------------------------------
//-- data port defines ---------------------------------------------------------------------
#define     LCD_DATA_PORT           PORTA       // data port register
#define     LCD_DATA_PORT_IN        PINA        // data port in register
#define     LCD_DATA_PORT_DDR       DDRA        // data port data direction register

//-- ctrl port defines ---------------------------------------------------------------------
#define     LCD_CTRL_PORT           PORTD       // ctrl port register
#define     LCD_CTRL_PORT_DDR       DDRD        // ctrl port data direction register

#define     LCD_CTRL_BIT_RS         4           // bit in ctrl port for <register select>
#define     LCD_CTRL_BIT_RW         5           // bit in ctrl port for <read/write>
#define     LCD_CTRL_BIT_E          6           // bit in ctrl port for <enable>
#define     LCD_CTRL_BIT_CS         7           // bit in ctrl port for <chip select>

//-- lcd background power [out] port defines -----------------------------------------------
#define     LCD_BG_PORT             PORTB       // BG port register
#define     LCD_BG_PORT_DDR         DDRB        // BG data direction register
#define     LCD_BG_PORT_BIT         2           // BG port bit
#define     LCD_BG_INVERTED                     // BG is inverted, else comment out
#define     LCD_BACKGOUND_STARTUP   ON          // BG startup state (ON/OFF)
//------------------------------------------------------------------------------------------
```

Listing 7 – LCD, Hardware Config

4.4.1.3. INTERFACE

```
//-- Init -------------------------------------------------------------------------------------
    void LCD_PortInit( void );
    void LCD_Init( void );
//---------------------------------------------------------------------------------------------

//-- Power Management -------------------------------------------------------------------------
    void LCD_Power_LCDBackground( BOOL bState );
//---------------------------------------------------------------------------------------------

//-- Direct -----------------------------------------------------------------------------------
    void LCD_SetDisplayOn( BOOL bOn );
    void LCD_SetCursorOn( BOOL bCursor );
    void LCD_SetupDisplayConfig( BYTE byCharXDots, BYTE byCharYDots, BYTE byNumCharsPerLine, BYTE
                        byNumDisplayVerticalPixels, BYTE byCursorYPos );

    void LCD_SetDisplayStart( BYTE byDisplayStartLine );
    void LCD_SetCursorPos( BYTE byCursorPosX, BYTE byCursorPosY );
    void LCD_PutChar( char cChar, BOOL bTranslateASCII );
    void LCD_PutString( const char *strString, BOOL bTranslateASCII );
    void LCD_PutStringXY( BYTE byCursorPosX, BYTE byCursorPosY, const char *strString, BOOL bTranslateASCII
                        );
    void LCD_PutValueXY( BYTE byCursorPosX, BYTE byCursorPosY, DWORD dwValue, u8 byFlags );
    void LCD_PutStaticLine( BYTE byLineY, const PGM_P pgmstrString );

    void LCD_ClearAndResetScreen( void );
    void LCD_ClearAndResetScreen_Fast( void );
    BYTE LCD_TranslateASCII( BYTE cChar );
//---------------------------------------------------------------------------------------------


//-- Console ----------------------------------------------------------------------------------
    void LCDC_Init( void );
    void LCDC_Cls( void );
    void LCDC_Print( const char* strText );
    void LCDC_PrintChar( char cChar );
//---------------------------------------------------------------------------------------------
```

Listing 8 – LCD, Interface

4.4.1.4. AUFBAU DES MODULTREIBERS

Zuerst die notwendigen Treiber Funktionen:

- **`void LCD_PortInit( void );`**
  Setzt die Datenports auf Eingang und die Steuerports auf Ausgang mit den Standardwerten. Zuletzt wird die LCD Hintergrundbeleuchtung auf den Default-Wert gesetzt.

- **`void LCD_Init( void );`**
  Setzt initialen Modus (Display an, Textmodus, Cursor an und blinkend). Danach konfiguriert es den Controller entsprechend mit den für das Display gültigen Daten: Zeichengröße: 6x9, 42 Zeichen pro Zeile, 64 Pixel Höhe und den Cursor in der neunten Zeile eines Zeichens. Anschließend wird das Display komplett gelöscht.

Dann die Power-Management-Funktion:

- **`void LCD_Power_LCDBackground( BOOL bState );`**
  Schaltet anhand der Hardwarekonfiguration und `bState` die Beleuchtung an oder aus.

Um Befehle an das Display zu senden, wird folgende private Funktion verwendet:

- **`_private void LCD_Command( BYTE byInstruction, BYTE byData );`**
  Diese Funktion hält sich an die im Datenblatt [LCD] beschriebene Kommunikation und sendet zuerst das Instruction-Byte und anschließend ein Daten-Byte. Da dies die zentrale Funktion des LCD-Modultreibers ist, wird sie im nächsten Abschnitt genauer beschrieben.

Nun folgen einige Befehle zur Grundkonfiguration des Displays, welche die vorhanden Befehle des Displays benutzten:

- **`void LCD_SetDisplayOn( BOOL bOn );`**
  Sendet das Kommando zum Ein- oder Ausschalten des Displays mit dem Befehl `LCD_Command( LCD_CMD_SET_MODE, …)`.

- **`void LCD_SetCursorOn( BOOL bCursor );`**
  Schaltet den Cursor ebenfalls mit dem Befehl `LCD_Command( LCD_CMD_SET_MODE, …)` ein oder aus. Im normalen – dem direkten – Modus, ist der Cursor anfänglich ausgeschaltet und sollte nur bei Eingaben eingeschaltet werden. Im Konsolenmodus ist er jedoch aktiviert.

- **`void LCD_SetupDisplayConfig( BYTE byCharXDots, BYTE byCharYDots, BYTE byNumCharsPerLine, BYTE byNumDisplayVerticalPixels, BYTE byCursorYPos );`**
  Hier werden die übergebenen Parameter gemäß den einzelnen Befehlen verknüpft, auf Korrektheit hin überprüft und anschließend gleich über vier `LCD_Command(…)` Befehle infolge gesetzt. Die einzelnen Kommandos sind: `LCD_CMD_SET_CHAR_PITH`, `LCD_CMD_SET_NUM_XCHARS`, `LCD_CMD_SET_NUM_YLINES` und `LCD_CMD_SET_CURSORLINE_POS`.

- **`void LCD_SetDisplayStart( BYTE byDisplayStartLine );`**
  Hier wird zuerst die Zeilennummer in die Anzahl der Zeichen umgerechnet und diese dann mit den beiden `LCD_CMD_SET_DISP_START_LO/HI` Befehlen gesetzt. Damit kann man Scrollen realisieren oder einfach nur den Bildschirm umschalten, ohne die alten Daten zu verlieren.

- **`void LCD_SetCursorPos( BYTE byCursorPosX, BYTE byCursorPosY );`**
  Die letzte Funktion der Basiskommandos setzt den Cursor an eine bestimmte X/Y Koordinate anhand der gesetzten Auflösung mit den zwei `LCD_CMD_SET_CURSOR_POS_LO/HI` Befehlen.

Jetzt werden die Ausgabefunktionen näher beschrieben:

- **`void LCD_PutChar( char cChar, BOOL bTranslateASCII );`**
  Die einfachste und grundlegende Ausgabefunktion, die genau ein Zeichen `cChar` an der aktuellen Cursorposition ausgibt. Anhand des Parameters `bTranslateASCII` wird entschieden, ob vor der Ausgabe noch eine weitere private Funktion `LCD_TranslateASCII(…)` aufgerufen wird, welche die übergebenen ASCII-Werte anhand einer Tabelle auf den Zeichensatz des Displays ändert. Sie ändert insbesondere deutsche Sonderzeichen, wie „ä“, „ö“, „ü“, „ß“ und „°“ in die passenden Zeichen um und wandelt ein „\“ Zeichen in ein „/“ Zeichen um, da das Display keine passende Alternative besitzt. Zuletzt werden noch die globalen Werte für die aktuelle Cursorposition aktualisiert.

- **`void LCD_PutString( const char *strString, BOOL bTranslateASCII );`**
  Dies ist die konsequente Steigerung der `PutChar`-Funktion. Sie gibt einfach Schritt für Schritt alle Zeichen im String mit der gerade beschriebenen Funktion einzeln aus.

- **`void LCD_PutStringXY( BYTE byCursorPosX, BYTE byCursorPosY, const char *strString, BOOL bTranslateASCII );`**
  Jetzt geht es noch eine Stufe weiter: Vor dem Aufruf von `PutString(…)` wird zusätzlich noch die aktuelle Cursorposition verändert.

- **`void LCD_PutValueXY( BYTE byCursorPosX, BYTE byCursorPosY, DWORD dwValue, u8 byFlags );`**
  Das ist die mächtigste der Ausgabefunktionen. Sie wandelt zuerst den Wert von `dwValue` anhand der `byFlags`-Richtlinien um und ruft dann `PutStringXY(…)` mit dem neu erzeugten String auf. Die für die Umwandlung eines Zahlenwertes in einen String verantwortliche Funktion `cvt_32_to_str(…)` wird später im Implementierungsabschnitt noch ausführlich besprochen.

- **`void LCD_PutStaticLine( BYTE byLineY, const PGM_P pgmstrString );`**
  Jeder normal erzeugte String (z.B. mit `char strMyString[] = „Hallo Welt!“`) belegt zusätzlich – zum im Programmspeicher abgelegten Inhalt – noch einmal den gleichen Speicher im RAM, was natürlich bei einer größeren Anzahl von konstanten Texten unnötig viel RAM-Speicherverbrauch ist. Es gibt im avr-gcc eine Möglichkeit einen Text nur im Programmspeicher abzulegen und somit den wertvollen RAM-Platz zu sparen. Zwei Einschränkungen muß man dann jedoch hinnehmen: Der Text ist `const`, kann also verständlicherweise nicht verändert werden und um auf ihn zugreifen zu können, muß er zur Laufzeit erst in einen dynamischen String geladen werden. Dafür gibt es extra die `strncpy_P(…)` Funktionsgruppe (u.a. auch nur `strcpy_P(…)`, `strcat_P(…)`, `emcpy_P(…)`, usw.). Diese Funktionen erwarten einen Pointer im RAM und einen aus dem Flash. Für eine genauere Beschreibung sei hier auf die Dokumentation vom avr-gcc [AVRGCC] verwiesen.

  Diese Ausgabefunktion gibt nun einen im Programmspeicher hinterlegten String am Anfang der angegeben Zeile `byLineY` aus. Die maximale Länge des Strings ist auf die Zeilenbreite beschränkt. Ist er länger, wird er einfach am Ende der Zeile abgebrochen.

Nun noch die beiden Funktionen zum Löschen des Displays:

- **`void LCD_ClearAndResetScreen( void );`**
  Hier wird das Display ausgeschaltet, die Startposition sowohl des Displays, als auch des Cursors auf 0 gesetzt und der komplette Datenspeicher des Displays mit Leerzeichen überschrieben.

- **`void LCD_ClearAndResetScreen_Fast( void );`**
  Die wesentlich schnellere Funktion löscht nicht den kompletten Speicher, sondern nur den für die erste Seite sichtbaren Bereich.

Zur Vollständigkeit noch folgende Funktion:

- **`BYTE LCD_TranslateASCII( BYTE cChar );`**
  Die Arbeitsweise wurde schon in `LCD_PutChar(…)` genauer beschrieben.

Zum Abschluß noch die Funktionen für die Konsole, die nicht nur mit `LCD_` sondern mit `LCDC_`, für Console, starten:

- **`void LCDC_Init( void );`**
  Neben einem normalen Löschen und Initialisieren des Speichers schaltet der Befehl den Cursor ein.

- **`void LCDC_Cls( void );`**
  Ruft nur `LCD_ClearAndResetScreen(…)` auf.

- **`void LCDC_Print( const char* strText );`**
  Hier steckt nun die gesamte Logik der Konsole. Die Funktion gibt Zeichen für Zeichen über `LCD_PutChar(…)` aus, wobei einige davon speziell behandelt werden: Es gibt eine Unterstützung für das Tabulatorzeichen „\t“ und der Zeilenumbruch „\n“ wird korrekt behandelt. Nach jeder Zeichenausgabe überprüft der Befehl, ob ein Scrollen notwendig ist, weil der Cursor aus der untersten Zeile gerutscht ist. Beim Scrollen erhöht man einfach die Startadresse des Displays um eine Zeile.

- **`void LCDC_PrintChar( char cChar );`**
  Die letzte der doch recht umfangreichen LCD-API. Um einzelne Zeichen auf der Konsole auszugeben, wird ein kleiner Trick zu Hilfe genommen: Es wird lokal einfach ein String mit der Länge 1 angelegt und über die `Print(…)` Funktion ausgegeben.

4.4.1.5. Implementierung kritischer Stellen

Dieses Modul enthält nur zwei kritische Stellen: `LCD_Command(…)`, das für den kompletten Datentransfer zum Display zuständig ist und die Hilfsfunktion `cvt_32_to_str(…)`, die das Konvertieren von Zahlen zu Strings erledigt.

LCD_Command(…)

Um die Funktion möglichst einfach zu gestalten, kommen hier verstärkt Makros zum Einsatz. Unter anderem werden am Anfang des Treibers folgende definiert:

```
//------------------------------------------------------------------------------------------
// io defines
//------------------------------------------------------------------------------------------
// sets the direction of the data port
#define     LCD_SET_DATA_PORT_INPUT     { LCD_DATA_PORT = 0x00; LCD_DATA_PORT_DDR = 0x00; }
#define     LCD_SET_DATA_PORT_OUTPUT    { LCD_DATA_PORT_DDR = 0xff; }

// sets or clears one bit in a byte
#define     SET_BIT(val,bit)            { val |= (1<<bit); }
#define     CLR_BIT(val,bit)            { val &= ~(1<<bit); }

// control port macros (see data sheet for details)
// RS control line (register select)
#define     LCD_CTRL_RS_INSTRUCTION     { SET_BIT( LCD_CTRL_PORT, LCD_CTRL_BIT_RS ); }
#define     LCD_CTRL_RS_DATA            { CLR_BIT( LCD_CTRL_PORT, LCD_CTRL_BIT_RS ); }

// RW control line ( read/write)
#define     LCD_CTRL_RW_READ            { SET_BIT( LCD_CTRL_PORT, LCD_CTRL_BIT_RW ); }
#define     LCD_CTRL_RW_WRITE           { CLR_BIT( LCD_CTRL_PORT, LCD_CTRL_BIT_RW ); }

// E control line (enable)
#define     LCD_CTRL_E_ENABLE           { SET_BIT( LCD_CTRL_PORT, LCD_CTRL_BIT_E ); }
#define     LCD_CTRL_E_DISABLE          { CLR_BIT( LCD_CTRL_PORT, LCD_CTRL_BIT_E ); }

// CS control line (chip select)
#define     LCD_CTRL_CS_SELECT          { CLR_BIT( LCD_CTRL_PORT, LCD_CTRL_BIT_CS ); }
#define     LCD_CTRL_CS_UNSELECT        { SET_BIT( LCD_CTRL_PORT, LCD_CTRL_BIT_CS ); }

// busy flag in data register
#define     BUSY_FLAG_BIT               7
//------------------------------------------------------------------------------------------
```

Listing 9 – LCD, LCD_Command Defines

Das Makro `LCD_NOP` definiert über eine ausgeklügelte Makrostruktur eine bestimmte Anzahl von Assembler `nop`-Befehlen, in Abhängigkeit der aktuellen Taktgeschwindigkeit. Damit wird gewährleistet, daß das Display innerhalb der Spezifikation betrieben und nicht übertaktet wird. Hier die Liste mit der Anzahl der `nops` in Abhängigkeit von der Taktfrequenz:

AVR Takt [MHz]	Anzahl NOPs
<= 2 MHz	0
<= 4 MHz	1
<= 6 MHz	2
<= 8 MHz	3
<= 10 MHz	4
<= 14 MHz	6
<= 20 MHz	9

Tabelle 43 – LCD, LCD_Command Nops

Und nun der Code der besagten Funktion: Wenn man hier mit dem LCD-Timing-Diagramm aus dem Grundlagenkapitel vergleicht, sieht man sehr schön jeden einzelnen Schritt:

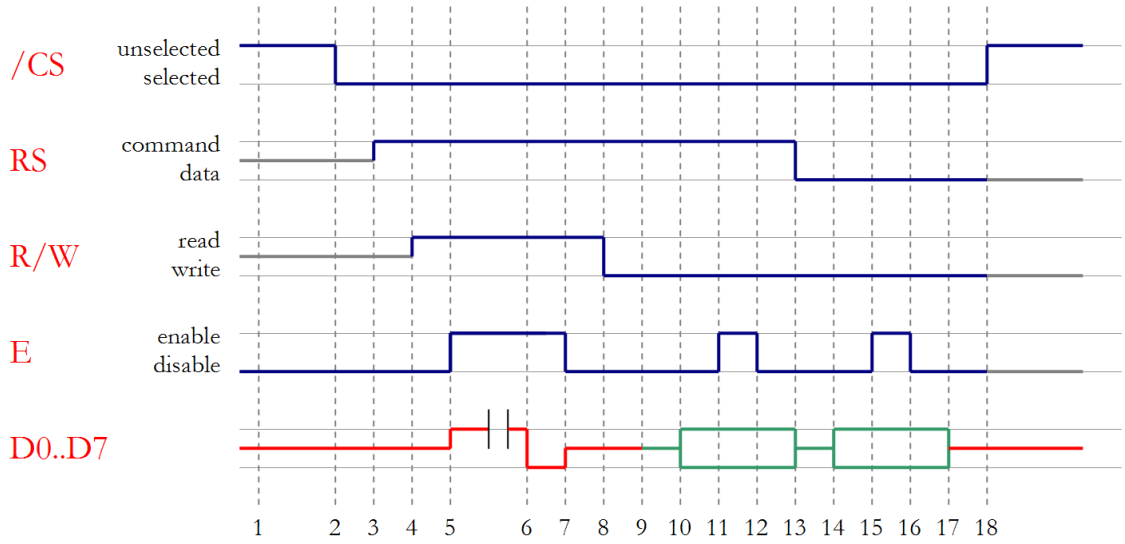


Abbildung 133 – LCD-Timing-Diagramm

```
_private void LCD_Command( BYTE byInstruction, BYTE byData )
{
    // start command sequence
    LCD_CTRL_CS_SELECT;                     // select chip

    // wait while LCD still busy (last command in process)
    LCD_SET_DATA_PORT_INPUT;                // set port direction as input
    LCD_CTRL_RS_INSTRUCTION;                // instruction byte
    LCD_CTRL_RW_READ;                       // read from LCD
    LCD_CTRL_E_ENABLE;                      // enable
    LCD_NOP;                                // wait for LCD
    loop_until_bit_is_clear( LCD_DATA_PORT_IN, BUSY_FLAG_BIT );   // wait while busy
    LCD_CTRL_E_DISABLE;                     // disable

    // prepare for command
    LCD_CTRL_RW_WRITE;                      // write to LCD
    LCD_SET_DATA_PORT_OUTPUT;               // set port direction as output

    // now write instruction byte
    LCD_CTRL_RS_INSTRUCTION;                // instruction bvte
    LCD_DATA_PORT = byInstruction;          // set instruction
    LCD_CTRL_E_ENABLE;                      // enabe strobe
    LCD_NOP;                                // wait for LCD
    LCD_CTRL_E_DISABLE;                     // disable
    LCD_NOP;                                // wait for LCD
```

```
    // now write data byte
    LCD_CTRL_RS_DATA;                   // data byte
    LCD_DATA_PORT = byData;             // set data
    LCD_CTRL_E_ENABLE;                  // enabe strobe
    LCD_NOP;                            // wait for LCD
    LCD_CTRL_E_DISABLE;                 // disable
    LCD_NOP;                            // wait for LCD

    // end command sequence
    LCD_SET_DATA_PORT_INPUT;            // set port direction as input again
    LCD_CTRL_CS_UNSELECT;               // deselect chip
}
```

Listing 10 – LCD, LCD_Command

CVT_32_TO_STR

Da das Umwandeln von Zahlen in Strings relativ aufwendig und von allgemeiner Bedeutung ist, befindet es sich nicht im LCD-Treiber sondern außerhalb in `misc/CmConvert.c`. Darin befinden sich zwei Funktionen, die eine mit einzelnen Parametern, die andere mit Flags zum Übergeben.

Es hätte auch die `sprintf(…)` Funktion verwendet werden können. Diese benötigt aber meiner Meinung nach extrem viel Speicherplatz, ist relativ langsam und kann viel mehr – meist aber ungenutzte Funktionalität.

- **`void cvt_32_to_str_ex( char *strBuffer, u32 dwVal, BOOL bSigned, u8 byMinDigits, u8 byMinChars )`**

  `strBuffer` ist der Ausgabepuffer, der vom Aufrufenden zur Verfügung gestellt werden muß und mindestens `CVT_MAX_STRLEN` lang sein muß. In `dwVal` wird der besagte Wert, als `BYTE, WORD, DWORD` oder dessen vorzeichenbehafteten Versionen übergeben. Die restlichen drei Parameter bestimmen die Formatierung der Ausgabe.

  Wenn `bSigned` gesetzt ist, wird die Zahl als vorzeichenbehaftet betrachtet und bei einem negativen Wert ein „-“ Zeichen vorangestellt. Ein `BYTE` mit dem Wert 255 wird also entweder als 255 oder als -1 ausgegeben.

  Mit `byMinDigits` wird die minimale Anzahl an Ziffern zur Ausgabe festgelegt. Restliche, links stehende, Ziffern werden mit 0 aufgefüllt. Bei einem Wert von 52 und `byMinDigits` gleich 0 wird auch nur 52 ausgegeben. Erhöht man diesen Wert auf 4, so lautet die Ausgabe 0052. Damit kann man feste Breiten erreichen oder Nachkommastellen darstellen.

  Der letzte Parameter funktioniert ähnlich dem eben beschriebenen, nur wird hier nicht mit „0“en aufgefüllt, sondern mit Leerzeichen. Aus den besagten „52“ werden bei einem Wert von 4 „ 52“, oder besser lesbar „`..52`“. Damit lassen sich rechtsbündig formatierte Ausgaben erstellen.

  Für den genauen Code sei hier direkt auf den Quellcode verwiesen. Diese Funktion ist, wie die meisten anderen Funktionen auch, sehr gut dokumentiert und selbsterklärend.

Mit den drei nötigen Parametern zur Formatierung ist diese Funktion allerdings recht umständlich zu handhaben. Deshalb wird eine neue eingeführt, die nur einen Parameter zur Format-Übergabe benötigt:

- **`void cvt_32_to_str( char *strBuffer, u32 dwVal, u8 byFlags );`**

  In dem Parameter byFlags wird das Format gepackt und nach folgender Tabelle übergeben:

```
// flags for cvt_32_to_str function
#define     CVT_SIGNED                  0x80
#define     CVT_UNSIGNED                0x00
```

```
#define     CVT_1DIGIT                  0x10
#define     CVT_2DIGITS                 0x20
#define     CVT_3DIGITS                 0x30
#define     CVT_4DIGITS                 0x40
#define     CVT_5DIGITS                 0x50
#define     CVT_6DIGITS                 0x60
#define     CVT_7DIGITS                 0x70

#define     CVT_1CHAR                   0x01
#define     CVT_2CHARS                  0x02
#define     CVT_3CHARS                  0x03
#define     CVT_4CHARS                  0x04
#define     CVT_5CHARS                  0x05
#define     CVT_6CHARS                  0x06
#define     CVT_7CHARS                  0x07
#define     CVT_8CHARS                  0x08
#define     CVT_9CHARS                  0x09
#define     CVT_10CHARS                 0x0a
#define     CVT_11CHARS                 0x0b
#define     CVT_12CHARS                 0x0c
```

Listing 11 – LCD, cvt_32_to_str Defines

Beim Aufruf der Funktion kann nun der Übergabewert einfach durch die OR-Funktion hergestellt werden. Möchte man einen vorzeichenbehafteten Wert mit mindestens 4 Ziffern, so ergibt sich ein Flag-Wert von `CVT_SIGNED | CVT_4DIGITS`.

4.4.1.6. Beispielverwendung und Tests

Einen ersten Test des direkten Modus, kann man mit folgendem Code ausführen:

```
// init
LCD_PortInit();
LCD_Init();

// enable backlight
LCD_Power_LCDBackground( ON );

// test text output functions
LCD_PutStringXY( 0, 0, "Temperatur:", true );
LCD_PutStringXY( 0, 1, "Feuchtigkeit:", true );
LCD_PutStringXY( 0, 3, "Uhrzeit:", true );

// test special chars
LCD_PutStringXY( 20, 0, "20 °C", true );
LCD_PutStringXY( 20, 1, "50 % rel", true );
LCD_PutStringXY( 20, 3, "10:37:15", true );
LCD_PutStringXY( 20, 5, "äöüß\\/", true );

// test convert engine
LCD_PutValueXY( 18, 0, sTemperature, CVT_SIGNED|CVT_4CHARS );

// clear screen
LCD_ClearAndResetScreen_Fast();
```

Listing 12 – LCD, Test 1

Die Konsole kann mit diesem Code getestet werden. Zur Vereinfachung und zu Testzwecken wird hier die `sprint`-Funktion doch verwendet:

```
// init
LCD_PortInit();
LCD_Init();

// init console
LCDC_Init();

LCDC_Print( "C:\\> Hallo?\nabcdefghijklmnopqrstuvwxyz\nA\tB\tCC;\nAA\tBBBBBB\t[]<>{}|~;\näöü ÄÖÜ ß \n^°
                          !\"$%&/()=? +* #' -_ .: ,;" );

short   i = 7;
while (1)
{
    char    s[16];
    sprintf( s, "N %d:\n", i++ );

    _delay_ms(200.);
    LCDC_Print( s );
}
```

Listing 13 – LCD, Test 2

Fotos vom Display:

Abbildung 134 – LCD, GPS-NMEA-Ausgabe

Erste Gehversuche des Displays. Hier werden die empfangenen NMEA Daten direkt auf dem Display im Konsolenmodus ausgegeben.

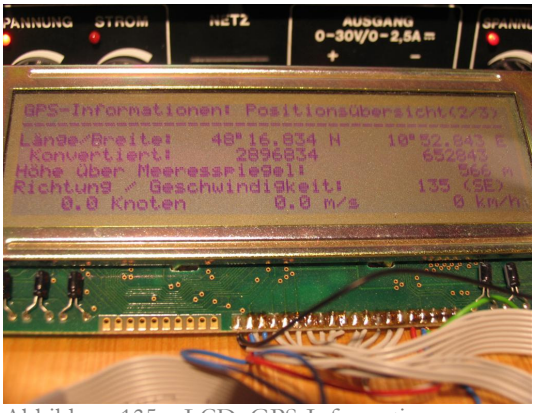


Abbildung 135 – LCD, GPS-Informationen

Hier sieht man das Display bei Tests zur Anzeige der GPS-Informationen im direkten Modus.

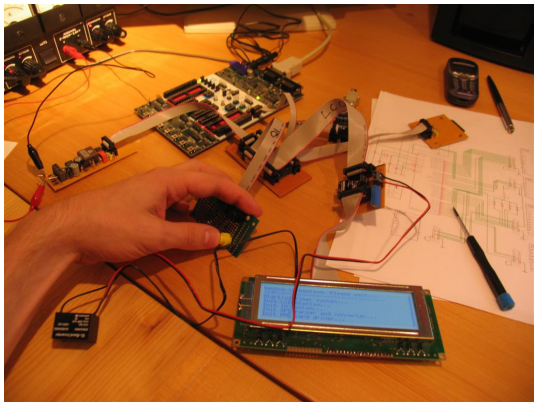


Abbildung 136 – LCD, Hintergrundbeleuchtung

Nach dem Erstellen der Platinen wurde selbstverständlich auch das LCD-Modul ausführlich getestet und das erste Mal mit seiner Hilfe die Hintergrundbeleuchtung eingeschaltet.

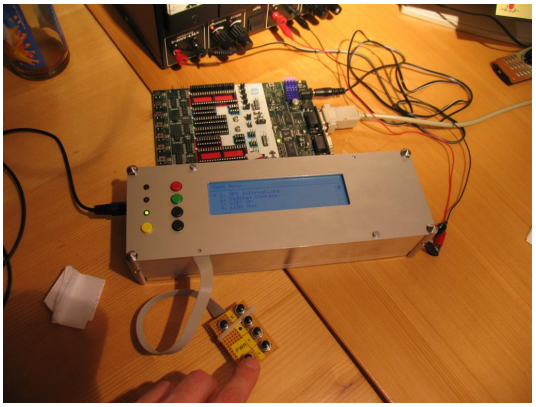


Abbildung 137 – LCD, eingebauter Zustand

Abschließend noch ein Foto im eingebauten Zustand mit einem Menü.

4.4.1.7. Berechnungen, Analysen und Statistiken

Hier eine Aufstellung des ungefähr benötigten Speichers:

Speicherart	Größe [Bytes]
Programmcode	1944
RAM	5

Tabelle 44 – LCD-Modul Speicherverbrauch

4.4.2. GPS

4.4.2.1. AUFGABE

Die Aufgabe dieses Treiber-Moduls ist es, einen NMEA-Parser – nebst dafür zuständigem API für den GPS-Empfänger – bereitzustellen.

Neben den notwendigen Grundfunktionen zur Initialisierung sind diesmal zwei Power-Management-Funktionen nötig: Die eine zum Schalten des GPS-Empfängers und die andere zum Auswählen der Eingangssignalquelle.

Die Daten werden auf Grund der Vielzahl von Elementen nicht über eine Funktion übergeben sondern befinden sich in einer globalen Struktur.

Einzig und allein eine Funktion muß vorhanden sein, die meldet, ob seit dem letzten Aufruf eine Aktualisierung der Struktur stattgefunden hat.

4.4.2.2. HARDWAREKONFIGURATION

Das Modul braucht zur Steuerung drei Bits, von denen jedes einzeln definiert werden kann.

Der Receive-Eingang des seriellen Signals muß nicht extra definiert werden, da er sich physikalisch am PortD0, dem RxD-Pin, befinden muß.

```
//------------------------------------------------------------------------------------------
// ### hardware config
//------------------------------------------------------------------------------------------
// gps power [out]
#define     GPS_PWR_PORT            PORTB       // GPS port register
#define     GPS_PWR_PORT_DDR        DDRB        // GPS data direction register
#define     GPS_PWR_PORT_BIT        1           // GPS port bit
//#define   GPS_PWR_INVERTED                    // GPS is inverted, else comment out
#define     GPS_POWER_STARTUP       OFF         // GPS startup power status (ON/OFF)

// gps channel 0/1 [out]
#define     GPS_CH0_PORT            PORTD       // GPS ch0 port register
#define     GPS_CH0_PORT_DDR        DDRD        // GPS ch0 data direction register
#define     GPS_CH0_PORT_BIT        1           // GPS ch0 port bit
#define     GPS_CH1_PORT            PORTC       // GPS ch1 port register
#define     GPS_CH1_PORT_DDR        DDRC        // GPS ch1 data direction register
#define     GPS_CH1_PORT_BIT        7           // GPS ch1 port bit
//------------------------------------------------------------------------------------------
```

Listing 14 – GPS, Hardware Config

4.4.2.3. INTERFACE

Das API gestaltet sich recht schlank:

```
//-- Init and start ------------------------------------------------------------------------
    void GPS_PortInit( void );
    void GPS_Init( void );
//------------------------------------------------------------------------------------------

//-- Power Management ----------------------------------------------------------------------
    void GPS_Power_GPS( BOOL bState );
    void GPS_SelectChannel( BYTE byChannel );
//------------------------------------------------------------------------------------------

//-- Control -------------------------------------------------------------------------------
    BOOL GPS_NewDataAvailable( void );
//------------------------------------------------------------------------------------------
```

Listing 15 – GPS, Interface

Die Struktur der empfangenen und dekodierten NMEA-Datensätzen nimmt jedoch schon ganz andere Dimensionen an:

```
//------------------------------------------------------------------------------------------
// GPS data typedef
//------------------------------------------------------------------------------------------
typedef struct _tGPS_DATA
{
    // time and date
    BYTE        byTime_Hour;                // current hour (gmt time)      0..23
    BYTE        byTime_Minute;              // current minute               0..59
    BYTE        byTime_Second;              // current second               0..59

    BYTE        byDate_Day;                 // current day                  1..31
    BYTE        byDate_Month;               // current month                1..12
    BYTE        byDate_Year;                // current year (+2000)  01 -> 2001

    // latitude: Format: dd°mm.mmm
    BYTE        byLatitude_Degree;          // dd                           0..90
    BYTE        byLatitude_Minute;          // mm                           0..59
    WORD        wLatitude_Minute1000th;     // .mmm                         000..999
    char        cLatitude_Direction;        // N or S

    // longitude: Format: ddd°mm.mmm
    BYTE        byLongitude_Degree;         // ddd                          0..180
    BYTE        byLongitude_Minute;         // mm                           0..59
    WORD        wLongitude_Minute1000th;    // .mmm                         000..999
    char        cLongitude_Direction;       // E or W

    // altitude
    short       sAltitude_Meter;            // heigth over sea in meters

    // speed and direction
    WORD        wSpeed_Knots10th;           // speed over ground in 1/10 knots
    WORD        wCourse_Degree;             // direction over ground        0...359

    // quality and information
    BOOL        bGPSConnected;              // FALSE=not connected
    BYTE        byQuality;                  // 0=invalid, 1=GPS fix, 2 = DGPS fix
    BYTE        byNumberOfSatellites;       // number of visible satellites
} tGPS_DATA;
//------------------------------------------------------------------------------------------


//------------------------------------------------------------------------------------------
// exported globals
//------------------------------------------------------------------------------------------
// current GPS status: see type _tGPS_DATA for details
extern volatile tGPS_DATA       GPS_DATA;
//------------------------------------------------------------------------------------------
```

Listing 16 – GPS, Data Structures

Bei `exported globals` sieht man, daß der oben definierte Typ einmal angelegt und exportiert wird.

4.4.2.4. AUFBAU DES MODULTREIBERS

Zum Begin wieder die Standardfunktionen:

- **`void GPS_PortInit( void );`**
  Hier werden nur die drei Signale für das Power-Management initialisiert.

- **`void GPS_Init( void );`**
  In dieser Funktion werden die globalen Variablen initialisiert. Der empfangende Teil der seriellen Schnittstelle, inklusive Interrupt, wird aktiviert und das GPS-Gerät in den Startzustand gebracht. Danach wird vorerst der Eingangskanal NULL ausgewählt und ein Timeout-Timer, der später noch beschrieben wird, gestartet.

Danach die Power-Management-Funktionen:

- **`void GPS_Power_GPS( BOOL bState );`**
  Schaltet den Empfänger ein oder aus.

- **`void GPS_SelectChannel( BYTE byChannel );`**
  Wählt den gewünschten Eingangskanal im Multiplexer auf dem GPS-Modul aus.

Zuletzt die Funktion zur Erkennung neuer Daten:

- **`BOOL GPS_NewDataAvailable( void );`**
  Falls seit dem letzten Aufruf ein neuer Datensatz eingegangen ist und die Datenstruktur verändert wurde, liefert diese Funktion `TRUE` zurück.

Wie man hier schon erkennen kann, muß die gesamte Logik des Moduls wohl im Receive-Interrupt des Treibers ablaufen.

Doch zuerst noch einmal zurück zum Timeout-Timer: Dieser wird alle drei Sekunden aufgerufen und prüft, ob seit dem letzten Aufruf Daten eingegangen sind. Falls nicht, wird das Feld `bGPSConnected` in der GPS-Struktur auf `FALSE` gesetzt. Damit kann später von außen erkannt werden, ob überhaupt ein GPS-Empfänger angeschlossen ist.

Zurück zum Empfangsinterrupt:

- **`Serial Receive Interrupt`**

Der Interrupt ist eine Status-Maschine und befindet sich zu jeder Zeit in einem bestimmten Zustand. Zu Begin in `GPS_CMD_INVALID`. Nach dem Empfang eines neuen Zeichens wird der Empfang auf einen Übertragungsfehler oder einen Überlauf hin geprüft. Ist dies der Fall, wird sofort wieder in den Status `GPS_CMD_INVALID` gesprungen. Wird ein neuer Befehl empfangen, also ein „$“ Zeichen, so wird unabhängig vom vorherigen Status in `GPS_CMD_NEW` geschaltet.

Wenn bis dato alles in Ordnung ist und sich die Maschine nicht in `GPS_CMD_INVALID` befindet, wird das neue Zeichen ausgewertet. Ist es ein Trennzeichen eines Parameters, also ein Komma oder ein Asterix (Trennzeichen vor der Checksumme), so wird der empfangene Puffer geparst. Ansonsten wird das empfangene Zeichen diesem Puffer einfach nur angehängt und der Interrupt beendet.

Die Parserfunktion wird also nur aufgerufen, wenn ein neuer Parameter empfangen wurde. Befindet sich die Maschine in `GPS_CMD_NEW`, so beinhaltet der Puffer nicht einen Parameter sondern den neuen Befehl, der erst ausgewertet werden muß. Anhand des Befehls wird dann entschieden, ob der Befehl von Interesse ist oder ignoriert wird. In diesem Fall wechselt die Maschine wieder in `GPS_CMD_INVALID`. Wird der Befehl allerdings akzeptiert, schaltet die Maschine in den Status des jeweiligen Befehls und setzt den Parameter Index auf 1. Mögliche Stati sind `GPS_CMD_GGA/RMC/GSA/GSV`.

Wenn nun ein weiteres Trennzeichen empfangen wird und sich die Maschine in einem Befehlsstatus befindet, also nicht in `NEW`, so kann der empfangene Puffer eindeutig einem Befehl und einem Parameterindex zugeordnet werden, der gleich darauf ausgewertet und in die globale Struktur abgespeichert wird. Anschließend wird nur noch der Parameterindex erhöht, damit beim nächsten Mal auch der darauf folgende Parameter erkannt wird.

Falls der letzte interessante Parameter oder das Asterix-Trennzeichen empfangen wurde, ist der Befehl damit abgeschlossen und die Maschine geht wieder in den `INVALID`-Status.

Zur Veranschaulichung ein kleines Beispiel. Folgender Datenstream wird empfangen:
`old_values$GPFOO,param1,param2*checksum_other_...`

Die Maschine befindet sich zu Begin in INVALID, bis sie das „$“ Zeichen bekommt und wechselt dann in NEW. Die nachfolgenden Zeichen werden im Puffer solange gespeichert, bis das erste Komma empfangen wird. Da der aktuelle Zustand NEW ist, beinhaltet der Puffer den Befehl, in unserem Fall `GPFOO`. Jetzt wechselt die Maschine nach `FOO` und setzt den Parameterindex auf 1. Bis zum zweiten Komma werden die nächsten Zeichen wieder in den Puffer geschrieben. Jetzt beinhaltet der Puffer den Wert `param1` und kann vom Parser über den Parameterindex dem Befehl `Foo` mit dem Index 1 zugeordnet werden. Eine weitere Funktion wertet diesen Parameter jetzt aus und aktualisiert die globale Datenstruktur. Beim nächsten Trennzeichen befindet sich diesmal `param2` im Puffer und wird wieder von der eben genannten weiteren Funktion, diesmal als Parameter 2, ausgewertet. Da es jedoch der letzte interessante Parameter war, wird die Maschine wieder in den `INVALID`-Zustand geschickt und ignoriert somit alle weiteren Zeichen bis zum nächsten Befehlsstart, dem „$“.

4.4.2.5. IMPLEMENTIERUNG KRITISCHER STELLEN

Neben dem recht komplizierten Aufbau der Maschine und des Parsers ist vor allem ein Faktor sehr entscheidend: Die Geschwindigkeit! Keine Berechnung darf auch nur annähernd so lange dauern, bis ein neues Zeichen von der seriellen Schnittstelle kommt.

Neben der Auswertung der Zeichen und Zuordnung zu dem jeweiligen Status, Befehl und Parameter, die nicht vermeidbar ist, geschieht noch ein weiterer, weitaus zeitraubenderer Vorgang: Das Parsen eines neuen Parameters und anschließende Aktualisieren der globalen Struktur.

Hier müssen Strings in Zahlenwerte umgewandelt werden. Dabei treten verschiedene Situationen auf:

Die einfachste Umwandlung ist die mit nur einer Ziffer:

```
// converts a 1 digits long value (a char) into a byte
#define     str1_to_byte(idx)       g_GPS_strIn[idx] - '0'
```

Listing 17 – GPS, str1_to_byte

Die nächst größere Version ist die mit zwei Ziffern: Auch hier kann noch sicher von einem BYTE ausgegangen werden:

```
// converts a 2 digits long value into a byte
_private BYTE str2_to_byte( BYTE idx )
{
    char    *str = &g_GPS_strIn[idx];
    return ( str[0] - '0' ) * 10 + str[1] - '0';
}
```

Listing 18 – GPS, str2_to_byte

Bei drei Stellen, muß schon ein 16-Bit Wertebereich her:

```
// converts a 3 digits long value into a word
_private WORD str3_to_word( BYTE idx )
{
    char    *str = &g_GPS_strIn[idx];
    return ( str[0] - '0' ) * 100 + ( str[1] - '0' ) * 10 + str[2] - '0';
}
```

Listing 19 – GPS, str3_to_word

Diese Funktionen müssen so of wie möglich verwendet werden. Nur wenn keine andere Möglichkeit besteht, kann die universelle Funktion eingesetzt werden:

```
// converts a variable length value into a short with handling of the sign
_private short str_to_short( BYTE idx )
{
    char    *str = &g_GPS_strIn[idx];

    // check for neg sign
    BOOL    bNeg = FALSE;
    if ( *str == '-' )
    {
        // mark to negate result later
        bNeg = TRUE;
        // inc string pointer one char to ignore sign char
        str++;
    }

    // read value
    short val = 0;
    // assume a maximum of 5 chars (its a short!)
    for ( idx = 0; idx < 5; idx++ )
    {
        // abort if not in numeric char range (0..9)
        if ( ( *str < '0' ) || ( *str > '9' ) ) // || ( *str == 0 ))
            goto _Return;

        // shift current val one digit ( *= 10 ) and add new digit
        val = val * 10 + (WORD)(*str++ - '0');
    }

_Return:
    // return result or negated result
    return bNeg ?       -val : val;
}
```

Listing 20 – GPS, str_to_short

Beispielsweise wird folgender Breitengrad empfangen:
`4816.8350`

Anstatt diesen Wert nun als einen ganzen mit 4816.8350 einzulesen und wohlmöglich noch in einem `double` zu speichern, muß er möglichst gut aufgeteilt werden. In diesem Fall in diese drei Werte:

```
48xxxxxxx
xx16xxxxx
xxxxx835x
```

Der erste Wert ist direkt der Breitengrad und wird in einem Byte gespeichert. Der zweite ist die ganze Anzahl der Sekunden. Der dritte Wert sind die ersten drei Nachkommastellen. Die letzte kann getrost ignoriert werden, da die Genauigkeit des Empfängers solche Werte sowieso nicht ermöglicht. Damit werden anstelle einer großen nur drei einfache Konvertierungen angestoßen. Allerdings muß bei so einem Fall der Aufbau streng vorgeschrieben sein. Es müssen also immer die ersten beiden Ziffern der Breitengrad sein, usw. Auch darf bei einem einstelligen Breitengrad das Komma nicht verrutschen. Wird zum Beispiel der Breitengrad 3 ausgegeben, so wird er korrekt in 03 umgewandelt. In der NMEA-Spezifikation ist das für die meisten Parameter aber so vorgesehen. Nur die Anzahl der Nachkommastellen kann variieren. Solange sie aber nicht kleiner als drei werden, stört das unseren Algorithmus herzlich wenig.

Als einer der wenigen hartnäckigen Ausnahmen hat sich die Höhe herausgestellt, die leider nicht mit führenden Nullen aufgefüllt wird und somit die vollständige Konvertierungsfunktion benötigt.

4.4.2.6. BEISPIELVERWENDUNG UND TESTS

Ein einfacher Test:

```
…
GPS_InitPort();
…
GPS_Init();

GPS_Power_GPS( ON );
GPS_SelectChannel( GPS_CHANNEL_GPS_CMOS );

while( 1 )
{
    if ( GPS_NewDataAvailable() )
    {
        char    str[72];
        sprintf( str, "Time: %02d:%02d:%02d", GPS_DATA.byTimeHour, GPS_DATA.byTimeMinute,
                      GPS_DATA.byTimeSecond );
        LCD_PutStringXY( 0, 0, str, true );

        sprintf( str, "Date: %02d.%02d.20%02d", GPS_DATA.byDateDay, GPS_DATA.byDateMonth,
                      GPS_DATA.byDateYear );
        LCD_PutStringXY( 20, 0, str, true );


        sprintf( str, "Lat./Lon.: %02d°%02d.%02d' %c    %03d°%02d.%02d' %c", GPS_DATA.byLatitudeDegree,
                      GPS_DATA.byLatitudeSeconds, GPS_DATA.byLatitudeDot,
                      GPS_DATA.byLatitudeDirection, GPS_DATA.byLongitudeDegree,
                      GPS_DATA.byLongitudeSeconds, GPS_DATA.byLongitudeDot,
                      GPS_DATA.byLongitudeDirection );
        LCD_PutStringXY( 0, 2, str, true );

        sprintf( str, "Altitude: %d    Speed: %d", GPS_DATA.wAltitude, GPS_DATA.wSpeed_10th );
        LCD_PutStringXY( 0, 3, str, true );

        sprintf( str, "Quality: %d    Number Of Sattelites: %d", GPS_DATA.byQuality,
                      GPS_DATA.byNumberOfSatellites );
        LCD_PutStringXY( 0, 5, str, true );
    }
}
```

Listing 21 – GPS, Test

Nachfolgend wieder ein paar Bilder:
Die direkt empfangenen NMEA-Datensätze sieht man auch im letzten Abschnitt bei den Fotos für das LCD-Modul.

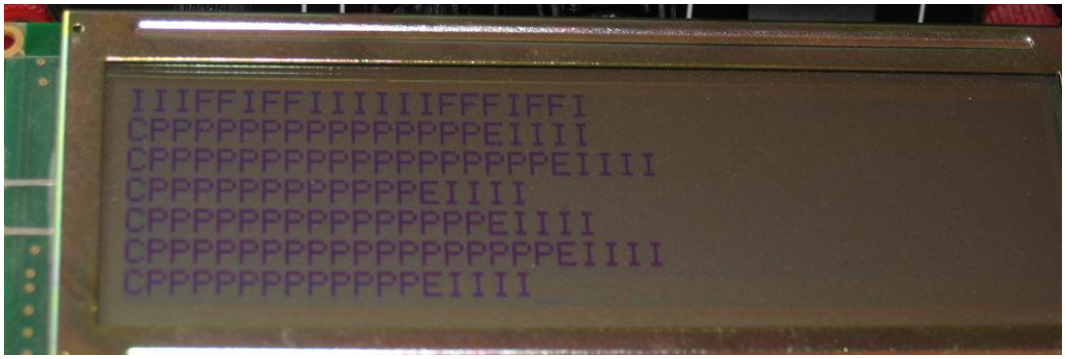


Abbildung 138 – LCD, GPS-Parserausgabe

Hier Testausgaben des Parsers. `I=INVALID; F=FRAMEERROR; C=NEW_CMD; P=PARAMETER; E=END;`

Am Anfang wurde die Datenleitung aufgesteckt und durch absichtliches Reiben an den Steckverbindungen ein Übertragungsfehler simuliert, was an den `F`-Zuständen erkennbar ist. Danach wird korrekt ein neuer Befehl mit `C` erkannt und die einzelnen Parameter, jeweils mit `P`, ausgewertet. Ganz zum Ende des Befehls wird ein `E` ausgegeben und bei den restlichen Daten befindet sich die Maschine wieder im `I`-Zustand.

Abbildung 139 – LCD, GPS-Informationen

Hier sieht man einen Teil der Daten aus der globalen Struktur.

4.4.2.7. BERECHNUNGEN, ANALYSEN UND STATISTIKEN

Eine Aufstellung des ungefähr benötigten Speichers inklusive der Werte für den Timer:

Speicherart	Größe [Bytes]
Programmcode	2292
RAM	65

Tabelle 45 – LCD-Modul Speicherverbrauch

4.4.3. MMC

4.4.3.1. AUFGABE

Auch hier sind wieder die beiden Init-Funktionen nötig. Zusätzlich aber noch eine dritte, mit der man die eingelegte Speicherkarte erkennen und zur Verwendung vorbereiten kann, damit auch während dem Betrieb die Karte gewechselt werden kann.

Die Zugriffsarten auf die Karte müssen in zwei Kategorien unterteilt werden: Zum einen den direkten, kompletten Sektorzugriff, bei dem immer die vollen 512 Bytes gelesen oder geschrieben werden. Zum anderen die Byte-orientierten Zugriffe, mit denen man aus einem Sektor, ab einem Offset, eine beliebige Anzahl an Bytes lesen kann. Beim Schreiben können Daten innerhalb eines Sektors nicht direkt angesprochen werden, deshalb soll hier wenigstens ein stückweiser Transfer zur Karte möglich sein. Laut Datenblatt kann die Schreibgeschwindigkeit beliebig langsam sein.

4.4.3.2. HARDWAREKONFIGURATION

Hier kommen nur die vom Controller vorgesehenen vier Pins für die SPI-Schnittstelle zum Einsatz:

```
//-------------------------------------------------------------------------------------------
// ### hardware config
//-------------------------------------------------------------------------------------------
//-- spi port defines ---------------------------------------------------------------------
#define     MMC_SPI_PORT            PORTB       // spi port register
#define     MMC_SPI_PORT_IN         PINB        // spi port in register
#define     MMC_SPI_PORT_DDR        DDRB        // spi port direction register

#define     MMC_SPI_BIT_SS          4           // #fixed: bit in spi port for </chip select>
#define     MMC_SPI_BIT_SCK         7           // #fixed: bit in spi port for <serial clock>
#define     MMC_SPI_BIT_MOSI        5           // #fixed: bit in spi port for <MMC DI>
#define     MMC_SPI_BIT_MISO        6           // #fixed: bit in spi port for <MMC DO>
//-------------------------------------------------------------------------------------------
```

Listing 22 – MMC, Hardware Config

4.4.3.3. INTERFACE

```
//-- Init ---------------------------------------------------------------------------------
    void        MMC_PortInit( void );
    void        MMC_Init( BYTE bySpeedMode );

    MMC_RET     MMC_InitCard( void );
    MMC_RET     MMC_GetCSD( tCSD *pCSD );
//-------------------------------------------------------------------------------------------

//-- full sector operations ---------------------------------------------------------------
    MMC_RET     MMC_ReadSector  ( void *pBuffer, DWORD lSector );
    MMC_RET     MMC_WriteSector ( const void *pbyBuffer, DWORD lSector );
    MMC_RET     MMC_ZeroSector  ( DWORD lSector );
//-------------------------------------------------------------------------------------------

//-- partial block operations -------------------------------------------------------------
    MMC_RET     MMC_Read        ( void *pbyBuffer, DWORD lSector, WORD wStart, WORD wLen );

    MMC_RET     MMC_Write_Open  ( DWORD lSector );
    MMC_RET     MMC_Write_Data  ( const void *pbyBuffer, WORD wLen );
    MMC_RET     MMC_Write_Close ( void );
//-------------------------------------------------------------------------------------------

//-- error message handling ---------------------------------------------------------------
    PGM_P       MMC_GetErrorText( MMC_RET byRet );
//-------------------------------------------------------------------------------------------
```

Listing 23 – MMC, Interface

4.4.3.4. AUFBAU DES MODULTREIBERS

Der große Unterschied zu allen anderen Modul-Treibern ist beim MMC-Treiber, daß hier auch Fehler auftreten können. Bei allen anderen Modulen kann immer davon ausgegangen werden, daß die Funktion einwandfrei ausgeführt wurde. In diesem Modul aber spielen der Benutzer und die Speicherkarte mit. So kann entweder gar keine Karte im Gerät sein, der Benutzter die Karte unbemerkt entfernen, der Speicherplatz nicht mehr ausreichen oder die Karte einfach sonst einen Fehler zurückgeben. Um das Problem zu lösen, werden in allen Funktionen Rückgabewerte eingeführt, die über den Status des Aufrufs informieren:

```
//-- return status defines ----------------------------------------------------------------
typedef     BYTE                    MMC_RET;    // return value

#define     MMC_OK                  0           // OK
#define     MMC_FAILED_NOCARD       1           // FAILED: No card detected
#define     MMC_FAILED_TIMEOUT      2           // FAILED due timeout
#define     MMC_FAILED              3           // FAILED
#define     MMC_SECTORFULL          4           // FAILED due writing data (more than 512 bytes)
```

Listing 24 – MMC, Return Values

Zuerst, wie immer, die Initialisierungen:

- **`void MMC_PortInit( void );`**
  Die Richtung der vier SPI-Pins setzen.

- **`void MMC_Init( BYTE bySpeedMode );`**
  Der Parameter kann `MMC_SPEED_SAFE`, `MMC_SPEED_FAST` oder `MMC_SPEED_EXTREME` annehmen und gibt die zu verwendende Geschwindigkeit des SPI-Busses an. Safe ist f/128,

Fast ist f/4 und Extreme ist f/2. Je nach Prozessortakt und verwendeter Leitungslänge sollte man hier entweder Fast oder Extreme wählen.

- **`MMC_RET MMC_InitCard( void );`**
  Hier wird streng nach Spezifikation vorgegangen: Erst mindestens 74 Bits zum Initialisieren der Karte senden, anschließend bis zu zehn Mal `CMD0` (Reset) ausführen. Falls die Karte noch nicht mit einer Erfolgsmeldung reagiert hat, wird einfach davon ausgegangen, daß keine Karte im Slot ist und deshalb `MMC_FAILED_NOCARD` zurückgegeben.

  Danach wird bis zu 100 Mal versucht, das Kommando CMD1 (Init) abzuschicken, jeweils mit einer Millisekunde Abstand, damit die Karte genug Zeit für den Reset bekommt. Falls das nicht ausreicht, gibt die Funktion `MMC_FAILED_TIMEOUT` zurück und man kann von einem Hardwaredefekt ausgehen.

  Jetzt ist die Karte fertig initialisiert und bereit, weitere Kommandos zu empfangen. Zuletzt wird noch die Größe der Speicherkarte ermittelt und in die globale Variable `g_MMC_dwMaxSectors` abgespeichert.

- **`MMC_RET MMC_GetCSD( tCSD *pCSD );`**
  Diese Funktion liest die CSD-Daten von der MMC-Karte aus. Für Details siehe das Grundlagenkapitel im MMC-Abschitt oder direkt das Datenblatt zu MMC-Karten [MMC].

Jetzt würden die drei Funktionen zum Zugriff auf einen jeweils kompletten Sektor folgen. Da diese allerdings von den Byte-orientierten Zugriffen abhängig sind, werden sie vorgezogen:

Zum Einlesen von Daten:

- **`MMC_RET MMC_Read( void *pBuffer, DWORD lSector, WORD wStart, WORD wLen );`**
  Den Sektor `lSector` öffnen und in den Puffer `pbyBuffer` `wLen` Bytes ab dem Offset `wLen` einlesen.

  Dazu wird erst einmal die Sektoradresse nach Bytes umgerechnet, da die MMC-Karten nur mit auf 512 Byte ausgerichteten Adressen arbeiten. Anschließend schickt die Funktion den Befehl `MMC_CMD17_READ` an die Karte und liest den kompletten Sektor ein, überträgt aber erst ab `wStart` in den Puffer und zwar für genau `wLen` Bytes.

  Der folgende Abschnitt der behandelt den genauen Ablauf und die dafür benötigten weiteren Funktionen.

Das Schreiben ist etwas komplizierter, da hier kein Random-Access innerhalb eines Sektors möglich ist:

- **`MMC_RET MMC_Write_Open( DWORD lSector );`**
  Das Schreiben wird damit auf den ausgewählten Sektor initialisiert, indem die Funktion den Befehl MMC_CMD24_WRITE – mit der ebenfalls umgerechneten Adresse – ausführt. Zusätzlich wird der Zähler `g_MMC_wWrite_BytesLeft` für die restlich verfügbaren Bytes in diesem Sektor auf die anfänglichen 512 Bytes gesetzt. Der Zugriff auf die MMC-Karte bleibt nun aktiv. Jetzt darf man nur noch die `Write_Data(…)` und die `Write_Close(…)`-Funktion aufrufen.

- **`MMC_RET MMC_Write_Data( const void *pBuffer, WORD wLen );`**
  Jetzt kann beliebig oft die Funktion `Write_Data(…)` mit variablen Datenmengen aufgerufen werden, jedoch selbstverständlich nicht über die 512 Bytes Grenze hinaus. Im Anschluß wird der Zähler für die restlichen Bytes im Sektor aktualisiert.

- **`MMC_RET MMC_Write_Close( void );`**
  Falls der Sektor voll ist oder keine weiteren Daten mehr in diesen Sektor müssen, so ruft man diese Funktion auf. Damit wird der Befehl auch für die MMC-Karte abgeschlossen und dauerhaft gespeichert. Falls der Zähler `g_MMC_wWrite_BytesLeft` zu diesem Zeitpunkt noch nicht Null ist, also nicht alle 512 Bytes beschrieben wurden, füllt die Funktion die restlichen mit 0x00 auf.

Zuletzt noch mal auf die vorhin übergangenen Funktionen zum kompletten Sektorzugriff:

- **`MMC_RET MMC_ReadSector( void *pBuffer, DWORD lSector );`**
  Ruft einfach die Read-Funktion mit dem `lSector` auf und übergibt als Startadresse 0 und für die Länge 512.

- **`MMC_RET MMC_WriteSector( const void *pBuffer, DWORD lSector );`**
  Funktioniert nach demselben Prinzip, muß jedoch die drei Speicherfunktionen aufrufen.

- **`MMC_RET MMC_ZeroSector( DWORD lSector );`**
  Damit läßt sich ein kompletter Sektor mit Nullen füllen. Das funktioniert sehr einfach: Zuerst `Write_Open(…)` und danach gleich `Write_Close(…)`. Wie oben beschrieben, füllt die Funktion den Rest dann mit Nullen auf. In diesem Fall eben alles.

Fehlerinformationsfunktion:

- **`PGM_P MMC_GetErrorText( MMC_RET byRet );`**
  Diese Funktion wertet den übergebenen Fehlercode in einen beschreibenden String um und übergibt den Pointer auf die Programmcodestelle, in der sich der Text befindet.

Um die Verwendung dieser Funktionen etwas zu vereinfachen und sich die immer notwendigen Überprüfungen der Rückgabewerte – wenigsten schreibtechnisch – zu ersparen, sollte man folgendes Makro zum Aufruf der MMC-Funktionen benutzen:

```
#define     MMC_SAFE(stmt)          { MMC_RET ret;  if ( ( ret = (stmt) ) != MMC_OK )   return ret; }
```

Listing 25 – MMC, MMC_SAFE

Richtig formatiert sieht der eingefügt Block damit so aus:

```
#define MMC_SAFE(stmt)
{
    MMC_RET ret;
    if ( ( ret = (stmt) ) != MMC_OK )
        return ret;
}
```

Listing 26 – MMC, MMC_SAFE 2

Beispiele der Verwendung finden sich im nächsten Beispiel- und Testabschnitt.

4.4.3.5. IMPLEMENTIERUNG KRITISCHER STELLEN

Hier existieren leider recht viele kritische Stellen, vor allem für den Datenverkehr mit der Karte.

Die zentrale Funktion ist hier:

- **`_private inline BYTE __MMC_SPI_TransferByte( BYTE bySend );`**
  Sie ist als `inline` definiert, um keine Performance zu verlieren und schickt das zu sendende Byte an das Hardware-SPI-Ausgangsregister `SPDR`. Danach wird aktiv auf das Ende der Übertragung gewartet und das dabei gleichzeitig empfangene Byte zurückgeliefert. Damit kann diese Funktion sowohl zum Schreiben, als auch zum Lesen eingesetzt werden:

```
/*************************************************************************************************
| Handles SPI Tranfer
+-------------------------------------------------------------------------------------------------
| Send a byte to SPI and recieve one
|
| * INLINE!
+-------------------------------------------------------------------------------------------------
| @Params:
|   * bySend            byte to send
| @Return Value:
|   * Received byte
+************************************************************************************************/
_private inline     BYTE __MMC_SPI_TransferByte( BYTE bySend )
{
    // send data
    SPDR = bySend;
    // wait until data is sent completely
    while ( !( SPSR & (1<<SPIF) ) );
    // return received data
    return SPDR;
}
```

Listing 27 – MMC, __MMC_SPI_TransferByte

Die Kommunikation mit der Karte funktioniert über Befehle, von denen jeder über die privaten Funktionen `_MMC_CommandSequence_Start(…)` eingeleitet und über `_MMC_CommandSequence_Stop(…)` wieder beendet werden:

- **`_private BYTE _MMC_CommandSequence_Start( BYTE *pbyCommand );`**
  Das Kommando wird als gepacktes Array von Bytes mit folgendem Format, mit dem sie auch zur Karte übertragen werden müssen, übergeben: Das Array besteht aus sechs Bytes, das erste ist der Befehl, die nächsten vier beschreiben den Parameter und der letzte ist der CRC-Wert, der bis auf den Reset-Befehl immer Null sein kann. Anschließend wird die Rückmeldung (Response) der Karte abgewartet und zurückgegeben (vgl. Grundlagenkapitel, MMC):

```
/*************************************************************************************************
| Starting a MMC Command Sequence
+-------------------------------------------------------------------------------------------------
| * Select card
| * Send 6 bytes command
| * Wait until card responses
| _MMC_CommandSequence_Stop MUST be called after a command sequence!
+-------------------------------------------------------------------------------------------------
| @Params:
|   * pbyCommand:       6 byte command array (1 command, 4 param, 1 crc)
| @Return Value:
|   * Card response
+************************************************************************************************/
_private BYTE   _MMC_CommandSequence_Start( BYTE *pbyCommand )
{
    // select card
    MMC_SPI_CS_SELECT

    // send command (6 bytes: 1 command, 4 param, 1 crc)
    BYTE    i;
    for ( i=0; i<6; i++ )
        __MMC_SPI_TransferByte( *pbyCommand++ );

    // get card response
    BYTE    byRet;
    for ( i=0; i<10; i++ )          // 8 == max response time, 10 to go sure
        if ( ( byRet = __MMC_SPI_TransferByte( 0xff ) ) != MMC_RESPONSE_WAIT )
            break;
```

```
    // return response (0 == no error)
    return byRet;
}
```

Listing 28 – MMC, _MMC_CommandSequence_Start

Jetzt können die Nutzdaten des soeben eingeleiteten Befehls übertragen werden. Für Leseoperationen gibt es die Funktion `_MMC_CommandSequence_Read`:

- **`_private MMC_RET _MMC_CommandSequence_Read( BYTE *pbyBuffer, WORD wBlockSize, WORD wStart, WORD wLen );`**
  Hier wird zuerst auf den Blockstart laut Spezifikation gewartet, dann die einzelnen Bytes empfangen und je nach Parameter in den Ausgabepuffer geschrieben. Danach werden noch die beiden CRC-Bytes empfangen:

```
/*************************************************************************************************
| Reads a block in MMC Command Sequence
+-------------------------------------------------------------------------------------------------
| * Wait until block start byte detected
| * Read the specified amount of bytes
|
| MUST be called between Start and Stop of a MMC command sequence
+-------------------------------------------------------------------------------------------------
| @Params:
|   * pbyBuffer             Output array buffer to store data
|   * wBlockSize            Block size to read from card
|   * wStart                Start address for tranfering data into output array
|   * wLen                  number of bytes to tranfer into output array
| @Return Value:
|   * MMC_OK                OK
|   * MMC_FAILED_TIMEOUT    Timeout
+*************************************************************************************************/
_private MMC_RET    _MMC_CommandSequence_Read( BYTE *pbyBuffer, WORD wBlockSize, WORD wStart, WORD wLen )
{
    WORD    i = 0;
    BYTE    byData;

    // wait until block start byte detected
    while ( __MMC_SPI_TransferByte( 0xff ) != MMC_BLOCKSTART_SINGLEBLOCK_READ )
    {
        if ( ++i > 1000 )
            return MMC_FAILED_TIMEOUT;

        // wait a bit (max of 1000 * 10µs = ~10ms)
        _delay_us( 10. );
    }


    // calculate data end
    wLen += wStart;

    // read block from mmc
    // for complete block size
    for ( WORD wByte = 0; wByte < wBlockSize; wByte++ )
    {
        // read byte from card
        byData = __MMC_SPI_TransferByte( 0xff );

        // if in requested range copy it to output buffer
        if ( ( wByte >= wStart ) & ( wByte < wLen ) )
                *pbyBuffer++ = byData;
    }


    // read crc
    __MMC_SPI_TransferByte( 0xff );
    __MMC_SPI_TransferByte( 0xff );

    // all ok
    return MMC_OK;
}
```

Listing 29 – MMC, _MMC_CommandSequence_Read

Um einen Befehl abzuschließen, existiert die Funktion:

- **`_private void _MMC_CommandSequence_Stop( void );`**
  Vor dem Aufruf dieser Funktion müssen die Daten des Befehls vollständig übertragen sein. Um einen Befehl endgültig abzuschließen, selektiert man die Karte ab und sendet danach noch einmal acht Taktzyklen.

```
/*************************************************************************************************
| Finish a MMC Command Sequence
+-------------------------------------------------------------------------------------------------
| * Unselect card
| * Send clock
|
| MUST be called after a MMC command sequence
+-------------------------------------------------------------------------------------------------
| @Params:          none
| @Return Value:    none
+*************************************************************************************************/
_private void   _MMC_CommandSequence_Stop( void )
{
    // unselect card
    MMC_SPI_CS_UNSELECT

    // send 8 clocks to finish
    __MMC_SPI_TransferByte( 0xff );
}
```

Listing 30 – MMC, _MMC_CommandSequence_Stop

Um nun einen Sektor oder Teile daraus, einzulesen, verwendet man die oben schon beschriebene Funktion `MMC_Read`:

- **`MMC_RET MMC_Read( void *pBuffer, DWORD lSector, WORD wStart, WORD wLen );`**

  Die Vorgehensweise dieser Funktion ist oben schon beschrieben. Hier zur Verdeutlichung noch der Quellcode:

```
/*************************************************************************************************
| Reads data from a sector from the MMC Card
+-------------------------------------------------------------------------------------------------
+-------------------------------------------------------------------------------------------------
| @Params:
|   * pBuffer               Output array buffer to store data
|   * lSector               sector number to read from card
|   * wStart                Start address for tranfering data into output array
|   * wLen                  number of bytes to tranfer into output array
| @Return Value:
|   * MMC_OK                OK
|   * MMC_FAILED_TIMEOUT    Timeout
|   * MMC_FAILED            Command failed
+*************************************************************************************************/
_public MMC_RET     MMC_Read( void *pBuffer, DWORD lSector, WORD wStart, WORD wLen )
{
    BYTE    byRet;

    // convert pointer to BYTE[]
    BYTE    *pbyBuffer =  (BYTE*) pBuffer;


    // calc address out of sector ( => * 512, but do only <<1 and add one 0x00 byte )
    lSector <<= 1;

    // read single block
    BYTE    byCommand[] = { MMC_CMD17_READ, (BYTE)(lSector>>16), (BYTE)(lSector>>8), (BYTE)(lSector), 0x00,
                            MMC_CMD_CRC };

    // start read sequence
    if ( _MMC_CommandSequence_Start( byCommand ) != MMC_RESPONSE_OK )
    {
        byRet = MMC_FAILED;
        goto _Exit;
    }

    // read data (blocksize is always 512)
    byRet = _MMC_CommandSequence_Read( pbyBuffer, MMC_SECTORSIZE, wStart, wLen );

_Exit:
    // finish read sequence
    _MMC_CommandSequence_Stop();
    return byRet;
}
```

Listing 31 – MMC, MMC_Read

Das Schreiben mit den Befehlen `MMC_Write_Open(…)`, `MMC_Write_Data(…)` und `MMC_Write_Close(…)` ist noch mal ein Stück komplizierter, funktioniert aber nach dem gleichen Prinzip der Lesefunktion. Außerdem muß noch der Response-Typ R1b etwas anders behandelt werden (siehe Grundlagenkapitel). Bei Interesse sei hier noch mal auf den Quellcode und die Spezifikation [MMC] verwiesen.

4.4.3.6. Beispielverwendung und Tests

Um den Zugriff auf die Speicherkarte und damit die Verwendung des Treibers besser zu verstehen eignet sich am besten ein Beispiel:

```
// init driver
MMC_PortInit();
MMC_Init( MMC_SPEED_EXTREME );

// init card
MMC_SAFE( MMC_InitCard( MMC_SPEED_FAST ) );

// sector data
BYTE    abyB[ 512 ];

// sector access
MMC_SAFE( MMC_ReadSector( abyB, THE_SECTOR_NUMBER ) );
ChangeData( abyB );
MMC_SAFE( MMC_WriteSector( abyB, SECTOR_NUMBER ) );
MMC_SAFE( MMC_ZeroSector( THE_SECTOR_NUMBER + 1 ) );

// offset access
DWORD   dwValue1 = 123456;
DWORD   dwValue2 = 654321;

MMC_SAFE( MMC_Write_Open( THE_SECTOR_NUMBER ) );
MMC_SAFE( MMC_Write_Data( &dwValue1, sizeof( dwValue1 ) ) );
MMC_SAFE( MMC_Write_Data( &dwValue2, sizeof( dwValue2 ) ) );
MMC_SAFE( MMC_Write_Close() );

DWORD   dwValue3;
MMC_SAFE( MMC_Read( &dwValue3, THE_SECTOR_NUMBER, THE_OFFSET, sizeof( dwValue3 ) ) );
```

Listing 32 – MMC, Test 2

Mit direkter Auswertung, also ohne `MMC_SAFE(...)` könnten die Aufrufe so aussehen:

```
MMC_RET ret;

ret = MMC_InitCard( MMC_SPEED_FAST ) );
if ( ret != MMC_OK )
{
    switch ( ret )
    {
        case MMC_FAILED_NOCARD:
            break;

        case MMC_FAILED_TIMEOUT:
            break;

        case MMC_FAILED:
            break;

        default:
            break;
    }

    return SOME_ERROR_VALUE;
}
```

Listing 33 – MMC, Test, Direkte Auswertung

4.4.3.7. Berechnungen, Analysen und Statistiken

Hier eine Aufstellung des ungefähr benötigten Speichers für den MMC-Treiber:

Speicherart	Größe [Bytes]
Programmcode	1760
RAM	19

Tabelle 46 – MMC-Modul Speicherverbrauch

4.4.4. IO

4.4.4.1. AUFGABE

Dieser Treiber hat die am Anfang recht trivial erscheinende Aufgabe, den Status der Taster-Eingänge zu ermitteln und den Zustand der LEDs zu setzten.

Für die Ausgänge soll es nur zwei Befehlsvariationen geben: Eine zum gezielten Festlegen eines LED-Zustands und die andere zum Wechseln des aktuellen Zustands. Damit kann dann zum Beispiel Blinken sehr einfach realisiert werden, ohne daß sich der Aufrufer um den aktuellen Status der LED kümmern muß.

Bei der Eingabe sollen auch zwei Methoden zum Einsatz kommen, erstere erlaubt den direkten Zugriff auf den aktuellen Zustand der Taster. Wenn nun der Benutzer aber nur kurz eine Taste drückt und ausgerechnet in diesem Moment das Programm den Zustand nicht abfragt, so ist dieser Tastendruck verloren. Außerdem, falls zwei Tasten gleichzeitig gedrückt werden, ist so ebenfalls nicht mehr feststellbar, welche Taste der Benutzer zuerst gedrückt hat.

Aus diesen Problemen ergibt sich die zweite Methode, bei der eine Nachrichtenschleife zum Einsatz kommt, die das serielle Auslesen und Auswerten der Nachrichten ermöglicht.

4.4.4.2. HARDWAREKONFIGURATION

Im Treiber können die Ein- und Ausgangsleitungen frei definiert werden. Er geht nur davon aus, daß sich die Pins fortlaufend an einem Port befinden, also ohne Unterbrechung. Die Anzahl und das Startbit lassen sich ebenso frei definieren, als auch ob die Ausgangsleitungen invertiert sind, also bei 0 aktiv:

```
//------------------------------------------------------------------------------------------
// ### hardware config
//------------------------------------------------------------------------------------------
// KEY port defines
// Example: To use PORTC bits 0 to 3 you have to set STARTBIT = 0 and BITCOUNT = 4
#define     IO_KEY_PORT             PORTC       // key port register
#define     IO_KEY_PORT_IN          PINC        // key port in register
#define     IO_KEY_PORT_DDR         DDRC        // key port data direction register
#define     IO_KEY_PORT_STARTBIT    0           // key port start bit number (0..7)
#define     IO_KEY_PORT_BITCOUNT    4           // key port bit count (1..7)
#define     IO_KEY_ENABLE_MESSAGEMAP            // if you want to use the key message map

// LED port defines
// Example: To use PORTC bits 4 to 7 you have to set STARTBIT = 4 and BITCOUNT = 4
#define     IO_LED_PORT             PORTC       // key port register
#define     IO_LED_PORT_DDR         DDRC        // key port data direction register
#define     IO_LED_PORT_STARTBIT    4           // key port start bit number (0..7)
#define     IO_LED_PORT_BITCOUNT    2           // key port bit count (1..7)
//#define   IO_LED_INVERTED                     // defined: 0=>ON 1=>OFF, else comment out
//------------------------------------------------------------------------------------------
```

Listing 34 – IO, Hardware Config

4.4.4.3. INTERFACE

```
//-- Init and start ------------------------------------------------------------
    void IO_PortInit( void );
    void IO_Init( void );
//------------------------------------------------------------------------------

//-- Key -----------------------------------------------------------------------
    BOOL IO_Key_IsPressed( BYTE byKeyNo );
    BOOL IO_Key_IsMsgAvailable( void );
    BYTE IO_Key_GetMsg( void );
//------------------------------------------------------------------------------

//-- LED -----------------------------------------------------------------------
    void IO_LED_Set( BYTE byLEDNo, BOOL bOn );
    void IO_LED_Toggle( BYTE byLEDNo );
//------------------------------------------------------------------------------
```

Listing 35 – IO, Interface

4.4.4.4. AUFBAU DES MODULTREIBERS

Zuerst wieder die Initialisierungsfunktionen:

- **`void IO_PortInit( void );`**
  Setzt die Taster-Pins auf Eingang und aktiviert die Pull-Up-Widerstände. Ebenso setzt die Funktion die LED-Leitungen auf Ausgang.

- **`void IO_Init( void );`**
  Initialisiert die Nachrichtenschleife, läßt die LEDs zum Test kurz aufblinken und startet den Timer, der sich um die Eingabe der Tasten und die Verwaltung der Nachrichtenschleife kümmert.

Die Zuordnung der Tasten und LEDs geschieht über folgende Definitionsliste:

```
//------------------------------------------------------------------------------
// Key and LED globals name defines
//------------------------------------------------------------------------------
#define     IO_KEY_UP               1
#define     IO_KEY_DOWN             0
#define     IO_KEY_MENUOK           2
#define     IO_KEY_ESC              3
#define     IO_KEY_NONE             0xff

#define     IO_LED_POWER            0
#define     IO_LED_REC              1
//------------------------------------------------------------------------------
```

Listing 36 – IO, Zuordnungsliste

Zuerst die einfachen LED-Funktionen:

- **`void IO_LED_Set( BYTE byLEDNo, BOOL bOn );`**
  Schaltet die angegebene LED ein oder aus. `byLEDNo` kann hier Werte von 0 für die erste LED bis hin zu `IO_LED_PORT_BITCOUNT-1` für die letzte annehmen. Die entsprechenden Makros sind in der Headerdatei definiert.

- **`void IO_LED_Toggle( BYTE byLEDNo );`**
  Funktioniert nach dem gleichen Prinzip, nur wird hier der Zustand geändert. Eine zuvor ausgeschaltete LED wird eingeschalt und umgekehrt.

Jetzt die Funktionen bezüglich der Eingabe: Hierum kümmert sich der IO-Timer, der alle 50ms aufgerufen wird. Er schreibt den aktuellen Zustand der Taster in die globale Variable `g_IO_byKeyStatus` und aktualisiert bei Veränderungen die Nachrichtenschleife.

Für diese setzt der Treiber einen Ringpuffer ein. Er stellt sofort, wenn er einen Tastendruck erkennt, hinten einen neuen Eintrag in die Schleife ein. Die Anwendung, die den Treiber verwendet, kann nun auch in unregelmäßigen Abständen die Ereignisse abfragen und bekommt eine strikte Reihenfolge der Tastendrücke zurückgeliefert. Also in der Art wie `getch(…)` für die Tastatur.

Der Timer an sich wird noch einmal ausführlich im nächsten Abschnitt behandelt.

- **`BOOL IO_Key_IsPressed( BYTE byKeyNo );`**
  Das ist die direkte Methode, um den aktuellen Status des angegeben Tasters abzufragen. Die Funktion prüft nur, ob das entsprechende Bit in `g_IO_byKeyStatus` – dem Abbild des aktuellen Tastenstatus – gesetzt ist und liefert das Ergebnis zurück.

- **`BOOL IO_Key_IsMsgAvailable( void );`**
  Prüft, ob sich Nachrichten in der Nachrichtenschleife befinden. Wenn ja, dann existieren noch unbehandelte Tastendrücke in der Schleife.

- **`BYTE IO_Key_GetMsg( void );`**
  Gibt den nächsten Tastendruck aus der Nachrichtenschleife zurück und entfernt diesen daraus. Der Wert entspricht direkt der gedrückten Taste aus obiger Zuordnungsliste. Falls keine Nachricht mehr vorhanden ist, gibt die Funktion `IO_KEY_NONE` zurück. Das bedeutet, daß die Funktion auf jeden Fall sofort zurückkehrt und nicht aktiv auf einen Tastendruck wartet.

4.4.4.5. Implementierung kritischer Stellen

Nachrichtenschleife

Zuerst einmal ist der Aufbau der Nachrichtenschleife von Bedeutung:

```
#define             IO_KEYRINGBUFFER_SIZE       8           // definge ring size as 2^3
volatile BYTE       g_IO_abyKeyRingBuffer[IO_KEYRINGBUFFER_SIZE];
volatile BYTE       g_IO_byKeyRingBuffer_Store;
volatile BYTE       g_IO_byKeyRingBuffer_Load;
volatile BYTE       g_IO_byKeyRingBuffer_Size;
```

Listing 37 – IO, Definition Nachrichtenschleife

Die Schleife `g_IO_abyKeyRingBuffer` kann also maximal acht aufeinanderfolgende Tastendrücke zwischenspeichern. Die drei Variablen `Store`, `Load` und `Size` referenzieren Positionen im Ringpuffer: Die Position zum Schreiben der nächsten einkommenden Nachricht, zum Laden der nächsten noch nicht behandelten Nachricht und die Anzahl der unbehandelten Nachrichten in der Schleife.

Folgender Algorithmus liest eine neue Nachricht aus der Nachrichtenschleife aus:

```
/*************************************************************************************************
| Get next Message
|---------------------
| Get next pressed key number
+------------------------------------------------------------------------------------------------
| @Params:          none
| @Return Value:
|   Message         Number identifies pressed key number (None -> IO_KEY_NONE)
+************************************************************************************************/
_public BYTE IO_Key_GetMsg( void )
{
    // assume no msg
    BYTE    bRet = IO_KEY_NONE;

    // msg available?
    if ( g_IO_byKeyRingBuffer_Size > 0 )
    {
        // read key no from buffer
        bRet = g_IO_abyKeyRingBuffer[ g_IO_byKeyRingBuffer_Load++ ];
        // make sure ring buffer will stay in bounds
        g_IO_byKeyRingBuffer_Load &= IO_KEYRINGBUFFER_SIZE - 1;
        // decrease buffer size
        g_IO_byKeyRingBuffer_Size--;
    }

    // return result
    return bRet;
}
```

Listing 38 – IO, IO_Key_GetMsg

Falls eine neue Nachricht vorhanden ist, liest die innere `if`-Anweisung sie aus, aktualisiert die Index-Variablen und gibt anschließend die Nachricht zurück.

TIMER

Jetzt zum Timer: Die verwendete Intervallzeit von 50 Millisekunden erwies sich in der Praxis als guter Mittelwert. Es treten keine Prellungen mehr auf und auch schnelle Tastendrücke werden einzeln erkannt.

Zur Information:

Tastenprellung: Beim Drücken eines Tasters entsteht manchmal kein einmaliger Übergang zwischen den beiden Zuständen sondern vereinzelt mehrmalige. So kann eine schnelle Abfrage eines Tasters von 0 auf 1 etwa so aussehen: 0000000000110000011011111111111111111111. Dieses Phänomen kommt von der Mechanik: Beim Schließen prallen die Schalterkontakte aufeinander und federn wieder auseinander, bevor sie einen endgültigen Kontakt ergeben.

Die leicht vereinfachte Timerfunktion sieht so aus:

```
//-------------------------------------------------------------------------------------------
// Key Handler (Timer)
//-------------------------------------------------------------------------------------------
_private void IO_Key_Timer( void )
{
    // read new key status (and transform according to STARTBIT and BITCOUNT)
    BYTE    byKeyStatus = ( (~IO_KEY_PORT_IN) >> IO_KEY_PORT_STARTBIT ) & ( ( 1<<IO_KEY_PORT_BITCOUNT )-1);

    // reading old key status from volatile var
    BYTE    byKeyStatus_Old = g_IO_byKeyStatus;

    // set now-valid key status
    g_IO_byKeyStatus = byKeyStatus;

    // what we have:
    // byKeyStatus      => current key status
    // byKeyStatus_Old  => last key status

    // current key index (bit)
    BYTE    byKeyNo = 0;

    // do as long some key is still pressed
    while ( byKeyStatus )
    {
        // if CURRENT keys is pressed
        if ( byKeyStatus & 0x01 )
        {
            // if OLD key was not pressed
            if ( !( byKeyStatus_Old & 0x01 ) )
            {
                // a new key was pressed, so add it in the ring buffer
                // still space in the buffer? else --> key lost
                if ( g_IO_byKeyRingBuffer_Size < IO_KEYRINGBUFFER_SIZE )
                {
                    // add key no into buffer
                    g_IO_abyKeyRingBuffer[ g_IO_byKeyRingBuffer_Store++ ] = byKeyNo;
                    // make sure ring buffer will stay in bounds
                    g_IO_byKeyRingBuffer_Store &= IO_KEYRINGBUFFER_SIZE - 1;
                    // increase buffer size
                    g_IO_byKeyRingBuffer_Size++;
                }
            }
        }

        // shift to next key bit
        byKeyStatus >>= 1;
        byKeyStatus_Old >>= 1;
        byKeyNo++;
    }
}
```

Listing 39 – IO Timer

Um zu erkennen, ob sich seit dem letzten Aufruf der Status geändert hat, wird der alte Zustand in `byKeyStatus_Old` gespeichert. Für jeden möglichen Taster überprüft der Timer nun folgendes: Ist der aktuelle Taster gedrückt und war es das letzte Mal noch nicht, so wird er der Nachrichtenschleife hinzugefügt, falls diese noch den entsprechenden Platz bietet.

4.4.4.6. Beispielverwendung und Tests

```
// init
IO_PortInit();
IO_Init();

// LEDs
IO_LED_Set( IO_LED_REC, ON );

while ( TRUE )
{
    IO_LED_Toggle( IO_LED_POWER );
    WaitABit();
}

// keys
while ( IO_Key_IsMsgAvailable() )
{
    BYTE    byKey;
    byKey = IO_Key_GetMsg();

    HandleNextKeyPress( byKey );
}
```

Listing 40 – IO, Test

4.4.4.7. BERECHNUNGEN, ANALYSEN UND STATISTIKEN

Hier eine Aufstellung des ungefähr benötigten Speichers des IO-Treibers inklusive des dafür benötigen Timers:

Speicherart	Größe [Bytes]
Programmcode	1050
RAM	36

Tabelle 47 – IO-Modul Speicherverbrauch

4.4.5. PWR

4.4.5.1. AUFGABE

Zum einen hat der PWR-Moultreiber die Aufgabe die Stromversorgung aufrecht zu halten und einen niedrigen Batteriestand zu erkennen und daraufhin das LoBAT-Signal auszugeben. Aber – im Gegensatz zu allen anderen Treibern – greift er selbst nicht nur auf sein Hardware-Modul, sondern auch auf andere Treiber zurück: In diesem Fall auf die LCD- und IO-Treiber.

Zum anderen ist es aber auch die Aufgabe des Treibers, den Powerknopf auszuwerten und entsprechend zu behandeln.

Das Modul stellt eine zentrale Funktion zur Verfügung, die im IDLE-Zustand immer wieder aufgerufen werden sollte. Diese überprüft die Power- und LoBAT-Zustände und liefert die Ergebnisse zurück.

4.4.5.2. HARDWAREKONFIGURATION

Das PWR-Modul besitzt drei voneinander unabhängige IO-Leitungen. Dabei kann jeweils auch ausgewählt werden, ob die Leitung invertiert ist, also aktiv-LOW ist und ob bei den Eingängen zusätzlich der Pull-Up-Widerstand im Controller aktiviert werden soll:

```
//------------------------------------------------------------------------------------------
// ### hardware config
//------------------------------------------------------------------------------------------
// Example: To use PORTB bits 4 set PORT = xxxB and BIT = 4

// keep [out]
#define     PWR_KEEP_PORT           PORTB       // pwr keep port register
#define     PWR_KEEP_PORT_DDR       DDRB        // pwr keep data direction register
#define     PWR_KEEP_PORT_BIT       0           // pwr keep port bit
//#define   PWR_KEEP_INVERTED                   // pwr keep is inverted, else comment out


// pwrbut [in, int]
#define     PWR_PWRBUT_PORT         PORTD       // pwr button port register
#define     PWR_PWRBUT_PORT_DDR     DDRD        // pwr button data direction register
#define     PWR_PWRBUT_PORT_IN      PIND        // pwr button in register
#define     PWR_PWRBUT_PORT_BIT     2           // pwr button port bit
#define     PWR_PWRBUT_INT          0           // pwr button interrupt line
#define     PWR_PWRBUT_INVERTED                 // pwr button is inverted, else comment out
//#define   PWR_PWRBUT_USEPULLUP                // pwr button enable pullup resistor


// lobat [in]
#define     PWR_LOBAT_PORT          PORTC       // pwr lobat port register
#define     PWR_LOBAT_PORT_DDR      DDRC        // pwr lobat data direction register
#define     PWR_LOBAT_PORT_IN       PINC        // pwr lobat in register
#define     PWR_LOBATPORT_BIT       6           // pwr lobat port bit
#define     PWR_LOBAT_INVERTED                  // pwr lobat is inverted, else comment out
//#define   PWR_LOBAT_USEPULLUP                 // pwr lobat enable pullup resistor
//------------------------------------------------------------------------------------------
```

Listing 41 – PWR, Hardwarekonfiguration

4.4.5.3. INTERFACE

```
//-- Init -------------------------------------------------------------------------------
    void        PWR_PortInit( void );
    void        PWR_Init( void );
//---------------------------------------------------------------------------------------


//-- PWR --------------------------------------------------------------------------------
    PWRSTATUS   PWR_DoCyclicPowerManagement( void );
    void        PWR_ShutDown_MainPowerSupply( void );
    BOOL        PWR_IsPowerButtonPressed( void );
    BOOL        PWR_IsLobat( void );
//---------------------------------------------------------------------------------------
```

Listing 42 – PWR, Interface

4.4.5.4. AUFBAU DES MODULTREIBERS

Auch hier wieder die Initialisierungsfunktionen zu Beginn:

- **`void PWR_PortInit( void );`**
  Stellt die IO-Pins entsprechend der Hardwarekonfiguration ein und setzt das KEEP-Signal, um die Stromversorgung aufrecht zu halten.

- **`void PWR_Init( void );`**
  Initialisiert ein paar globale Variablen, wartet, bis die Powertaste nicht mehr gedrückt ist – damit dies nicht als Ausschaltsignal verstanden wird – und setzt zum Schluß einen Timer.

Jetzt sollen die Hilfsfunktionen besprochen werden:

- **`BOOL PWR_IsPowerButtonPressed( void );`**
  Liefert zurück, ob der Powerknopf zur Zeit gedrückt wird.

- **`BOOL PWR_IsLobat( void );`**
  Gibt ebenfalls einen BOOL-Wert zurück, der angibt, ob das LoBAT-Signal aktiv ist.

- **`Void PWR_ShutDown_MainPowerSupply( void );`**
  Zeigt eine Meldung an, daß das Gerät heruntergefahren wird und setzt dann das KEEP-Signal auf inaktiv. Somit unterbricht das PWR-Modul die Stromversorgung. Diese Funktion kehrt nicht zurück!

Und zum Abschluß noch die Hauptfunktion:

- **`PWRSTATUS PWR_DoCyclicPowerManagement( void );`**
  Sie sollte von den Applikationen im IDLE-Zustand ständig aufgerufen werden und prüft, ob LoBAT aktiv ist oder das Gerät ausgeschaltet werden soll. Anschließend liefert die Funktion diesen Wert dann zurück. Falls der Aufrufer ein `PWRSTATUS_SHUTDOWN` bekommt, sollte er möglichst schnell seine Aufräumarbeiten durchführen und sich anschließend beenden. Dieser Kreislauf setzt sich immer weiter nach oben fort, bis sich letztendlich auch das Hauptmenü beendet und die Kontrolle der `main(...)`-Funktion zurückgibt, die dann die Funktion für den finalen Shutdown aufruft.

4.4.5.5. Implementierung kritischer Stellen

In diesem Treiber existieren relativ wenig kritische Stellen. Um die geforderten Funktionen ausführen zu können, findet hier ein Timer Verwendung, der alle 200ms aufgerufen wird.

Power Button

Das Gerät soll nur abgeschaltet werden, wenn der Benutzer die Taste eine gewisse Zeit gedrückt hält. Damit können die Applikationen die Power-Taste auch noch für andere Zwecke einsetzen. Zum Beispiel kann der Anwender mit einem kurzen Tastendruck die Hintergrundbeleuchtung wieder einschalteten, falls diese im automatischen Zustand ist und nach einer bestimmten inaktiven Zeit von selbst ausgegangen ist.

Das verzögerte Ausschalten realisiert eine Zählvariable, die der Timer auf Null setzt, wenn die Power-Taste nicht gedrückt ist und erhöht, sobald die Taste gehalten wird. Erreicht die Variable nun einen gewissen Schwellwert, so setzt der Timer eine globale Variable `g_PWR_bPowerButton_RequestShutDown` auf TRUE. Der nächste Aufruf von `PWR_DoCyclicPowerManagement(…)` liefert nun als Ergebnis zurück, daß das Gerät heruntergefahren werden soll.

LoBat Anzeige

Die Anzeige für einen zu niedrigen Batteriestand wird mit Hilfe der Power-LED angezeigt. Diese leuchtet nach dem Bootvorgang durchgehend im normalen Betriebszustand. Sinkt die Batteriespannung unter den eingestellten Wert, so beginnt die LED zu blinken.

Um im Grenzfall der Spannung nicht ständig zwischen dem LoBat- und dem Normal-Zustand zu wechseln, wird folgender Mechanismus verwendet:

Sobald die Spannung unter den Schwellwert fällt, geht der Treiber in den LoBat-Zustand. Aber erst nachdem die Spannung mindestens für drei Sekunden konstant über dem Schwellwert liegt, kann der Treiber wieder zurück in den normalen Modus wechseln.

4.4.5.6. Beispielverwendung und Tests

Ein typischer Aufbau einer Applikation sieht so aus: Wichtig ist vor allem, daß bei `PWRSTATUS_SHUTDOWN` nach dem Aufräumen ein `return` folgt:

```
// loop
while ( TRUE )
{
    // Application work
    [...]

    // look for keypress (returns IO_KEY_NONE if none)
    switch ( IO_Key_GetMsg() )
    {
        [...]
    }

    // do power management
    switch ( PWR_DoCyclicPowerManagement() )
    {
        case PWRSTATUS_OK:
            break;

        case PWRSTATUS_LOBAT:
            break;

        case PWRSTATUS_SHUTDOWN:
            return;
    }
}
```

Listing 43 – PWR, Beispielanwendung

4.4.5.7. Berechnungen, Analysen und Statistiken

Hier eine Aufstellung des ungefähr benötigten Speichers für den PWR-Treiber. Allerdings sind hier die Werte für die benötigten LCD- und IO-Treiber mit dabei.

Speicherart	Größe [Bytes]
Programmcode	3152
RAM	53

Tabelle 48 – PWR-Modul Speicherverbrauch

4.5. Aufsätze auf die Treiber

Die Treiber kommunizieren direkt mit den entsprechenden Hardware-Modulen: Mit den gegebenen Möglichkeiten, aber auch mit deren Einschränkungen. Um nun etwas Abstraktion oder auch nur einen größeren Funktionsumfang zu bekommen, werden weitere Zwischenschichten eingeführt.

Die Module GPS und MMC besitzen so einen Aufsatz.

4.5.1. GPS Aufsatz: GPSFormat

4.5.1.1. Aufgabe

Der GPS-Treiber liefert die direkten Ausgabedaten aus den NMEA-Datensätzen. Allerdings sind große Teile dieser Daten für die weitere Verarbeitung denkbar ungeeignet. Das Problem liegt daran, daß viele logisch zusammengehörende Daten in physikalisch getrennten Variablen gespeichert werden.

So zu Beispiel die Breiten- und Längengrade: Jeder von ihnen besteht aus vier einzelnen Variablen: `_Degree`, `_Minute`, `_Minute1000th` (3 Nachkommastellen der Minutenzahl) und noch dazu aus `Direction`. Um nun zwei verschiedene Werte miteinander vergleichen und auch einen genauen Abstand zwischen den Werten berechnen zu können, werden diese in einen einzigen großen Wert – der den kompletten Wertebereich abdeckt – zusammengefaßt. Dieser `long`-Wert `_Minutes1000th` gibt den absoluten Breiten- oder Längengrad in $^{1}/_{1000}$-Minuten an.

Ebenso verhält es sich mit der Zeitangabe. Die aktuelle Zeit besteht sogar aus sechs einzelnen Daten: Jahr, Monat, Tag, Stunde, Minute und Sekunde. Sie wird in eine einzelne `long`-Zahl umgerechnet, die die Anzahl der vergangenen Sekunden seit dem 01.01.2000 angibt. Diese Berechnung wird uns noch mit einigen Problemen der Datumsberechnung konfrontieren, wie später ausgiebig zu sehen ist.

Letztendlich sollen noch Werte aus – für uns – unbrauchbaren Einheiten in verwendbare konvertiert werden. Beispielsweise gibt der GPS-Empfänger die Geschwindigkeit in Knoten aus. Diese wird zusätzlich in $^{m}/_{s}$ und in $^{km}/_{h}$ umgerechnet.

4.5.1.2. INTERFACE

```
//-- Convert -----------------------------------------------------------------------------
    void GPSFormat_Convert( void );
//----------------------------------------------------------------------------------------

//-- Helper ------------------------------------------------------------------------------
    void GPSFormat_GetTextDirection( char *strCourse, WORD Course_Degree );
//----------------------------------------------------------------------------------------
```

Listing 44 – GPSFormat, Interface

Wie auch schon im GPS-Treiber sind hier die globalen Daten interessanter:

```
//----------------------------------------------------------------------------------------
// GPS format data typedef
//----------------------------------------------------------------------------------------
typedef struct _tGPS_DATA_EX
{
    WORD    wDays_SinceYear2000;
    DWORD   dwSeconds_SinceMidnight;
    DWORD   dwSeconds_SinceYear2000;
    tCompressedDateTime     CompressedDateTime;

    long    lLatitude_Minutes1000th;
    long    lLongitude_Minutes1000th;

    WORD    wSpeed_MeterPerSecond10th;
    WORD    wSpeed_kmh;
} tGPS_DATA_EX;
//----------------------------------------------------------------------------------------


//----------------------------------------------------------------------------------------
// exported globals
//----------------------------------------------------------------------------------------
extern volatile tGPS_DATA_EX    GPS_DATA_EX;            // converted GPS state
//----------------------------------------------------------------------------------------
```

Listing 45 – GPSFormat, Interface Daten

4.5.1.3. AUFBAU DER ZWISCHENSCHICHT

Die eigentliche Frage hier ist wohl, warum diese Berechnungen nicht direkt im NMEA-Parser ausgeführt werden. Die nötigen Daten wären dort schon alle vorhanden und müßten nur noch umgerechnet werden. Die Antwort ist ebenso einfach wie einleuchtend: Performance. Da Teile der Berechnungen – vor allem die der Datumsfunktionen – zeitaufwendig sind, kann nicht mehr gewährleistet werden, daß die Berechnungen vor dem Eintreffen des nächsten Zeichens abgeschlossen sind. Somit wäre das nächste Zeichen verloren und damit der ganze Datensatz unbrauchbar. Außerdem ist mit diesem Aufbau die Trennung zwischen den wirklich empfangen Daten des Treibers und der von uns berechneten, eindeutiger.

Zu den einzelnen Funktionen:

- **`void GPSFormat_Convert( void );`**
  Die Umrechnung wird nur innerhalb dieser Funktion angestoßen. Nach einem positiven Aufruf von `GPS_NewDataAvailable(…)` sollte die Applikation gleich die Funktion `GPSFormat_Convert(…)` ausführen:

```
while ( TRUE )
{
    if ( GPS_NewDataAvailable() )
    {
        GPSFormat_Convert();

        // a new data record received
        // now we can use GPS_DATA and GPS_DATA_EX

        [...]
    }
    [...]
}
```

Listing 46 – GPSFormat, Verwendung

- **`void GPSFormat_GetTextDirection( char *strCourse, WORD Course_Degree );`**

  Diese Funktion ist nur eine Hilfsfunktion um einen bestimmten Kurs einer Himmelsrichtung nach folgender Tabelle zuzuordnen. Der Wertebereich des Kurses ist beliebig, er muß allerdings positiv sein.

Idx	Von [°]	Bis [°]	Richtung
0	348.75	11.25	N
1	11.25	33.75	NNE
2	33.75	56.25	NE
3	56.25	78.75	ENE
4	78.75	101.25	E
5	101.25	123.75	ESE
6	123.75	146.25	SE
7	146.25	168.75	SSE
8	168.75	191.25	S
9	191.25	213.75	SSW
10	213.75	236.25	SW
11	236.25	258.75	WSW
12	258.75	281.25	W
13	281.25	303.75	WNW
14	303.75	326.25	NW
15	326.25	348.75	NNW

Tabelle 49 – GPSFormat, Zuordnung der Himmelsrichtungen

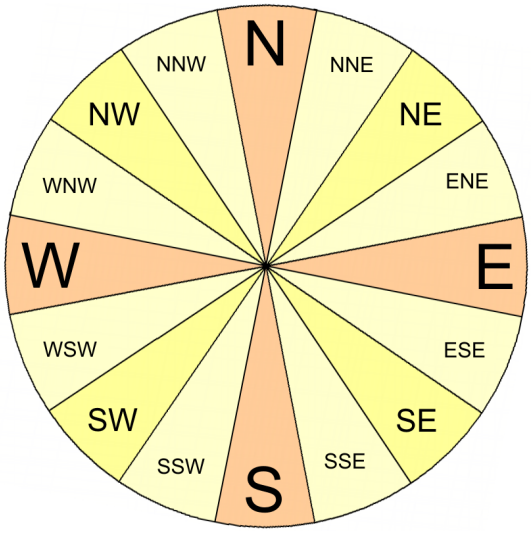


Abbildung 140 – GPSFormat, Himmelsrichtungen

Der nächste Abschnitt beschreibt den eingesetzten Algorithmus:

4.5.1.4. Implementierung kritischer Stellen

GPSFormat_GetTextDirection

Zuerst zu dem eben angesprochenen Algorithmus:

```
/*********************************************************************************************
| Converts Course_Degree into text
|--------------------------------------------------
|--------------------------------------------------------------------------------------------
| @Params:
|   strCourse           destination for direction text (size must be >= 4)
|   Course_Degree       Course to convert
| @Return Value:        - none -
+*********************************************************************************************/
public void GPSFormat_GetTextDirection( char *strCourse, WORD Course_Degree )
{
    // calculate index:
    // idx = ( Course + 11.25 ) / 22.5
    // idx = ( Course * 4 + 45 ) / 90
    WORD    idx = ( ( ( Course_Degree % 360 ) << 2 ) + 45 ) / 90;

    // load text
    PGM_P   pgmp;
    memcpy_P( &pgmp, &GPSFormat_TextDirection[ idx % 16 ], sizeof(PGM_P) );

    // finally copy text
    strncpy_P( strCourse, pgmp, 4 );
}
```

Listing 47 – GPSFormat, GPSFormat_GetTextDirection

Der Index für den entsprechenden Eintrag wird über folgende Formel berechnet:

$$idx = \frac{Course + 11.25}{22.5}$$

Um sich die Arbeit mit Kommazahlen zu ersparen werden der Divisor und der Dividend mit vier erweitert:

$$idx = \frac{Course \cdot 4 + 45}{90}$$

Somit kann die Funktion die Berechnung mit nur Integer-Zahlen ausführen.

BREITEN- / LÄNGENGRADBERECHNUNG

Die Formel zur Berechnung der $^{1}/_{1000}$ Minuten lautet:

$$lLat = \underset{bNegate}{\pm}\left(\left(byDegree \cdot 60\right) + byMinute \cdot 1000\right) + wMinute1000th$$

Somit sind alle Ost-Werte positiv und alle West-Werte negativ, die Nord-Werte positiv und die südlichen wieder negativ. Umgesetzt in die entsprechende Funktion:

```
_private long GPSFormat_ConvertLatLon( BYTE byDegree, BYTE byMinute, WORD wMinute1000th, BOOL bNegate )
{
    // get lat/lon minutes and add degrees*60 to it
    long    lLat = ((WORD)( (WORD)byDegree * (WORD)60 )) + ((WORD)byMinute);

    // get lat/lon 1/1000 th minutes, add them and return result
    lLat = ( lLat * 1000 ) + wMinute1000th;

    // negate result?
    if ( bNegate )
        lLat = -lLat;

    // return result
    return lLat;
}
```

Listing 48 – GPSFormat, GPSFormat_ConvertLatLon

GESCHWINDIGKEITSUMRECHNUNG

Ein Knoten entspricht 0.5144444 $^{m}/_{s}$. Zur Umrechnung muß aber eine performante Möglichkeit ohne aufwendige Gleitkommaberechnung gefunden werden:

Zusätzlich müssen noch die Einheiten beachtet werden: In der Variablen `wSpeed_Knots10th` werden $^{1}/_{10}$ Knoten gespeichert, in der Zielvariablen `wSpeed_MeterPerSecond10th` $^{1}/_{10}$ $^{m}/_{s}$. Hier heben sich die beiden Faktoren gegenseitig auf.

Den eben ermittelten Faktor 0.5144 erreicht man annähernd mit dem Bruch:

$$0.5144 \approx \frac{33715}{65536}$$

Der untere Divisor ist nicht umsonst so gewählt, denn 65536 = 2^{16}, also genau die Größe eines `WORD`s. Damit kann man das Ergebnis einfach um 16 Bits nach rechts verschieben und spart sich die zeitraubende Division. Intern optimiert der C-Compiler die Verschiebung sogar soweit, daß der Prozessor gar nichts mehr verschiebt sondern direkt auf die oberen zwei Bytes – dem `HIGHWORD` – zugreift.

Um die Knoten auch passend in Stundenkilometer umzurechnen, benötigen wir einen Faktor von 1.852. Allerdings werden die Knoten – wie oben schon beschrieben – in 1/10 Knoten angegeben, die Stundenkilometer aber in vollen Einheiten. Deshalb muß der Faktor noch mit 0.1 multipliziert werden:

$$0.1852 \approx \frac{121373}{65536}$$

Der Code dafür sieht dann so aus:

```
_private void GPSFormat_ConvertSpeed( void )
{
    WORD wSpeed_Knots10th = GPS_DATA.wSpeed_Knots10th;

    // convert knots --> m/s  (1 knots = 0.5144444 m/s) (approxed: * 33715 / 65536 )
    GPS_DATA_EX.wSpeed_MeterPerSecond10th = (WORD)( (DWORD)( (DWORD)wSpeed_Knots10th * (DWORD)33715 ) >> 16
                          );

    // convert knots --> km/h  (1 knots = 1.852 km/h) (approxed: 12137 * / 65536 )
    GPS_DATA_EX.wSpeed_kmh = (WORD)( (DWORD)( (DWORD)wSpeed_Knots10th * (DWORD)12137 ) >> 16 );
}
```

Listing 49 – GPSFormat, GPSFormat_ConvertSpeed

DATUMS- / ZEITBERECHNUNG

Jetzt wird es etwas aufwendiger, deshalb zuerst ein wenig Kalenderkunde: Um ein Datum mit Zeit – also das Format, das wir vom GPS-Empfänger bekommen – in einen linear monoton steigenden Wert umzurechnen, wird als Grundeinheit die Sekunde genommen. Zusätzlich benötigen wir noch einen Bezugspunkt für die Sekunde Null. Dafür legen wir den 01.01.2000 fest. Der errechnete Wert gibt also die Sekunden seit dem Startdatum 0 an.

Als Speichergröße verwenden wir einen `DWORD`-Wert. Ein Tag hat 24 * 60 * 60 Sekunden, also genau 86'400 Sekunden. Wenn man davon ausgeht, daß ein Jahr im Durchschnitt ungefähr 365.25 Tage lang ist, kommen wir auf 31'557'600 Sekunden pro Jahr. Mit einem `DWORD` sind deshalb mindestens 2^{32} / `SekundenProJahr` = 136 Jahre abbildbar. Bei unserem Bezugspunkt geht unser Wertebereich bis etwa 2136.

Wie kann man aber das Datum in Sekunden umrechnen? Den reinen Zeitteil mit Stunden, Minuten und Sekunden kann man sehr einfach berechnen und in einer Zwischenvariablen speichern:

$$SekundenSeitMitternacht = (Stunden \cdot 60 + Minuten) \cdot 60 + Sekunden$$

Die andere Zwischenvariable beinhaltet die Tage seit 01.01.2000. Dafür muß man aber wissen, wie unser Kalender aufgebaut ist:

Unseren jetzigen Kalender nennt man den *Gregorianischen Kalender*. Der Vorgänger davor war der *Julianische Kalender*: Ein Kalender ordnet einem bestimmten Datum einen Wochentag zu.

Bezugspunkt unserer Zeitrechnung ist der 01.01.1 nach Christus. Neben den verschieden langen Monaten, gibt es auch noch eine Sonderregel: Das Schaltjahr. Alle durch vier teilbaren Jahre sind Schaltjahre und verlängern damit den Februar auf 29 Tage, also zusammen 366 Tage im Jahr.

Der jetzige *Gregorianische Kalender* geht noch einen Schritt weiter und definiert die Ausnahme der Ausnahme: Das Schaltjahr entfällt, wenn die Jahreszahl durch 100 teilbar ist, nicht aber durch 400. So kommt es, daß 1700, 1800 und 1900 jeweils kein Schaltjahr war, wohl aber das Jahr 2000.

Unsere Zeitrechnung beginnt im Jahr 2000; dieses war ein Schaltjahr. Wenn wir nun allerdings den alten *Julianischen Kalender* annehmen, treten bei allen durch vier teilbaren Jahren ein Schaltjahr auf. Diese Rechnung paßt exakt zu dem echten Kalender, zumindest bis zum Jahr 2100. Bei der kurzlebigen Elektronik wird das aber in Kauf genommen. Dadurch lassen sich Optimierungen nicht einsetzen.

Das Jahr wird nicht mit Tausenderstellen angegeben sondern nur die letzten beiden Ziffern, zum Beispiel statt 2005 nur 05. Die Tage seit Neujahr 2000 berechnen sich dann wie folgt:

$$TageSeit2000 = Jahr \cdot 365 + AnzahlDerSchaltjahre(Jahr) +$$
$$+ TageDieses JahrBisMonat(Monat) + Tag$$

Die optimierte Form `AnzahlDerSchaltjahre` – gültig ab 2001, also 1 – zu berechnen, kann man am schnellsten so:

$$AnzahlDerSchaltjahre(Jahr) = floor\left(\frac{Jahr-1}{4}\right)+1$$

Um `TageDiesesJahrBisMonat` auszurechnen, zählt man die Tage pro Monat ab Januar bis hin zum gewünschten Monat. Die Funktion `GPSFormat_DaysInMonth(…)` liefert die Länge des Monats in Tagen zurück:

```
_private BYTE GPSFormat_DaysInMonth( BYTE byMonth, BYTE byYear )
{
    BYTE byDays;
    if ( byMonth == 2 )
        // feb.: see if it is a lap year an return 28 or 29 days
        byDays = ( byYear&3 ) ? 28:29;
    else
        // all other months: return 30 or 31 days
        byDays = ( ( byMonth>>3 ) ^ ( byMonth&1 ) ) + 30;
    return byDays;
}
```

Listing 50 – GPSFormat, GPSFormat_DaysInMonth

Wenn der Februar gewünscht wird, muß zusätzlich noch das Jahr auf ein Schaltjahr hin überprüft werden und dann direkt 28 beziehungsweise 29 zurückgegeben werden.

Ansonsten wird die Formel

$$byDays = \left(byMonth_{Bit3} \oplus byMonth_{Bit0}\right) + 30$$

angewendet. $\oplus$ ist das Zeichen für ein exklusives Oder, kurz `XOR`. Daraus ergibt sich die Programmzeile:

```
        byDays = ( ( byMonth>>3 ) ^ ( byMonth&1 ) ) + 30;
```

Listing 51 – GPSFormat, GPSFormat_DaysInMonth, Ausschnitt

Um das besser zu verstehen, sehen wir uns eine Tabelle mit den Monaten und deren Länge an: In der ersten Spalte steht die Monatszahl, danach die darin vorkommenden Tage. Anschließend ist die Monatszahl binär dargestellt und zuletzt eine Zahl, die angibt welche Zahl zu 30 addiert werden muß, um auf die Monatslänge zu kommen:

Monat	#Tage	Binär	30+x
1	31	0001	1
2	--	--	--
3	31	0011	1
4	30	0100	
5	31	0101	1
6	30	0110	
7	31	0111	1
8	31	1000	1
9	30	1001	
10	31	1010	1
11	30	1011	
12	31	1100	1

Tabelle 50 – GPSFormat, Monatslänge

Das Ergebnis, ob inkrementiert werden soll, erhält man durch logische Verknüpfung und ist `bit3 XOR bit0`.

Die ganze Funktion zur Berechnung der Sekunden sieht nun wie folgt aus. Die einzelnen, oben besprochenen, Schritte sieht man darin recht gut. Zuerst werden die Tage seit Anfang 2000 berechnet, danach die Sekunden seit Mitternacht und zuletzt die gesamte Summe:

```
_private void GPSFormat_ConvertDateTime( void )
{
    […]

    //.....................................................................
    // wDays_SinceYear2000
    //.....................................................................

    // get year since 2000
    BYTE    byYear = GPS_DATA.byDate_Year;

    // one year has 365 days
    WORD    wDays = ( (WORD)byYear * (WORD)365 );

    // add lap years (use only simple formula without 100/400 year rule, it will
    // be valid until 2099)
    wDays += ( (byYear-1) >> 2 ) + 1;

    // add month-days in current year
    for ( BYTE m = 1; m < GPS_DATA.byDate_Month; m++ )
        wDays += GPSFormat_DaysInMonth( m, byYear );

    // add days in current month
    wDays += GPS_DATA.byDate_Day;

    // store
    GPS_DATA_EX.wDays_SinceYear2000 = wDays;


    //.....................................................................
    // dwSeconds_SinceMidnight
    //.....................................................................
    WORD    wSeconds = (WORD)((WORD)GPS_DATA.byTime_Minute * (WORD)60 ) + (WORD)GPS_DATA.byTime_Second;
    DWORD   dwSeconds = ( (DWORD)GPS_DATA.byTime_Hour * (DWORD)3600 ) + wSeconds;
    GPS_DATA_EX.dwSeconds_SinceMidnight = dwSeconds;


    //.....................................................................
    // dwSeconds_SinceYear2000
    //.....................................................................
    //  one day has 24 * 3600 = 86400 seconds, so this would be:
    //  --> Seconds_SinceYear2000 = wDays * 86400 + dwSeconds

    GPS_DATA_EX.dwSeconds_SinceYear2000 = ( (DWORD)wDays * (DWORD)86400 ) + dwSeconds;
}
```

Listing 52 – GPSFormat, GPSFormat_ConvertDateTime

4.5.1.5. BEISPIELVERWENDUNG UND TESTS

Wie schon oben beschrieben, läßt sich die Funktion ganz einfach in die bestehende GPS-Schleife einbauen:

```
// init
[...]

while ( TRUE )
{
    if ( GPS_NewDataAvailable() )
    {
        GPSFormat_Convert();

        // a new data record received
        // now we can use GPS_DATA and GPS_DATA_EX

        [...]
    }
    [...]
}
```

Listing 53 – GPSFormat, Verwendung

4.5.1.6. BERECHNUNGEN, ANALYSEN UND STATISTIKEN

Hier eine Aufstellung des ungefähr benötigten Speichers für den GPSFormat-Zusatz, inklusive dem originalen GPS-Treiber:

Speicherart	Größe [Bytes]
Programmcode	3382
RAM	91

Tabelle 51 – GPSFormat Speicherverbrauch

4.5.2. MMC AUFSATZ: STORAGE

4.5.2.1. AUFGABE

Der darunterliegende MMC-Treiber ermöglicht einen wahlfreien Sektorzugriff auf die Speicherkarte. Damit müssen jeweils 512 Bytes auf einmal gelesen, bearbeitet und wieder geschrieben werden.

Um die Routen zu verwalten, muß ein Verwaltungssystem in Aktion treten, ähnlich einem Dateisystem. Die Anforderungen hierfür sind, daß die Daten zu jeder Zeit konsistent sind und auch durch plötzliches Entfernen der MMC-Karte kein Schaden oder Verlust vorheriger Daten auftritt.

Der Zugriff soll nun nicht mehr direkt über Sektoren geschehen, sondern über Routen. Es muß zwei Hauptfunktionen geben, die eine zum Anlegen einer neuen Route und die andere, um dieser Route neue Daten bytegenau hinzuzufügen.

Ferner muß die Möglichkeit bestehen eine neue, teilweise benutzte oder volle Disk zu formatieren, also mit unserem Dateisystem zu beschreiben.

Zuletzt sollen noch Funktionen zum Abfragen der bereits vorhandenen Routen implementiert werden.

4.5.2.2. INTERFACE

```
//-- Init and Access Functions ------------------------------------------------------------
    STG_RET Storage_OpenDisk( void );

    STG_RET Storage_NewRoute( tCompressedDateTime DateTime );
    STG_RET Storage_AddData( void *pData, WORD wLen );
    STG_RET Storage_Flush( void );

    STG_RET Storage_FormatDisk( void );
//-----------------------------------------------------------------------------------------

//-- Info Functions -----------------------------------------------------------------------
    STG_RET Storage_GetStoredRoutesCount( WORD *pwCount );
    STG_RET Storage_GetStoredRouteInfo( WORD wRoute, tStorage_RouteInfo *pRouteInfo );
//-----------------------------------------------------------------------------------------

//-- error message handling ---------------------------------------------------------------
    PGM_P   Storage_GetErrorText( STG_RET byRet );
//-----------------------------------------------------------------------------------------
```

Listing 54 – Storage, Interface

4.5.2.3. AUFBAU DER ZWISCHENSCHICHT

Als erstes soll hier der Aufbau des Storage-Dateisystems vorgestellt werden:

ROOTSEKTOR

Der einzig fest definierte Sektor, ist Sektor 0 und heißt Rootsektor. Er ist wie folgt aufgebaut:

```
typedef struct _tStorage_RootSector {
    BYTE                abyMagicID[4];
    tCompressedDateTime dtVersion;
    DWORD               dwSectorPos_NextFreeSector1;
    DWORD               dwSectorPos_Routes4;
} tStorage_RootSector;
```

Listing 55 – Storage, Rootsektor

`abyMagicID` ist eine ID, damit die Karte mit dem Storage-Dateisystem als gültige Speicherkarte akzeptiert wird. `dtVersion` beinhaltet die Version der Karte, gespeichert als DateTime-Typ. Die nächsten beiden Variablen sind sehr wichtig und geben jeweils die Sektornummer der beiden weiteren Systemsektoren an:

DWSECTORPOS_NEXTFREESECTOR1

In diesem Sektor werden nur die ersten vier Bytes verwendet. An Offset 0 befindet sich die Variable die angibt, welches der nächste freie Sektor der Speicherkarte ist. Wenn eine neue Route angelegt wird, oder einfach nur in einen neuen Sektor geschrieben werden soll, so ist dieser Sektor – beziehungsweise der Wert daraus – zu verwenden.

DWSECTORPOS_ROUTES4

Diese Variable zeigt auf den ersten von vier Sektoren, die direkt auf diesen Sektor folgen. Darin befinden sich die einzelnen Verweise der aufgezeichneten Routen. Jede Route hat folgende Information:

```
typedef struct _tStorage_RouteInfo {
    tCompressedDateTime     DateTime;
    DWORD                   dwStartSector;
} tStorage_RouteInfo;
```

Listing 56 – Storage, RouteInfo

Sie besteht also aus dem Startzeitpunkt und dem Anfangssektor dieser Route. Jede Route belegt somit acht Bytes. In den vier Sektoren stehen insgesamt 2kB zur Verfügung. Somit können maximal 256 Routen abgespeichert werden.

Zuerst die Zugriffsfunktionen:

- **`STG_RET Storage_OpenDisk( void );`**
  Zu Beginn ruft die Funktion `MMC_InitCard(…)` auf, um die MMC-Karte zu initialisieren. Anschließend muß der Rootsektor ausgelesen und überprüft werden.

- **`STG_RET Storage_Read_NextFreeSector( DWORD *pdwNextFreeSector );`**
- **`STG_RET Storage_Write_NextFreeSector( DWORD dwNextFreeSector );`**
  Diese beiden Funktionen liefern den Wert im NextFreeSektor-Sektor zurück oder schreiben ihn. Da das Dateisystem immer konsistent gehalten werden muß, sind keine globalen Variablen dafür vorhanden und jeder Zugriff geschieht direkt in diesem Sektor.

- **`STG_RET Storage_NewRoute( tCompressedDateTime DateTime );`**
  Diese Funktion ist ziemlich umfangreich: Zuerst ermittelt sie den nächsten freien Sektor, initialisiert diesen mit lauter Nullen, erhöht die eben ausgelesene Variable und schreibt sie zurück.

  Danach wird gezählt wie viele Routen bereits auf der Karte sind. Diese Zahl ist zugleich der Index der neu zu erstellenden Route. Anhand dieses Indexes muß Sektor und Offset in einem der vier Routeneintragssektoren für den neuen Routeneintrag berechnet werden. Dann liest die Funktion diesen Sektor aus, bearbeitet die Routeninformation mit den neuen Routendaten und schreibt den Sektor anschließend wieder zurück.

  Zuletzt müssen noch ein paar Verwaltungsvariablen gesetzt werden.

- **`STG_RET Storage_AddData( void *pData, WORD wLen );`**
  Diese Funktion bekommt als Parameter einen Pointer auf eine Datenstruktur und deren Länge übergeben. Diese Daten werden nun byteweise in den Sektorpuffer (einer globalen Variablen) geschrieben. Falls dabei die Sektorgrenze erreicht wird, ruft die Funktion `Storage_Flush(…)` auf und aktualisiert den NextFreeSektor-Wert entsprechend. Danach wird alles für den nächsten Sektor vorbereitet und initialisiert.

- **`STG_RET Storage_Flush( void );`**
  Hier wird einfach nur der aktuelle Inhalt des Sektorpuffers auf die Karte geschrieben. Danach sind die Daten fest auf der Karte. Entweder muß der Aufrufer diese Funktion nach jedem Hinzufügen von Daten aufrufen oder nur am Schluß der Route. Dadurch gehen, bei einem eventuellen Ausschalten oder Entfernen der Karte, allerdings alle Daten bis zur letzten Sektorgrenze verloren.

- **`STG_RET Storage_FormatDisk( void );`**
  Initialisiert eine funktionstüchtige Karte mit einem neuen Storage-Dateisystem. Falls bereits Daten auf der Karte vorhanden sind, zum Beispiel Dateien in einem FAT-Dateisystem, sind diese nach dem Aufruf unbrauchbar. Nach dem Formatieren kann der Benutzer die Karte im GPS-Logger einsetzen.

- **`STG_RET Storage_GetStoredRoutesCount( WORD *pwCount )`**
  Gibt die Anzahl der gespeicherten Routen auf der Karte zurück.

- **`STG_RET Storage_GetStoredRouteInfo( WORD wRoute, tStorage_RouteInfo *pRouteInfo )`**
  Diese Funktion liefert den entsprechenden Eintrag in der RouteInfo-Liste, aus einem der vier RouteInfo-Sektoren, zurück.

Zuletzt noch die konsequente Weiterführung der `MMC_GetErrorText(...)`-Funktion:

- **`PGM_P Storage_GetErrorText( STG_RET byRet );`**
  Liefert eine entsprechende Fehlerbeschreibung zurück. Falls der Storagetreiber keine passende Meldung besitzt, so ruft er die `MMC_GetErrorText(...)`-Funktion auf.

Der Treiber definiert drei neue Fehlertypen, zusätzlich zu den schon vorhanden MMC-Fehlern:

```
#define     STG_FAILED              STG_FBASE + 1       // storage error
#define     STG_FAILED_WRONGCARD    STG_FBASE + 2       // wrong card version
#define     STG_FAILED_CARDFULL     STG_FBASE + 3       // card full
```

Listing 57 – Storage Fehlerdefinitionen

4.5.2.4. Implementierung kritischer Stellen

Dieser Treiberaufsatz ist allgemein recht komplex, da die Daten zu jeder Zeit persistent sein müssen. Die detaillierte Beschreibung der einzelnen Funktionen würde den Rahmen um einiges sprengen. Bei Interesse sei hier auf den gut dokumentierten Quelltext verwiesen. Die Kenntnis der genauen Funktionsweise ist für das Verständnis der weiteren Funktionen nicht relevant. Man sollte nur wissen, daß sich die einzelnen Storages wie Dateien verhalten. Sie können neu angelegt werden und es kann sequentiell byteweise in sie geschrieben werden.

4.5.2.5. Beispielverwendung und Tests

Eine einfache Testanwendung – ohne Fehlerüberprüfung – könnte so aussehen:

```
// open disk
Storage_OpenDisk( void );

// create new route
Storage_NewRoute( dtNow );

// write date
while ( TRUE )
{
    // get data
    dwLen = GetSomeData( &pBuffer );

    // add date
    Storage_AddData( pBuffer, dwLen );
    Storage_Flush();
}
```

Listing 58 – Storage Testprogramm

4.5.2.6. Berechnungen, Analysen und Statistiken

Hier eine Aufstellung des ungefähr benötigten Speichers für den Storage-Aufsatz, inklusive dem notwendigen MMC-Treiber:

Speicherart	Größe [Bytes]
Programmcode	4042
RAM	557

Tabelle 52 – GPSFormat Speicherverbrauch

4.6. ROUTENAUFZEICHNUNG UND KOMPRESSION – RECORD

4.6.1. AUFGABE

Das Record-Modul baut – im Gegensatz zu den bisherigen Aufsätzen – nicht nur auf einem Treiber auf sondern gleich auf mehreren: Zuerst einmal natürlich auf Storage zum Abspeichern aber auch auf GPS und GPSFormat als Datenquelle.

Die Hauptaufgabe ist das Anlegen und Verwalten von Routen, also das eigentliche **Aufzeichnen**. Als Dateneingang dienen die GPS-Werte, die Record erst analysiert, dann komprimiert und anschließend mit Hilfe des Storage-Treibers abspeichert.

Es gilt eine möglichst gute und einfache Kompression für die Aufzeichnung zu finden, um mit dem vorhandenem Speicher möglichst sparsam umzugehen und damit die Aufzeichnungsdauer zu maximieren.

4.6.2. INTERFACE

```
//-- Init and Access Functions ------------------------------------------------------
    REC_RET Record_New( void );
    REC_RET Record_Sync( void );
    REC_RET Record_End( void );
//-----------------------------------------------------------------------------------

//-- error message handling ---------------------------------------------------------
    PGM_P   Record_GetErrorText( REC_RET byRet );
//-----------------------------------------------------------------------------------
```

Listing 59 – Record Interface

4.6.3. AUFBAU DER ZWISCHENSCHICHT

Zuerst die eigentlichen Record-Funktionen:

- **`REC_RET Record_New( void );`**
  Zum Erstellen einer neuen Route. Mit Hilfe des Storage-Treibers wird eine neue Route erstellt und gleich der erste Eintrag eingefügt.

- **`REC_RET Record_Sync( void );`**
  Jede Sekunde, also sobald `GPS_NewDataAvailable(…) TRUE` wird, ist diese Funktion aufzurufen. Sie speichert den aktuellen GPS-Zustand auf der Karte ab.

- **`REC_RET Record_End( void );`**
  Soll nun die Aufzeichnung einer Route beendet werden, so muß die Applikation diese Funktion aufrufen.

Und zur Fehlerbehandlung kann auch wieder die `GetErrorText`-Funktion verwendet werden:

- **`PGM_P   Record_GetErrorText( REC_RET byRet );`**
  Sie liefert, wie schon die Funktionen aus MMC und Storage, einen Pointer auf eine knappe Fehlerbeschreibung zurück.

Das Interface selbst gestaltet sich offensichtlich nicht sonderlich anspruchsvoll, allerdings steckt ein wenig mehr dahinter als zunächst vermutet, was detailliert in den nächsten Abschnitten erklärt wird.

4.6.4. KOMPRESSION

Um dem Speicherplatz möglichst effektiv einzusetzen, muß ein möglichst guter Kompressionsalgorithmus gefunden werden. Sehen wir uns also erst einmal an, welche Daten wir überhaupt vom GPS- (`GPS_DATA.*`) bzw. GPSFormat-Treiber (`GPS_DATA_EX.*`) bekommen, bzw. welche davon aufgezeichnet werden sollen:

Funktion	Variable	Beschreibung
Datum/Zeit	`dwSeconds_SinceYear2000`	Sekunden seit 01.01.2000 um Mitternacht
	`CompressedDateTime`	Bitkomprimierte Datums- und Zeitvariable
Position	`lLatitude_Minutes1000th`	Breitengrad
	`lLongitude_Minutes1000th`	Längengrad
	`sAltitude_Meter`	Höhe
Bewegung	`wSpeed_kmh`	Geschwindigkeit
Qualität	`byQuality`	Qualität
	`byNumberOfSatellites`	Anzahl der empfangenen Satelliten

Tabelle 53 – Kompression, Variablen

Die beiden Variablen `dwSeconds_SinceYear2000` und `CompressedDateTime` beinhalten die gleiche logische Information und sind deshalb redundant. `CompressedDateTime` wird nur ein einziges Mal – im ersten Datensatz – mitgespeichert, damit sich der Client die Umrechnung sparen und direkt die mitgelieferten Daten verwenden kann.

4.6.4.1. UNKOMPRIMIERTE SPEICHERUNG

Die einfachste Methode wäre nun jede Sekunde die Daten komplett abzuspeichern:

Variable	Variablengröße [Byte]
`dwSeconds_SinceYear2000`	4
`lLatitude_Minutes1000th`	4
`lLongitude_Minutes1000th`	4
`sAltitude_Meter`	2
`wSpeed_kmh`	2
`byQuality`	1
`byNumberOfSatellites`	1

Tabelle 54 – Kompression, unkomprimierte Variablengröße

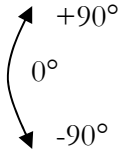
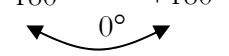
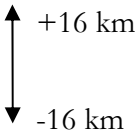
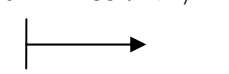
Zusammen also 18 Bytes pro Sekunde. Somit braucht eine Stunde unkomprimierte Aufzeichnung:

60 * 60 * 18 Bytes = 64800 Bytes

4.6.4.2. Effektive Datenfeldgrößen

Die Variablengrößen sind vom Computer, bzw. vom Compiler, vorgegeben: `BYTE`, `WORD`, `DWORD`, …

In Wirklichkeit haben unsere Werte jedoch – bis auf `dwSeconds_SinceYear2000`, das beliebig groß werden kann – einen anders fest definierten Wertebereich:

Variable Visualisierung	Wertebereich Ein Bit entspricht			Nötige Bits
lLatitude_Minutes1000th				
Breitengrad:  	**Min:** -90 * 60 * 1000 **Max:** 90 * 60 * 1000 **Ein Bit entspricht:** ~2 m	= -5.4 M = 5.4 M	> 10.8 M	**24, signed**
lLongitude_Minutes1000th				
Längengrad:  	**Min:** -180 * 60 * 1000 **Max:** 180 * 60 * 1000 **Ein Bit entspricht:** ~2 m	= -10.8 M = 10.8 M	> 21.6 M	**25, signed**
sAltitude_Meter				
Höhe:  	**Min:** -16000 **Max:** +16000 **Ein Bit entspricht:** 1 m		> 32000	**15, signed**
wSpeed_kmh				
Geschwindigkeit:  	**Min:** 0 **Max:** +1854 km/h **Ein Bit entspricht:** 1 km/h		> 1854	**11, unsigned**

BYQUALITY

Qualität:

Min:	0	= *invalid*	**1, unsigned**
Max:	1	= *valid*	

BYNUMBEROFSATELLITES

Satellitenanzahl:

Min:	0	**4, unsigned**
Max:	12	

Tabelle 55 – Kompression, effektive Datenfeldgrößen

Somit reduziert sich die Größe eines Datensatzes auf genau 80 Bit – also 10 Bytes statt den ehemaligen 18 Bytes – und damit auch der Verbrauch pro Stunde von 64 kB/h auf 36 kB/h.

4.6.4.3. KOMPRIMIERUNG AUF BITEBENE – BITSTREAM

Der erste Weg der Kompression ist es also, daß nicht mehr einzelne ganze Bytes unabhängig voneinander abgespeichert werden sondern jede Sekunde, also für jeden Datensatz ein logisch zusammenhängender Datenblock. Die ganze Kompression baut auf diesem Konzept auf. Deshalb ist das Verständnis der Methode sehr wichtig.

In einem Datenblock werden alle Daten zusammengefaßt und nur so viele Bits wie nötig belegt.

Die BitStream-Funktionen sind im Record-Quellcode mit eingebunden und bieten folgendes Interface:

```
// create a new BitStream
void BitStream_New( void );

// add n bits to the BitStream
void BitStream_Add( DWORD dwVal, BYTE byBits );

// receive the ready BitStream
void BitStream_Get( void **pBufferStart, BYTE *pbyLenBytes );


// add one 1 bits to the BitStream
void BitStream_Add1( void );

// add one 0 bits to the BitStream
void BitStream_Add0( void );

// add sx formatted bits to the BitStream (sign, and n bits starting with 0 => 1)
void BitStream_Add_sx( long lVal, BYTE byxBits );
```

Listing 60 – Komprimierung, BitStream Interface

Hier nur die Kurzbeschreibung:

- **`void BitStream_New( void );`**
  Löscht den alten Befehl und erzeugt einen neuen Bitstrom.

- **`void BitStream_Add( DWORD dwVal, BYTE byBits );`**
  Fügt dem Bitstrom `byBits` Bits aus der Variablen `dwVal` hinzu. Die Richtung ist LSB, das niederwertigste Bit kommt also zuerst (siehe Beispiel).

- **`void BitStream_Get( void **pBufferStart, BYTE *pbyLenBytes );`**
  Ist ein neuer Bitstrom fertig aufgebaut, so kann der Aufrufer das Ergebnis mit dieser Funktion ermitteln. Sie setzt den Puffer `pBufferStart` auf das erste Byte des Datenstroms, dessen Länge in `pbyLenBytes` geschrieben wird. Somit kann das Ergebnis ganz einfach dem Storage-Treiber übergeben werden und mit einem Aufruf komplett abgespeichert werden.

  Hier wird aber auch deutlich, daß nur ganze Bytes, also ein vielfaches von acht Bit, abgespeichert werden können. Wenn man einem Bitstrom zum Beispiel neun Bits hinzufügt, müssen trotzdem zwei Bytes abgespeichert werden. Die unbenutzten Bits füllt die Funktion mit Nullen auf.

Beispiel:
Ein neuer BitStream besteht aus lauter Nullen und beginnt beim LSB, also bei Byte 0 und Bit 0:

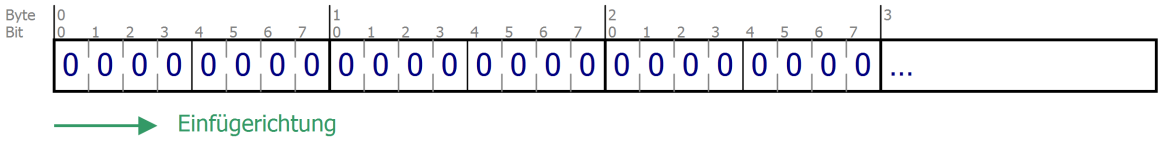


Abbildung 141 – BitStream, _Add(...) Funktionsweise 1

Sollen nun Daten hinzugefügt werden, so kommen diese von links nach rechts – auf Bitebene ausgerichtet – in den Datenstrom hinein.

Nehmen wir einmal die vier Aufrufe der `BitStream_Add(...)`-Funktion mit den gegebenen Parametern aus folgendem Listing an:

```
// create a new stream
BitStream_New();

// add data to stream
BitStream_Add( 1, 3 );          // #1 (val=1, 3 bits)
BitStream_Add( 1, 1 );          // #2 (val=1, 1 bit)
BitStream_Add( 2, 2 );          // #3 (val=2, 2 bits)
BitStream_Add( 0x01c4, 9 );     // #4 (val=0x01c4, 9 bits)

// get stream
BitStream_Get( &pBufferStart, &byLenBytes );
```

Listing 61 – BitStream, Beispielverwendung

BitStream_Add(…) wandelt die Werte der vier Aufrufe in Bitfelder um und hängt diese dem Bitstrom in der oben besprochenen Richtung an. Zum Verständnis des Bildes ist darauf zu achten, daß auch die Bits in der LSB-Schreibweise notiert sind. Aus einer 1 werden also bei 3-bit-Darstellung „`100`“:

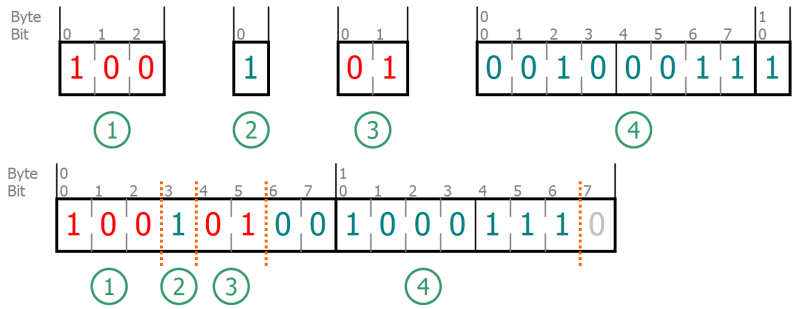


Abbildung 142 – BitStream, _Add(...) Funktionsweise 2

Das letzte Bit bleibt ungenutzt und somit 0. `byLenBytes` wird am Ende 2 beinhalten.

Das Ganze wird durch Shift-Operationen realisiert. Für den genauen Aufbau der Funktionen sei hier auf den Quellcode verwiesen.

Die restlichen drei Funktionen sind Erweiterungen von `BitStream_Add(…)`:

- **`void BitStream_Add1( void );`**
- **`void BitStream_Add0( void );`**
  Fügt dem Bitstrom genau ein Bit hinzu.

- **`void BitStream_Add_sx( long lVal, BYTE byxBits );`**
  Fügt dem Bitstrom einen vorzeichenbehafteten, symmetrischen Wert hinzu. Im Gegensatz zu normalen vorzeichenbehafteten Werten, wie `short` oder `long`, deren Wertebereich von [*–n, …, 0, …, +n-1*] reicht, ist unser Wertebereich [*–n, …, -1, +1, …, +n*] mit $n = 2^{byxBits}$.

  Wie der Name `_sx` schon vermuten läßt, wird zunächst das Vorzeichen (*sign*) als einzelnes Bit abgespeichert (0 → +, 1 → –) und anschließend `byxBits` viele Bits für den vorzeichenlosen Wert, also den Betrag von `|lVal|`, der allerdings nicht bei 0 startet sondern bei 1. Somit läßt sich der Wert 0 nicht darstellen, dieser wird aber auch nicht gebraucht, was später noch zu sehen ist. 0 bedeutet also 1, 1 bedeutet 2, usw…

  Beispiele:
  - `Add_sx( 1, 2 )` erzeugt „`000`" (`sxx` → positiv, |1| – 1 = 0)
  - `Add_sx( -1, 2 )` erzeugt „`100`" (`sxx` → negativ, |-1| – 1 = 0)
  - `Add_sx( 2, 2 )` erzeugt „`001`" (`sxx` → positiv, |2| – 1 = 1)
  - `Add_sx( -2, 2 )` erzeugt „`101`" (`sxx` → negativ, |-2| – 1 = 1)
  - `Add_sx( 3, 3 )` erzeugt „`0010`" (`sxxx` → positiv, |3| – 1 = 2)
  - `Add_sx( -4, 4 )` erzeugt „`10011`" (`sxxxx` → negativ, |-4| – 1 = 3)

  Der Code hierfür sieht wie folgt aus:

```
// add sx formatted bits to the BitStream (sign, and n bits starting with 0 => 1)
void BitStream_Add_sx( long lVal, BYTE byxBits )
{
    // 0 isnt allowed, so thread it as 1
    if ( lVal == 0 )
        lVal = 1;

    // get and add sign
    if ( lVal < 0 )
    {
        // negative
        lVal = -lVal;
        BitStream_Add1();
    }
    else
    {
        // positive
        BitStream_Add0();
    }

    // adjust value (0 --> 1)
    lVal--;

    // add value
    BitStream_Add( lVal, byxBits );
}
```

Listing 62 – BitStream, Add_sx(...)

Diese Funktion wird später für die Speicherung vorzeichenbehafteter Werte verwendet. Mit nur zwei Bits kann man also eine Veränderung von -2, -1, +1 und +2 darstellen. Somit kommen wir schon zum Thema des nächsten Abschnittes.

4.6.4.4. KOMPRESSIONSIDEE – RELATIVE DATEN

Die grundlegende Idee der Kompression ist es, nicht jede Sekunde die kompletten, absoluten Daten abzuspeichern sondern nur die Veränderung zum letzten Datensatz. Die Eingangsdaten werden also subtrahiert.

Die Formel dafür lautet:

```
DIFF = NOW – LAST
```

Die Differenz `DIFF` ist der aktuelle Wert `NOW` minus dem des letzen Datensatzes `LAST`, der letzten Sekunde.

Damit verringern sich die notwendigen Datenbreiten erheblich. Die Größe der Sekunden spielt sich im Gegensatz zu `dwSeconds_SinceYear2000` nicht mehr im Bereich von mehreren Hundert Millionen ab, sondern sollte – wenn alles in Ordnung ist – immer eins sein.

Hier soll ein Protokoll zum Einsatz kommen, welches variable Größenbereiche zuläßt – je nach Anforderung und Änderungsgröße. Dieses Protokoll wird im nächsten Abschnitt genauer beschrieben.

Es folgt eine Tabelle mit den Wertebereichen der relativen Daten mit verschiedenen Beispielswerten:

Variable **Bezeichnung**	**Abbildung** **Beispielwerte**	**Wertebereich**	**Bits, Typ**
Qualität	*Keinen relativen Daten*		1, unsigned
Δ-Satellitenanzahl	*1 Bit ≙ 1 Satellit*		
Absolutes Maximum:	12 Satelliten	→ ± 12	5, signed
Normal:	1 Satellit	→ ± 1	2, signed
Δ-Breiten-/Δ-Längengrad	*1 Bit ≙ 2m*		
Absolutes Maximum:	1854 $^{km}/_{h}$ ≈ 515 $^{m}/_{s}$	→ ± 256	10, signed
Normales Maximum:	460 $^{km}/_{h}$ ≈ 128 $^{m}/_{s}$	→ ± 64	7, signed
Schnell:	230 $^{km}/_{h}$ ≈ 64 $^{m}/_{s}$	→ ± 32	6, signed
Normal:	115 $^{km}/_{h}$ ≈ 32 $^{m}/_{s}$	→ ± 16	5, signed
Langsam:	15 $^{km}/_{h}$ ≈ 4 $^{m}/_{s}$	→ ± 2	2, signed
Δ-Höhe	*1 Bit ≙ 1m*		
Angenommenes Maximum:	64 $^{m}/_{s}$ (230 $^{km}/_{h}$)	→ ± 64	7, signed
Schnell:	4 $^{m}/_{s}$	→ ± 4	3, signed
Langsam:	1 $^{m}/_{s}$	→ ± 1	1, signed
Δ-Geschwindigkeit	*1 Bit ≙ 1 $^{km}/_{h}$*		
Angenommenes Maximum:	64 $^{km}/_{h}$ /$_{s}$	→ ± 64	7, signed
Schnell:	16 $^{km}/_{h}$ /$_{s}$	→ ± 16	5, signed
Langsam:	2 $^{km}/_{h}$ /$_{s}$	→ ± 2	2, signed

Tabelle 56 – Record, Δ-Wertebereiche und Beispiele

Daraus kann man sich nun eine Vorstellung machen, welche Daten bei normaler Fahrt anfallen können. Um das genauer belegen zu können und letztendlich einen effizienteren Algorithmus zu erstellen, soll nun eine möglichst repräsentative Fahrt analysiert werden.

Die nächste Tabelle stellt das statistische Vorkommen der Wertebereiche aus einer repräsentativen, normalen Fahrt dar. Diese wurden aus mehreren aufgezeichneten Routen gemittelt. Die Aufschlüsselung für Breiten- und Längengrad ist bewußt an das Tabellenende verschoben worden:

Variable	Wertebereich	Wahrscheinlichkeit
Δ-Satellitenanzahl	0	69 %
	± 1	96 %
	± 2	99 %
Δ-Höhe	0	82 %
	± 1	98 %
	± 2	99 %
Δ-Geschwindigkeit	0	35 %
	± 2	72 %
	± 4	92 %
	± 8	99 %
Δ-Breiten-/Δ-Längengrad	0	18 %
	± 2	36 %
	± 4	63 %
	± 8	74 %
	± 16	82 %

Tabelle 57 – Record, Δ-Wertebereiche, Wahrscheinlichkeiten

Wie man schön erkennen kann, kommt man in der Praxis mit sehr kleinen Deltawerten aus. Lediglich bei den Breiten- und Längengraden werden bei höheren Geschwindigkeiten noch sehr viele Bits gebraucht.

Auch hier ist die Lösung erstaunlich einfach: Der Algorithmus verwendet nicht die Veränderungswerte (Δ-Wert, Geschwindigkeit), sondern die Veränderung der Veränderung (ΔΔ-Wert, Beschleunigung), also die Krümmung. Bei genaueren Überlegungen kommt man auch auf die Ursache: Es treten zwar häufig hohe Geschwindigkeiten auf aber selten nur große Beschleunigungswerte; auf jeden Fall in einem wesentlich geringeren Wertebereich. Die Beschleunigungen lassen sich sehr einfach aus den vorhandenen Geschwindigkeitswerten berechnen.

Durch diese Verbesserung kommt man auf eine erstaunliche Wahrscheinlichkeitstabelle:

Variable	Wertebereich	Wahrscheinlichkeit
ΔΔ-Breiten-/ΔΔ-Längengrad	0	56 %
	± 2	97 %
	± 4	99 %

Tabelle 58 – Record, ΔΔ-Wertebereiche, Wahrscheinlichkeiten 2

Somit benötigt auch die Positionsbestimmung nur sehr wenige Bits im Bitstrom.

4.6.4.5. BEFEHLSSTROM – AUFZEICHNUNGSFORMAT

In diesem Kapitel soll ein geeignetes Format zur Aufzeichnung der Routen, mit Hilfe des in den letzten Abschnitten gewonnenen Wissens, gefunden werden.

Jede Sekunde erstellt der Algorithmus einen neuen BitStream, also einen neuen Befehlsblock.

STARTBLOCK

Wenn eine neue Route begonnen wird, müssen im ersten Block – dem Startblock – zuerst einmal die absoluten Werte gespeichert werden. Alle darauffolgenden Blöcke sind Deltablöcke, die also nur die Veränderungen zum letzten Block wiedergeben.

Den Startblock haben wir schon im Abschnitt „Effektive Datenfeldgrößen“ kennengelernt. Zur Vollständigkeit nochmals die Kurzzusammenfassung:

Startblock:

Variablenname	Bits
`CompressedDateTime`	32
`dwSeconds_SinceYear2000`	32
`lLatitude_Minutes1000th`	24
`lLongitude_Minutes1000th`	25
`sAltitude_Meter`	16
`wSpeed_kmh`	11
`byNumberOfSatellites`	4

Tabelle 59 – Record, Format, Startblock

Für `byQuality` wird 1 angenommen, da eine Route nur mit gültigem Signal gestartet werden kann und deshalb nicht extra abgespeichert wird.

Für den Startblock ergibt sich somit eine Gesamtgröße von 144 Bit, also genau 18 Bytes.

DELTABLOCK

Wie oben bereits angegeben, gibt dieser Block nur den Unterschied zum vorherigen Block wieder.

Anhand der Statistiktabellen aus dem letzten Abschnitt soll jetzt eine geeignete Datenrepräsentation gefunden werden. Die `sxx`-Werte geben – wie oben schon beschrieben – an, daß ein vorzeichenbehafteter, symmetrischer Wert gespeichert wird.

Der Deltablock besteht aus mehreren Teilen: Der erste ist der Deltablock-Start:

Deltablock-Start:

Bitformat	Bedeutung	Nächster Teil
`1`	Normaler Deltablock	Deltablock-Data
`01`	Erweiterter Deltablock mit Zeitversatz	Deltablock-Time
`001`	Ungültiger Block, `byQuality` = 0	Neuer Deltablock-Start
`000`	Routenende	-

Tabelle 60 – Record, Format, Deltablock-Start

Anschließend an diesen Block kommt in sehr seltenen Fällen der Timeblock, wenn der letzte Block nicht genau eine Sekunde zurückliegt. Deshalb wird auch im folgenden Block keine

Rücksicht auf die Größe gemacht. Der Block sollte nur nach längeren Verbindungsabbrüchen auftauchen, wie sie zum Beispiel bei Tunneldurchfahrten vorkommen können:

Deltablock-Time:

Variable	Bitformat	Bits	Bedeutung
wSeconds	xxxxxxxx xxxxxxxx	16, unsigned	Anzahl der Sekunden seit dem letzten Block

Tabelle 61 – Record, Format, Deltablock-Time

Normalerweise folgt nach dem Deltablock-Start gleich dieser Datenteil:

Deltablock-Data:

Variable	Werte-bereich	*Wahr-schein.*	Bitformat	Bit-aufbau	Bits
sddLatitude	0	*56 %*	0	1	1
(ΔΔ-Breitengrad)	± 2	*97 %*	10 sx	2 + 2	4
	± 32	*100 %*	11 sxxxxx	2 + 6	8
sddLongitude	0	*56 %*	0	1	1
(ΔΔ-Längengrad)	± 2	*97 %*	10 sx	2 + 2	4
	± 32	*100 %*	11 sxxxxx	2 + 6	8
sdAltitude	0	*82 %*	0	1	1
(Δ-Höhe)	± 2	*98 %*	10 sx	2 + 2	4
	± 32	*100 %*	11 sxxxxx	2 + 6	8
sdSpeed	0	*35 %*	00	2	2
(Δ-Geschwindigkeit)	± 2	*72 %*	01 sx	2 + 2	4
	± 8	*99 %*	10 sxxx	2 + 4	6
	± 32	*100 %*	11 sxxxxx	2 + 6	8
cdNumberOfSatellites	0	*69 %*	0	1	1
(Δ-Satellitenanzahl)	± 2	*99 %*	1 sx	1 + 2	3

Tabelle 62 – Record, Format, Deltablock-Data

Im Anschluß auf diesen Teilblock folgt immer ein neuer Deltablock-Start.

BEFEHLSSTROMGRÖßE

Aus diesen Daten kann man nun einfach die durchschnittlich zu erwartende Größe eines Blocks und damit auch die Routengröße bestimmen:

Variable	Berechnung	ø-Größe
Deltablock-Start	(99% * 1) + (1% * 17)	1.16 Bit
sddLatitude	(56% * 1) + (41% * 4) + (3% * 8)	2.44 Bit
sddLongitude	(56% * 1) + (41% * 4) + (3% * 8)	2.44 Bit
sdAltitude	(82% * 1) + (16% * 4) + (2% * 8)	1.62 Bit
sdSpeed	(35% * 2) + (37% * 4) + (27% * 6) + (1% * 8)	3.88 Bit
cdNumberOfSatellites	(69% * 1) + (31% * 3)	1.62 Bit
		13.16 Bit

Tabelle 63 – Record, Format, Blockgröße

Dieser Kompressionsalgorithmus erzeugt also durchschnittlich pro Sekunde 13.16 Bit Daten. Zur Erinnerung: Zuvor haben wir noch 80 Bit benötigt. Somit sinkt auch der Verbrauch pro Stunde von 36 kB/h auf unter 6 kB/h. Somit benötigt ein Jahr Daueraufzeichnung etwa 52 MB.

GRAPHISCHER BEFEHLSSTROM

Graphisch dargestellt sieht der Befehlsstrom folgendermaßen aus:

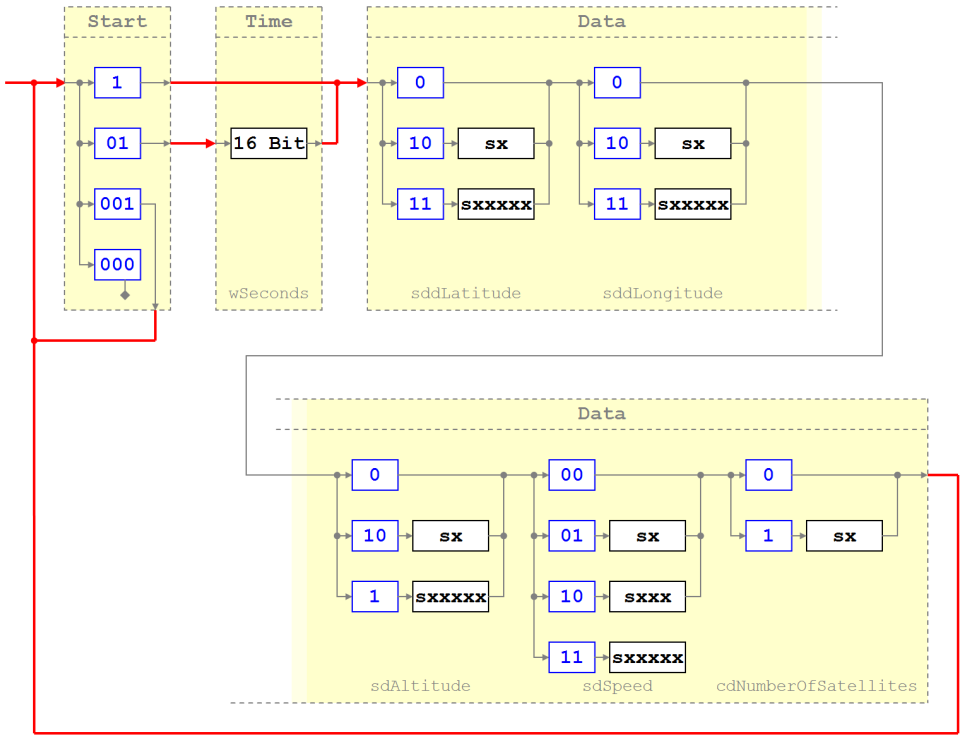


Abbildung 143 – Record, Befehlsstrom

4.6.5. IMPLEMENTIERUNG KRITISCHER STELLEN

Die Implementierung folgt streng den obigen Angaben. Die ausführliche Beschreibung würde erneut wieder zu weit gehen, der Quellcode ist aber gut dokumentiert.

Nur ein kleiner, beispielhafter Auszug des Hinzufügens des `sdAltitude` Wertes sei hier gestattet, der repräsentativ auch für die anderen Werte steht:

```
[…]
// . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
// altitude

// select correct size
if ( _DIFF_.sdAltitude_Meter == 0 )
{
    // add "0"
    BitStream_Add0();
}
else if ( abs( _DIFF_.sdAltitude_Meter ) <= 2 )
{
    // add "10sx"
    BitStream_Add1();
    BitStream_Add0();
    BitStream_Add_sx( _DIFF_.sdAltitude_Meter, 1 );
}
else
{   // altitude > 2
    // check boundaries
    SET_BOUNDARIES( _DIFF_.sdAltitude_Meter, 32 );

    // add "11sxxxxx"
    BitStream_Add1();
    BitStream_Add1();
    BitStream_Add_sx( _DIFF_.sdAltitude_Meter, 5 );
}
[…]
```

Listing 63 – Record, Befehlsstrom aufbauen

`_DIFF_` ist ein Define-Verweis auf die DIFF-Datenstruktur. `abs(…)` ist eine schnelle Funktion, um den Betrag eines Wertes zu finden und `SET_BOUNDARIES(…)` ist ein Makro, das dafür

sorgt, daß die angegebene Variable innerhalb `±limit` – dem zweiten Parameter – bleibt und setzt sie entsprechend.

4.6.6. Berechnungen, Analysen und Statistiken

Hier eine Aufstellung des ungefähr benötigten Speichers für den Record, inklusive den benötigten Treibern, Aufsätzen und sonstigen Modulen. Anhand der Liste erkennt man, wie verzweigt das Record-Modul ist: MMC, Storage, GPS, GPSFormat, Options, IO, LCD, Timer, CmConvert:

Speicherart	Größe [Bytes]
Programmcode	11846
RAM	742

Tabelle 64 – Record Speicherverbrauch

4.7. Ablaufsteuerung

4.7.1. Menü

4.7.1.1. Aufgabe

Dieses Projekt setzt Menüs zur Ablaufsteuerung ein, also zur Verzweigung des Programms durch den Benutzer.

Mit Hilfe dieses Moduls soll man ein Menü möglichst einfach definieren und dann darstellen können, um den Benutzer eine Wahl treffen zu lassen.

4.7.1.2. Interface

Dieses Modul besitzt nur eine einzige Funktion, die das Menu lädt, anzeigt, den Anwender auswählen läßt und anschließend die Wahl zurückliefert:

```
//-- Menu ----------------------------------------------------------------------
    BYTE Menu_ShowAndSelect( const tMenu_Definition *pgmpMenu );
//------------------------------------------------------------------------------
```

Listing 64 – Menü, Interface

Übergeben wird ein Pointer auf den Typ `tMenu_Definition`, der so definiert ist:

```
//------------------------------------------------------------------------------
// Typedefs
//------------------------------------------------------------------------------
typedef struct {                        // (careful: PROGMEM)
    PGM_P   cp_cstrMenuTitle;             // menu title (careful: PROGMEM)
    BYTE    nItems;                       // count of menu items
    PGM_P   ca_cstrMenuItems[];           // array of menu items (careful: PROGMEM)
} tMenu_Definition PROGMEM;
//------------------------------------------------------------------------------
```

Listing 65 – Menü, Interface, tMenu_Definition

4.7.1.3. Definition eines Menüs

Um ein Menü anzeigen lassen zu können, muß es zuerst definiert werden. Um die umständliche Handhabung mit PROGMEM-Variablen zu vermeiden und die Sache zu vereinfachen, wurden Makros eingeführt.

Eine einfache Menüdefinition sieht zum Beispiel wie folgt aus und muß sich außerhalb einer Funktion, also im Global-Space, befinden:

```
MENU_DEFINE_ITEM( menu_Main, 1, "Menu Item 1!" )
MENU_DEFINE_ITEM( menu_Main, 2, "Menu Item 2!" )
MENU_DEFINE_ITEM( menu_Main, 3, "Menu Item 3!" )

MENU_CREATE( menu_Main, "The Main Menu", 3 )
    MENU_ADD( menu_Main, 1 )
    MENU_ADD( menu_Main, 2 )
    MENU_ADD( menu_Main, 3 )
MENU_END()
```

Listing 66 – Menü, Interface, Menüdefinition

Intern baut es auf den oben definierten Typ auf und wird umgesetzt in:

```
const char menu_Main1[] PROGMEM = "Menu Item 1!";
const char menu_Main2[] PROGMEM = "Menu Item 2!";
const char menu_Main3[] PROGMEM = "Menu Item 3!";

const char menu_MainTitle[] PROGMEM = "The Main Menu";

tMenu_Definition menu_Main = {
    menu_MainTitle,
    3,
    {
    menu_Main1,
    menu_Main2,
    menu_Main3,
    }
};
```

Listing 67 – Menü, Interface, Menüdefinition manuell

Und letztendlich der eigentliche Aufruf des Menüs:

```
BYTE bySelected = Menu_ShowAndSelect( &menu_Main );
```

Listing 68 – Menü, Interface, Menüdaufruf

Diese Funktion liefert entweder die ID des ausgewählten Eintrags – der bei der Definition angegeben wurde – zurück oder `MENU_ABORTED`, falls der Benutzer abgebrochen hat oder das Gerät heruntergefahren werden muß.

4.7.1.4. Menüfunktionalität und -Design

Ein Beispielmenü sieht so aus:

Abbildung 144 – Menü, Beispielmenü

Oder hier noch mal in Textform:

```
 The Main Menu
----------------------------------------
 → 1. Menu Item 1!
   2. Menu Item 2!
   3. Menu Item 3!
   4. Menu Item 4!
   ...
```

Abbildung 145 – Menü, Beispielmenü

Oben steht der Titel des Menüs, darunter ein Separator und unten die einzelnen Elemente des Menüs. Links neben den Einträgen fügt das Menü automatisch eine laufende Nummer hinzu. Die aktuelle Auswahl wird durch den Pfeil gekennzeichnet. Am rechten Rand befindet sich eine Scroll-Leitste, die in Abhängigkeit des markierten Eintrags und der Gesamtanzahl der Einträge, die Position in der gesamten Liste kenntlich macht. In unserem Fall ganz oben.

Paßt das Menü nicht ganz auf das Display, so wird es abgeschnitten. Damit der Benutzer weiß, daß das Menü noch weiter geht, werden unten drei Punkte (…) angezeigt. Diese Technik wird natürlich auch in die andere Richtung angewandt: Sobald also noch Einträge vor/nach dem ersten/letzen sichtbaren vorhanden sind, macht das Menü das mit „…“ deutlich.

Zur Steuerung kommen nun unsere Taster zum Einsatz. Die Auf- und Ab-Tasten markieren einen Menüeintrag, der mit OK ausgewählt wird. Wenn der Anwender ESC betätigt, so liefert das Menü `MENU_ABORTED` zurück.

4.7.2. Das Hauptprogramm

Das Hauptprogramm initialisiert zuerst die Treiber mit den entsprechenden `PortInit(…)`- und `Init(…)`-Aufrufen. Anschließend lädt es die Optionen aus dem EEPROM und setzt diese. Danach wird das Hauptmenü solange ausgeführt, bis das Gerät herunterfahren soll.

Hier eine graphische Darstellung der Abläufe in `main.cpp`:

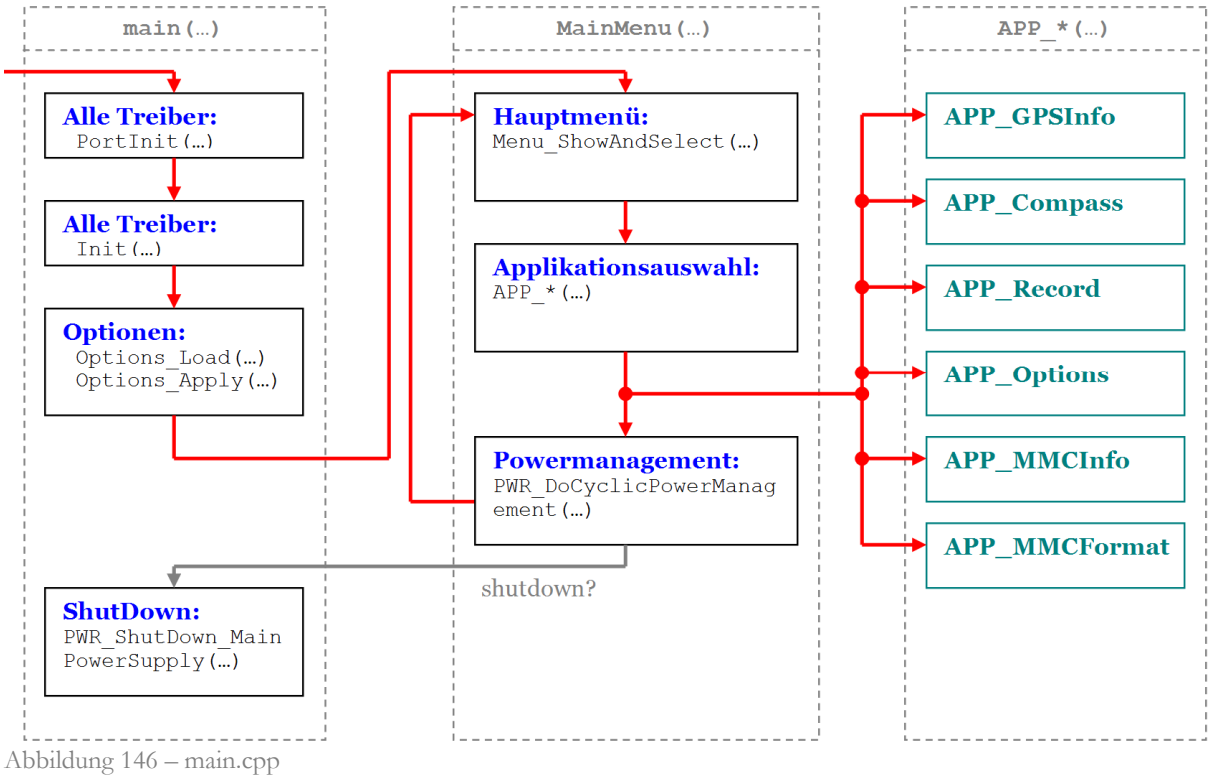


Abbildung 146 – main.cpp

4.8. APPLIKATIONEN

4.8.1. AUFBAU EINER APPLIKATION

Eine Applikation ist eine Anwendung, die auf dem Gerät läuft. Sie kann alle Treiber und Module verwenden. Alle Menüpunkte, die der Benutzer nach dem Systemstart aus dem Hauptmenü auswählen kann, sind Applikationen.

Um dem Gerät ein einheitliches Gesicht zu geben, müssen sich die Anwendungen an einen gewissen Rahmen halten. Dieser beschreibt grob die Darstellung, die Interaktion und die Verhaltensweise.

Jede Applikation definiert genau eine einzige globale Funktion mit dem Namen der jeweiligen Anwendung. Diese Funktion besitzt weder Parameter, noch einen Rückgabewert.

Alle Applikationen, die GPS-Signale verwenden, müssen zusätzlich `APP_General` – eine Sammlung von Hilfsfunktionen – einbinden und verwenden. Der Aufbau von `APP_General` wird im nächsten Abschnitt genauer beschrieben.

Der Grundaufbau in einer Kurzzusammenfassung sieht so aus:

```
_public void APP_Name( void )
{
    // init
    InitTheApplication();

    // loop
    while ( TRUE )
    {
        // only display data, if signal is valid
        if ( APP_GPSStatus( APP_GPSSTATUS_SHOW_ERRORS ) == GPS_STATE_VALID )
        {
            // *** background *********************************
            if ( necessary )
                DrawApplicationBackground();

            // *** data ***************************************
            // if new GPS data is available
            if ( GPS_NewDataAvailable() )
            {
                // convert data
                GPSFormat_Convert();

                // data
                DrawApplicationData();
                DoSomethingWithTheData();
            }
        }

        // look for keypress (returns IO_KEY_NONE if none)
        switch ( IO_Key_GetMsg() )
        {
            // handle keypress
        }

        // do power management
        switch ( PWR_DoCyclicPowerManagement() )
        {
            // handle power management
        }
    }
}
```

Listing 69 – Applikation, Grundaufbau Pseudocode

Zuerst initialisiert `InitTheApplication(...)` die notwendigen Daten, unter anderem löscht es auch das LCD-Display und initialisiert die `APP_General`-Funktionalität.

Danach kommt die Applikation in eine Endlosschleife, die immer folgende Schritte ausführt: Bei gültigen GPS-Daten diese empfangen, den Hintergrund zeichnen und die Daten einzeichnen. Bei ungültigen GPS-Daten übernimmt `APP_General` die Anzeige der Fehlermeldungen.

Anschließend sollten die Tastendrücke abgefragt und entsprechend behandelt werden.

Und zuletzt muß die Funktion `PWR_DoCyclicPowerManagement(…)` aufrufen, damit eventuell auftretende Ausnahmen auch korrekt behandelt werden können. Zur Erinnerung noch einmal: Diese Funktion liefert zum Beispiel auch zurück, falls das Gerät heruntergefahren werden soll. Dann muß die Anwendung alle nötigen Daten speichern und sich möglichst schnell beenden.

Eine funktionsfähige Applikation, mit Statusmaschine zur Verarbeitung mehrerer Anwendungsseiten, kann dann wie folgt aussehen. Dieser Aufbau ist auch der Grundaufbau für alle Applikationen:

```
/*****************************************************************************************
| App: <Name>
|---------------------------------
| <Description>
+----------------------------------------------------------------------------------------
| @Params:          none
| @Return Value:    none
+****************************************************************************************/
public void APP_Name( void )
{
    // init state with page 1
    BYTE    byState         = APP_NAME_STATE_PAGE1;
    BYTE    byOldState      = 0xff;

    // init
    LCD_ClearAndResetScreen_Fast();
    APP_GPSStatus_Reset();

    // loop
    while ( TRUE )
    {
        // only display data, if signal is valid
        if ( APP_GPSStatus( APP_GPSSTATUS_SHOW_ERRORS ) == GPS_STATE_VALID )
        {
            // *** background ********************************
            // to avoit drawing each loop (flickering), the background must
            // only be drawn on state change:
            // if state changed
            if ( byState != byOldState )
            {
                // set state
                byOldState = byState;

                // draw static background
                switch ( byState )
                {
                    // page 1
                    case APP_NAME_STATE_PAGE1:
                        APP_Name_Page1_Static();
                        break;

                    // ...
                }
            }

            // *** data **************************************
            // if new GPS data is available
            if ( GPS_NewDataAvailable() )
            {
                // convert data
                GPSFormat_Convert();

                // select active page and draw content
                switch ( byState )
                {
                    // page 1
                    case APP_NAME_STATE_PAGE1:
                        APP_Name_Page1_Content();
                        break;

                    // ...
                }
            }
        }
        else
        {
            // no valid data
            // invalidate old state
            byOldState = 0xff;
        }

        // look for keypress (returns IO_KEY_NONE if none)
        switch ( IO_Key_GetMsg() )
        {
            case IO_KEY_UP:
            case IO_KEY_DOWN:
            case IO_KEY_MENUOK:
```

```
                // do something
                break;

            // ESC
            case IO_KEY_ESC:
                // quit app or ask
                return;

            // -none-
            case IO_KEY_NONE:
                // do nothing
                break;
        }

        // do power management
        switch ( PWR_DoCyclicPowerManagement() )
        {
            case PWRSTATUS_OK:
                break;

            case PWRSTATUS_LOBAT:
                break;

            case PWRSTATUS_SHUTDOWN:
                // quit app
                return;
        }
    }
}
```

Listing 70 – Applikation, Grundaufbau

4.8.2. APP_GENERAL

Wie im vorherigen Abschnitt schon beschrieben, stellt APP_General allgemeine Funktionalität für Anwendungen zur Verfügung:

```
//-- App -------------------------------------------------------------------------------
    void    APP_GPSStatus_Reset( void );
    BYTE    APP_GPSStatus( BYTE byFlags );
//--------------------------------------------------------------------------------------

//-- error message handling ------------------------------------------------------------
    void    APP_ShowError_REC( REC_RET byRet );
    void    APP_ShowMessage( const PGM_P pgmp_cstrTitle, const PGM_P pgmp_cstrMessage );
//--------------------------------------------------------------------------------------
```

Listing 71 – APP_General Interface

Zuerst muß jede Anwendung beim Start die Funktion `APP_GPSStatus_Reset(…)` aufrufen. Bei jedem Durchgang der Anwendungsschleife ist dann die andere Funktion `APP_GPSStatus(…)` aufzurufen.

Mit dem Parameter kann man über eine Bitkombination – unter anderem – festlegen, ob man nur den Status der GPS-Daten zurückbekommen möchte oder ob sich die Funktion gleich auch noch um die Darstellung der Fehlermeldung kümmern soll, was empfehlenswert ist.

Die möglichen Rückgabewerte sind:

```
#define     GPS_STATE_VALID             1           // signal valid
#define     GPS_STATE_INVALID           2           // signal invalid
#define     GPS_STATE_NOTCONNECTED      3           // device not connected
```

Listing 72 – APP_General, Rückgabewerte

Nur bei `GPS_STATE_VALID` sollte die Anwendung auch die GPS-Daten verwenden und darstellen.

Die beiden anderen Funktionen zeigen (Fehler-) Meldungen an und reagieren auf Tastendruck.

Falls das Signal ungültig ist, zeigt `APP_GPSStatus(...)` folgendes an:

```
GPS-Fehler
------------------------------------------

  GPS-Signal ungültig (Kein Empfang):
    -> Warte auf Synchronisation...
       Dies kann einige Minuten dauern.
```

Abbildung 147 – APP_GPSInfo, Screen 1

Falls sogar gar kein Empfänger angeschlossen sein sollte, kommt diese Meldung:

```
GPS-Fehler
------------------------------------------

  GPS-Empfänger nicht angeschlossen
    -> Bitte Empfänger anschießen...
```

Abbildung 148 – APP_GPSInfo, Screen 1

Viele Displaybilder werden am Ende dieses Kapitels nochmals als Foto gezeigt.

4.8.3. APP_GPSINFO

Diese Anwendung soll dem Benutzer einen Gesamtüberblick über die empfangenen und berechneten GPS-Werte verschaffen und besteht aus drei Seiten:

- Gesamtübersicht (1/3)
- Positionsübersicht (2/3)
- Zeitübersicht (3/3)

Der Aufbau ist sehr einfach und geradlinig und baut, wie alle anderen Applikationen auch, auf dem Grundgerüst auf. Es werden, je nach aktiver Seite, zuerst der Hintergrund gezeichnet und anschließend die Daten darin eingefügt.

Durch die einzelnen Seiten kann man mit `UP` und `DOWN`, bzw. `OK`, navigieren. Mit `ESCAPE` beendet sich die ganze Anwendung und kehrt in das Hauptmenü zurück.

Es folgt eine Nachstellung der drei Displayseiten:

```
GPS-Informationen: Gesamtübersicht (1/3)
------------------------------------------
Datum / Uhrzeit:  21.10.2005  23:55:21 GMT
Breitengrad:                   48°16.832 N
Längengrad:                    10°52.851 E
Höhe über Meeresspiegel:             577 m
Anzahl der Sateliten:                  7
```

Abbildung 149 – APP_GPSInfo, Screen 1

```
GPS-Informationen: Positionsübersicht(2/3)
------------------------------------------
Breite/Länge:   48°16.832 N    10°52.851 E
 Konvertiert:     2896832         652851
Höhe über Meeresspiegel:             577 m
Richtung / Geschwindigkeit:        0° (N)
    0.0 Knoten       0.0 m/s         0 km/h
```

Abbildung 150 – APP_GPSInfo, Screen 2

```
GPS-Informationen: Zeitübersicht (3/3)
------------------------------------------
Datum / Uhrzeit:  21.10.2005  02:52:39 GMT
Tage seit 01.01.2000:              2121 Tage
Sekunden seit Mitternacht:        10359 Sek.
Sekunden seit 01.01.2000:    183264759 Sek.
Komprimiertes Datum/Zeit-Feld:   380251431
```

Abbildung 151 – APP_GPSInfo, Screen 3

4.8.4. APP_COMPASS

Dem Anwender bietet die Kompaß-Applikation einen digitalen Kompaß. Der Benutzer bewegt sich in eine Richtung und sieht auf dem Display einen virtuellen Kompaß, der in diese Richtung zeigt.

Auch hier gibt es nicht nur eine einzelne Seite sonder zwei. Der erste Kompaß ist stufenlos drehbar, der zweite nur in festen Winkeln. Es wird immer ein Halbkreis als Kompaß dargestellt, in dem die aktuelle Richtung nach oben zeigt. Bei der zweiten Kompaßart stehen die Himmelsrichtungen an festen Winkelpositionen, bei 0°, 22.5°, 45°, 67.5°, 90° und bei den entsprechend negativen Werten. Beim bewegbaren Kompaß wandern die Textbezeichnungen der Richtungen mit und man kann diese noch genauer bestimmen:

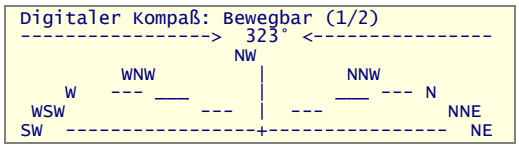


Abbildung 152 – APP_Compass, Screen bewegbar

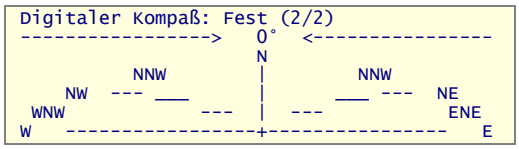


Abbildung 153 – APP_Compass, Screen fest

4.8.5. APP_OPTIONS

APP_Options verwendet keine GPS-Informationen, sondern bietet nur drei weitere Auswahlmenüs, um die Optionen entsprechend der Umstände und den eigenen Vorlieben einzustellen.

Zuerst muß der Benutzer die Hintergrundbeleuchtungsart wählen:

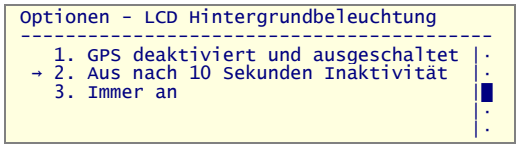


Abbildung 154 – APP_Options, Screen 1

Anschließend ist die Eingangsquelle an der Reihe:

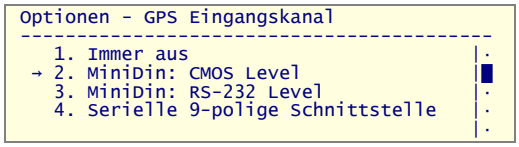


Abbildung 155 – APP_Options, Screen 2

Und zuletzt die Speicherart bei Routenaufzeichnung. Wird hier die erste Einstellung gewählt, so ruft der Record-Treiber bei einem `Sync(…)`-Aufruf keine `Flush(…)`-Funktion des Storage-

Treibers auf. Somit wird ein Sektor nur auf Karte geschrieben, wenn er voll ist. Falls die Karte zwischenzeitlich entfernt wurde, sind die Daten ab dem letzten Sektor verloren, was bis zu 6 Minuten ausmachen kann. Bei der zweiten Einstellung werden die Daten jede Sekunde neu geschrieben. Es kann also praktisch keinerlei Datenverlust geben, jedoch steigt der Stromverbrauch an und die Flash-Karte wird überdurchschnittlich beansprucht.

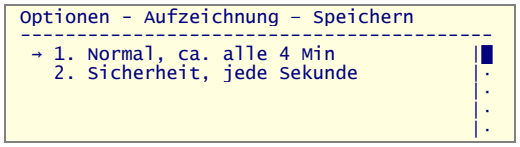


Abbildung 156 – APP_Options, Screen 3

Sobald der Benutzer eine Auswahl getroffen hat, speichert die Applikation die Optionen ab und aktiviert die Auswahl.

4.8.6. APP_MMCINFO

`APP_MMCInfo` gibt einen groben Überblick über den Inhalt einer MMC-Karte und stellt wichtige Daten dar:

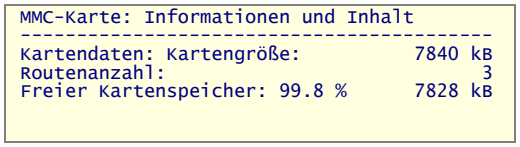


Abbildung 157 – APP_MMCInfo, Screen

4.8.7. APP_MMCFORMAT

Mit Hilfe dieser Funktion ist eine volle oder neue Speicherkarte zu formatieren. Nach einer Sicherheitsabfrage löscht die Anwendung alle Daten auf der Karte und initialisiert ein neues Storage-Dateisystem:

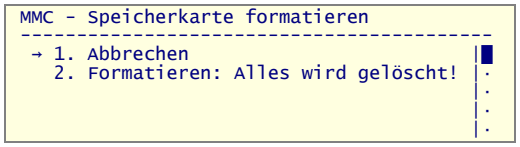


Abbildung 158 – APP_MMCFormat, Screen 1

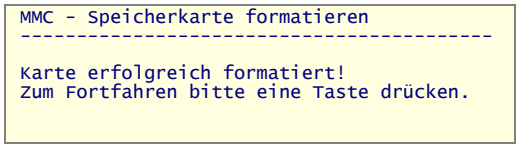


Abbildung 159 – APP_MMCFormat, Screen 2

4.8.8. APP_RECORD

Die Hauptanwendung für das Gerät ist sicherlicht `APP_Record`, das die Record-Bibliothek verwendet und somit Routen aufzeichnet. Zusätzlich werden die wichtigsten GPS-Daten und Informationen über die Route und die Karte selbst angezeigt.

Der typische Aufnahmebildschirm sieht so aus:

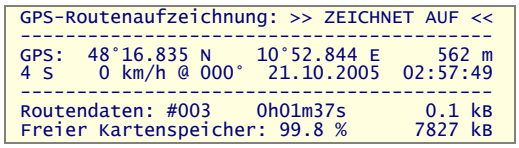


Abbildung 160 – APP_Record, Screen 2

Im oberen Abschnitt ist eine Zusammenfassung der GPS-Daten zu sehen: Breiten- und Längengrad, Höhe, Satellitenanzahl, Geschwindigkeit, Richtung, Uhrzeit und Datum.

Unten befindet sich die Routen-ID, bisherige Aufnahmezeit, Routengröße in Kilobytes und der freie Kartenspeicher, in Prozent und in Kilobytes.

Wenn die Anwendung aktiv ist, dann blinkt sowohl die Record-LED, als auch der Text `>> ZEICHNET AUF <<` rechts oben im Display.

Damit die Route nicht versehentlich beendet wird, gibt es beim Beenden der Anwendung noch eine Sicherheitsabfrage, die im folgenden Abschnitt der Fotos noch zu sehen ist.

4.8.9. Fotos

Hier soll zur besseren Vorstellung des Betriebs des Geräts einige Fotos in Aktion gezeigt werden:

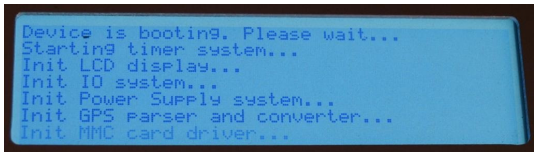


Abbildung 161 – Applikationen 1

Beim Druck auf die Powertaste startet das Gerät.

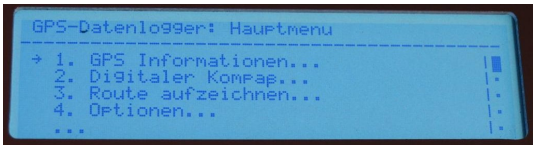


Abbildung 162 – Applikationen 2

Nach dem Bootvorgang befindet sich der Anwender im Hauptmenü.

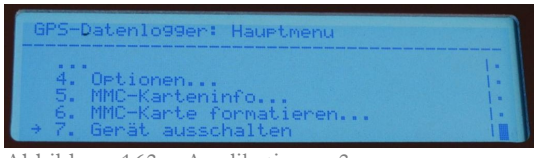


Abbildung 163 – Applikationen 3

Das Menü besteht aus 7 Einträgen.

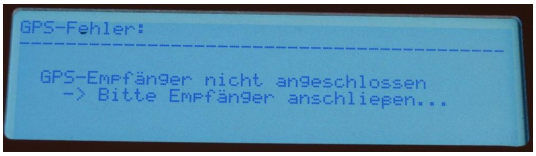


Abbildung 164 – Applikationen 4

Wird eine Applikation gestartet und ist kein Empfänger angeschlossen, so kommt diese Fehlermeldung.

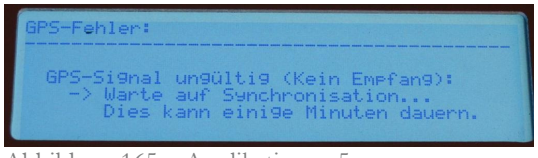


Abbildung 165 – Applikationen 5

Falls der Empfänger noch nicht synchronisiert ist, wird diese Warnung gezeigt.

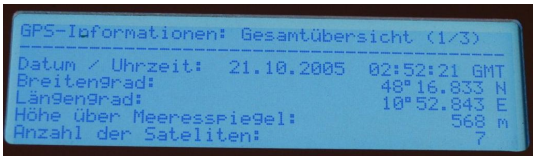


Abbildung 166 – Applikationen 6

`APP_GPSInfo` mit der ersten Seite in Aktion.

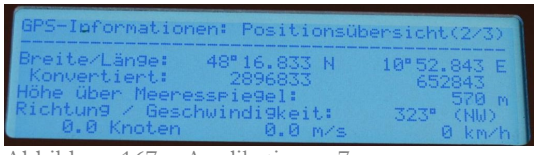


Abbildung 167 – Applikationen 7

Die zweite Seite …

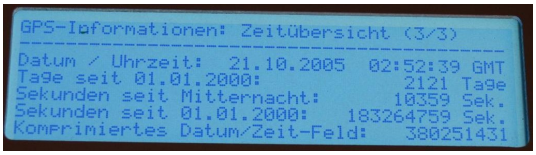


Abbildung 168 – Applikationen 8

… und die Letzte mit den Zeitinfos.

Abbildung 169 – Applikationen 9

Der digitale Kompaß bei 0° Winkel …

Abbildung 170 – Applikationen 10

… und bei 323°.

Abbildung 171 – Applikationen 11

Hier nimmt die Applikation gerade eine Route auf.

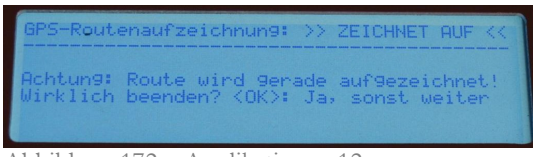


Abbildung 172 – Applikationen 12

Bei allen Anwendungen kommt man mit der ESCAPE-Taste sofort zurück, nur beim Aufzeichnen wird vorsichtshalber noch einmal nachgefragt.

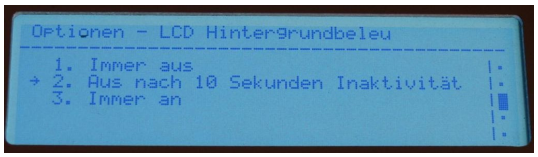


Abbildung 173 – Applikationen 13

Möchte man die Optionen neu einstellen, muß man hier die erste Auswahl, bezüglich der Hintergrundbeleuchtung, treffen.

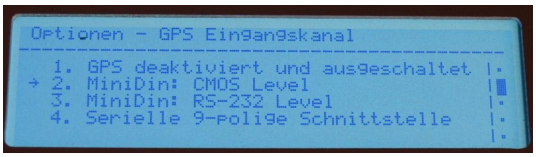


Abbildung 174 – Applikationen 14

Anschließend ist die Eingangsquelle aus der Liste zu wählen.

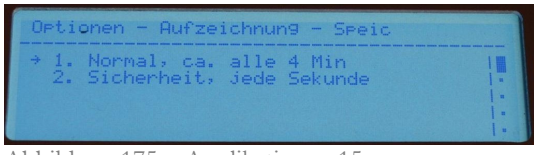


Abbildung 175 – Applikationen 15

Und zuletzt noch die Speicherungsart.

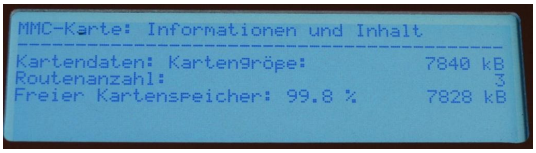


Abbildung 176 – Applikationen 16

Hier zeigt `APP_MMCInfo` Informationen über die eingelegte Speicherkarte an.

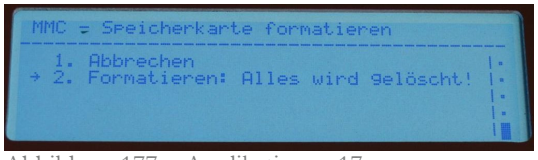


Abbildung 177 – Applikationen 17

Das letzte Bild zeigt die Anwendung `APP_MMCFormat`, kurz vor ihrem Einsatz.

5. PC SOFTWARE

5.1. AUFBAU

Der Schwerpunkt dieser Diplomarbeit liegt bei der Hardware und der Programmierung des mobilen Gerätes. Die PC-Software zum Auswerten soll nur zeigen, welche Möglichkeiten auf der anderen Seite bestehen. Deshalb fällt dieses Kapitel, wie auch die Quelltexte und Programmfunktionen, relativ kurz aus.

Der Ausgangspunkt ist, daß sich die aufgezeichneten Routen auf der MMC-Karte befinden.

Dieses Kapitel soll die drei vorhandenen PC-Programme und deren Funktionsweise kurz vorstellen:

- **StorageReader** liest die MMC-Karte aus und konvertiert die Routen aus dem Record-Dateisystem in vom PC lesbare Dateien. Anschließend werden die gepackten Routen noch entpackt, damit sich der Zugriff auf die Daten für andere Programme vereinfacht.

- **STGDump** ist ein einfaches Konsolenprogramm, das die einzelnen Datensätze der Reihe nach stur ausgibt.

- **STGView** ist ein Windows GUI-Programm, welches die MFC[26] benutzt. Damit lassen sich mit verschiedenen Ansichten die Daten betrachten. Es gibt eine Tabellenansicht, ähnlich dem STGDump Programm, ein Höhenprofil und eine Landkartenansicht.

5.2. STORAGEREADER

StorageReader besteht im Groben aus zwei Teilen. Der Erste zum Auslesen der Routen und der Zweite zum Entpacken.

5.2.1. AUSLESEN

Das Auslesen beinhaltet das Lesen der Daten durch direkten Sektorzugriff von der MMC-Karte und das anschließende Übertragen der einzelnen Routendaten in vom PC lesbare Dateien. Für jede auf dem Gerät gespeicherte Route wird eine Datei angelegt und die komprimierten Daten direkt 1:1 kopiert.

[26] Microsoft Foundation Classes, Homepage: www.msdn.com

5.2.1.1. Direkter Sektorzugriff, Theorie

Zum Auslesen der Daten muß ein direkter Zugriff auf die Sektoren der MMC-Karte möglich sein. Unter Windows geht das sehr einfach, man braucht allerdings – verständlicherweise – Administratorrechte. Bei einer öffentlichen Version, wo Administratorrechte nicht vorausgesetzt werden können, müßte zusätzlich noch ein Treiber geschrieben werden, der den direkten, lesenden Zugriff auf Speicherkarten ermöglicht.

Auf Geräte kann man unter Windows durch Dateien zugreifen. Der Aufbau von Speichermedien ist:

`Disk → Volume → File`

Dabei ist `Disk` das physikalische Medium, `Volume` die Partition und `File` die Datei im Dateisystem. Da unser Storage-Dateisystem keine Partitionen besitzt sondern direkt im Sektor 0 anfängt, brauchen wir den `Disk`-Zugriff auf niedrigster Ebene.

Um das Speichermedium zu öffnen, müßten wir aber den Namen des geladenen Treibers für diese Karte herausfinden. Windows bietet uns aber noch eine andere, wesentlich einfachere Möglichkeit. Da das System die Kartenleser automatisch erkennt und damit auch automatisch Laufwerksbuchstaben hinzufügt, müssen wir nur diesen Laufwerksbuchstaben für unsere MMC-Karte kennen. Der Gerätename heißt „`\\.\<drive>:`", wobei `<drive>` der Laufwerksbuchstabe ist. Wenn die MMC-Karte auf `m:\` gemountet ist, dann lautet der Gerätename: „`\\.\m:`".

Mit folgendem einfachen Aufruf wird die `Disk` geöffnet:

```
sprintf( strFileName, "\\\\.\\%c:", cDrive );
hFile = fopen( strFileName, "rb" );
```

Listing 73 – StorageReader, SectorAccess, Open

Um auf die einzelnen Sektoren zuzugreifen, muß man das mit 512 Byte – also der Sektorgröße – ausgerichteten Daten machen.

5.2.1.2. SectorAccess

Die Funktionalität zum direkten Sektorzugriff steht in SectorAccess. Das Interface sieht so aus:

```
//-- Init and start -----------------------------------------------------------
    bool SecorAccess_Init( char cDrive );
    void SecorAccess_Close();
    bool SecorAccess_Read( void *psecSectorData, DWORD nSector );
//-----------------------------------------------------------------------------
```

Listing 74 – StorageReader, SectorAccess, Interface

Die Implementierung ist auch nicht viel größer:

```
FILE *hFile = NULL;

bool SecorAccess_Init( char cDrive )
{
    SecorAccess_Close();

    char    strFileName[8];
    sprintf( strFileName, "\\\\.\\%c:", cDrive );
    hFile = fopen( strFileName, "rb" );
    return ( hFile != NULL );
}
```

```
void SecorAccess_Close( void )
{
    if ( hFile )
    {
        fclose( hFile );
        hFile = NULL;
    }
}

bool SecorAccess_Read( void *psecSectorData, DWORD nSector )
{
    if ( fseek( hFile, nSector * 512, SEEK_SET ) != 0 )
        return false;

    if ( fread( psecSectorData, 512, 1, hFile ) != 1 )
        return false;

    return true;
}
```

Listing 75 – StorageReader, SectorAccess, Implementierung

5.2.1.3. Übertragen der Routen

Das Hauptprogramm funktioniert nach folgendem Prinzip:

- Öffnen der Speicherkarte im direkten Sektorzugriff mit den `SectorAccess`-Funktionen.
- Einlesen des Bootsektors des Storage-Dateisystems mit anschließendem Überprüfen des Bootsektors (`MagicID` und `Version`).
- Einlesen der Zusatzsektoren: `NextFreeSector` und die vier `RouteInfo`-Sektoren, die gleich in eine zusammenhängende Struktur geladen werden.
- Danach für jeden möglichen Routeneintrag in der `RouteInfo`-Liste:
    - Festlegen des Startsektors: Dieser steht direkt in der Liste. Falls dieser Null ist, so kommen keine weiteren Routen mehr (vom Storage-Dateisystem so definiert) und die Schleife wird abgebrochen.
    - Festlegen des Stopsektors: Dieser Wert muß um eins kleiner sein, also der Startsektor der nächsten Route. Falls der nächste Startsektor Null ist, also keine weitere Route mehr folgt, wird als Ende der `NextFreeSector - 1` verwendet.
    - Eine neue Datei wird angelegt, deren Dateiname sich aus dem Datum und der Zeit – aus der `DateTime`-Variablen vom `RouteInfo`-Feld – zusammensetzt. Die Dateiendung für die unkomprimierten Daten ist `.compressed`.
    - Für jeden Sektor zwischen dem berechneten Startsektor und Stopsektor:
        - Auslesen des Sektors von der MMC-Karte
        - Abspeichern des Sektors in die erstellte Datei
    - Neu erstellte Datei wieder schließen.
    - Unterprogramm zum Entpacken der komprimierten Routen aufrufen.
- MMC-Karte wieder schließen.

5.2.2. Entpacken

Dieses Unterprogramm übernimmt das Entpacken der komprimierten Routen aus dem Record-Format in ein unkomprimiertes Format. Dazu lädt es das komprimierte Format wieder Datensatz für Datensatz ein, entpackt diese einzeln und speichert die unkomprimierten Daten wieder einzeln in der Zieldatei ab. Diese bekommt die Dateiendung `.rte` für Route und wird später von allen Visualisierungsprogrammen verwendet.

5.2.2.1. UNKOMPRIMIERTES FORMAT

Dieses unkomprimierte Format ist sehr einfach aufgebaut: An erster Stelle steht ein `CompressedDateTimeStart`-Feld aus vier Bytes, welches den Erstellungszeitpunkt angibt. Anschließend folgen jede Sekunde die Absolutdaten:

```
typedef struct _tGPS_State {
    DWORD   dwQuality;                  // 0=invalid, 1=valid

    DWORD   dwSeconds_SinceYear2000;    // timestamp
    DWORD   dwSecond;                   // timestamp

    long    lLatitude_Minutes1000th;
    long    lLongitude_Minutes1000th;
    long    lAltitude_Meter;

    DWORD   dwSpeed_kmh;
    DWORD   dwNumberOfSatellites;
} tGPS_State;
```

Listing 76 – StorageReader, unkomprimiertes Format

Die einzelnen Datenfelder sind schon aus dem letzten Kapitel bekannt, nur `dwSecond` ist neu. Das ist nicht die absolute Sekundenzahl seit dem 01.01.2000 sondern die absolute Sekundenanzahl seit Beginn der Routenaufzeichnung – startet also immer bei 0.

Diese Struktur ist bewußt 64 Bytes groß (*Annahme: `DWORD` und `long` sind 32-Bit lang, bei der Intel IA-32 Architektur der Fall*), auch wenn sie mit kleineren Datentypen auskommen würde. Aus Performancegründen spielen aber heute – auf dem PC – weniger die reine Speichergröße sondern mehr die Ausrichtung[27] der Daten und die Variablengröße selbst eine wichtigere Rolle. So verarbeitet ein Intel kompatibler Prozessor 32-Bit Werte um einiges effizienter, als 16- oder gar 8-Bit Werte. Intern müssen diese vom Prozessor nämlich erst ausgeschnitten, in 32-Bit umgewandelt, die Operation darauf angewandt, das Ergebnis wieder in die Ausgangsgröße zurückgewandelt und zuletzt wieder eingefügt werden. Selbst ein 1-Bit kleiner Boolean-Wert wird als 32-Bit Wert abgespeichert. Durch die Gesamtausrichtung auf 64 Bytes gibt es auch beim wahlfreien Zugriff auf die einzelnen Datenelemente keine Probleme. Die Cache-Größe bei den heutigen Pentium-Prozessoren ist genau 64 Byte groß. Somit paßt exakt ein Datensatz in jede *Cache Line*. Auführliche Informationen zu diesem Thema in Intels Dokument: *IA-32 Intel® Architecture Optimization Reference Manual* [IA32OPT].

5.2.2.2. BITSTREAM

Den schon bekannten `BitStream` aus `Record` gibt es hier wieder, allerdings in der invertierten Ausführung. Hier werden keine einzelnen Bits in einen Bitstrom geschrieben sondern einzeln ausgelesen.

Das Interface gestaltet sich deshalb recht ähnlich:

```
// create a new BitStream
void BitStream_New( BYTE *pBufferStart )

// add n bits to the BitStream
long BitStream_Get( BYTE byBits )

// sync to next byte in the BitStream
void BitStream_Sync( void )

// add sx formatted bits to the BitStream (sign, and n bits starting with 0 => 1)
long BitStream_Get_sx( BYTE byxBits )
```

Listing 77 – StorageReader, BitStream

Details zur Implementierung befinden sich direkt im Quellcode.

[27] Data Align, siehe [IA32OPT]

5.2.2.3. Dekomprimierung

Zum Verständnis noch einmal das Diagramm für das komprimierte Format aus dem `Record`-Kapitel:

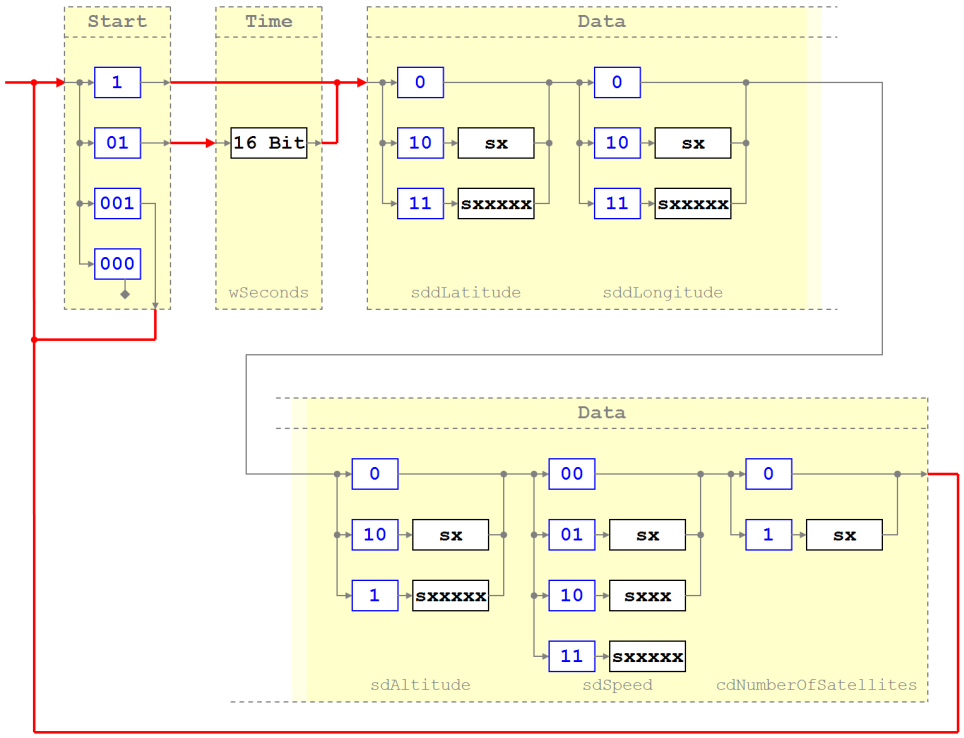


Abbildung 178 – StorageReader, Komprimierter Befehlsstrom

Das Dekomprimieren gestaltet sich im Prinzip genau umgekehrt zu der Vorgehensweise im Record-Modul. Aus den komprimierten, relativen Werten werden wieder unkomprimierte, absolute, die nach jeder Umrechnung in die Zieldatei gespeichert werden.

5.3. STGDump

5.3.1. Einlesen der Route

Zum Einlesen der Route wird die unkomprimierte Routendatei (`*.rte`) verwendet und die Daten am Stück ausgelesen.

Hier eine Version zum Einlesen von Routen in den Speicher – ohne Fehlerüberprüfung:

```
// absolute GPS Status
typedef struct _tGPS_State {
    DWORD   dwQuality;                  // 0=invalid, 1=valid

    DWORD   dwSeconds_SinceYear2000;    // timestamp
    DWORD   dwSecond;                   // timestamp

    long    lLatitude_Minutes1000th;
    long    lLongitude_Minutes1000th;
    long    lAltitude_Meter;

    DWORD   dwSpeed_kmh;
    DWORD   dwNumberOfSatellites;
} tGPS_State;
```

```
tCompressedDateTime     CompressedDateTimeStart;
tGPS_State              *paGPS = NULL;
DWORD                   nGPS = 0;


bool OpenRoute( char *strRoute )
{
    // free if already open
    if ( paGPS )
        free( paGPS );
    paGPS = NULL;

    // open file
    FILE    *f;
    f = fopen( strRoute, "rb" );
    if ( f == NULL )
        return false;

    // get size
    fseek( f, 0, SEEK_END );
    long fSize = ftell( f ) - 4;
    fseek( f, 0, SEEK_SET );

    // read CompressedDateTimeStart
    fread( &CompressedDateTimeStart, sizeof( CompressedDateTimeStart ), 1, f );

    // calc count of data elements
    nGPS = fSize / sizeof( tGPS_State );

    // alloc mem
    paGPS = (tGPS_State*) calloc( nGPS, sizeof( tGPS_State ) );

    // read elements
    fread( &paGPS[0], sizeof( tGPS_State ), nGPS, f );

    // close file
    fclose( f );

    return true;
}
```

Listing 78 – STGDump, Route einlesen

Anschließend sind die Anzahl der Elemente in `nGPS` gespeichert und die Elemente selbst im Array `paGPS`.

5.3.2. Ausgeben der Route

Bei diesem Demonstrationsprogramm sollen die einzelnen Werte nur stur auf der Konsole ausgegeben werden, was im Prinzip so funktioniert:

```
// open disk
printf( "Open route...\n" );
if ( !OpenRoute( argv[1] ) )
    goto _FSERROR;

// print route
for ( DWORD i = 0; i < nGPS; i++ )
{
    printf( "%10d %1d %11d %11d %6d %5d %2d\n", paGPS[i].dwSeconds_SinceYear2000, paGPS[i].dwQuality,
                            paGPS[i].lLatitude_Minutes1000th, paGPS[i].lLongitude_Minutes1000th,
                            paGPS[i].lAltitude_Meter, paGPS[i].dwSpeed_kmh, paGPS[i].dwNumberOfSatellites );
}
```

Listing 79 – STGDump, Ausgabe

Ein kleiner Ausschnitt der Ausgabe aus einer Route:

```
[...]
 182630263 1     2901325      651204    529    71  4
 182630264 1     2901333      651192    529    70  4
 182630265 1     2901340      651181    528    70  4
 182630266 1     2901348      651170    528    68  4
 182630267 1     2901356      651160    528    67  4
 182630268 1     2901364      651151    528    66  4
 182630269 1     2901373      651143    528    67  5
 182630270 1     2901383      651136    528    67  5
 182630271 1     2901392      651129    528    66  4
 182630272 1     2901401      651123    528    65  4
 182630273 1     2901411      651117    528    66  5
 182630274 1     2901421      651112    528    67  5
 182630275 1     2901431      651108    528    67  5
 182630276 1     2901441      651104    528    65  3
 182630277 1     2901452      651101    528    68  5
 182630278 1     2901462      651098    528    68  5
[...]
```

Listing 80 – STGDump, Beispiel Konsolenausgabe

5.4. STGView

STGView ist ein Windows Programm, das die MFC[28] benutzt und die Daten in verschiedenen Ansichten darstellt.

5.4.1. C++ Version von OpenRoute(…)

Die reine C-Version zum Laden einer Route (`OpenRoute(…)`) in `STGDump` hat einen entscheidenden Nachteil: Aufgrund der globalen Variablen kann maximal eine Route gleichzeitig geöffnet sein. Unser STGView Programm kann aber beliebig viele Routen und Ansichten gleichzeitig darstellen. Deshalb muß eine objektorientierte Version von `OpenRoute` verwendet werden. Hier die Definition:

```
class CRoute
{
public:
    CRoute();
    virtual ~CRoute();

    tCompressedDateTime CompressedDateTimeStart;
    tGPS_State          *paGPS;
    DWORD               nGPS;

    bool    Load( const char *strFileName );
};
```

Listing 81 – STGView, CRoute

Den OO-Gurus wird es bei diesem Design sicherlich den Magen umdrehen, es ging hier aber primär nicht um einen möglichst schönen objektorientierten Entwurf sondern eher darum, eine möglichst einfache, funktionsfähige Anwendung zu erstellen, um die Funktionalität zu beweisen beziehungsweise die Funktionsweise aufzuzeigen.

`CRoute` ist also die einfache Umsetzung von `OpenRoute(…)` in eine Klasse. Die früheren globalen Variablen werden hier zu Membervariablen. Somit sind auch mehrere Instanzen, also geladene Routen, gleichzeitig möglich.

5.4.2. Hauptmenü

Nach dem Start des Programms erscheint das Hauptmenü. Darin kann man zuerst nur eine Route aus einer `.rte`-Datei laden und danach aus folgenden Ansichtsarten auswählen:

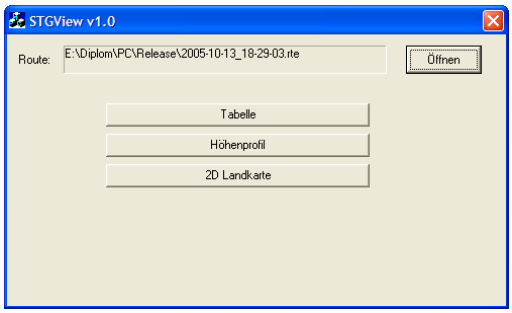
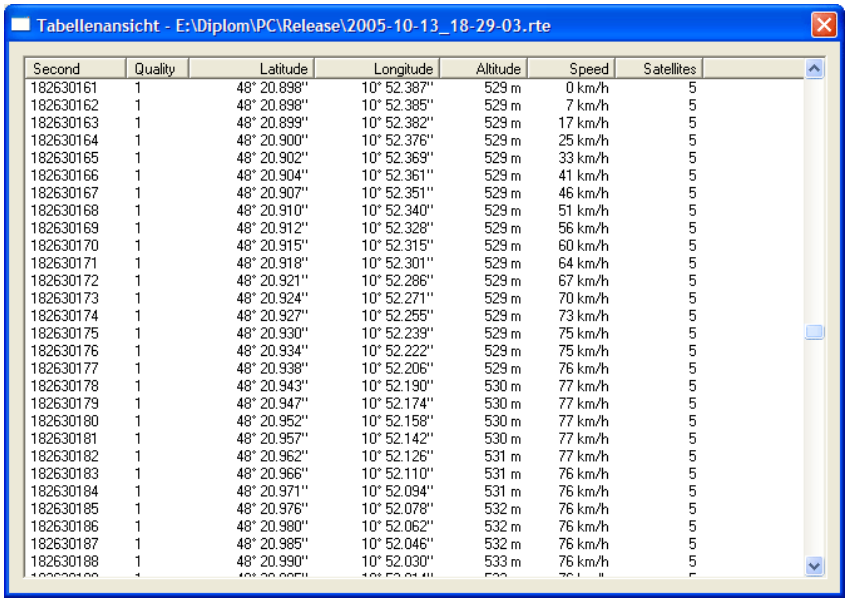


Abbildung 179 – STGView, Hauptmenü

[28] Microsoft Foundation Classes, Homepage: www.msdn.com

5.4.3. TABELLENANSICHT

Die einfachste der drei Ansichten ist die Tabellenansicht. Sie sieht so ähnlich aus wie die Ausgabe von STGDump, benutzt aber zur Anzeige eine ListView:



Second	Quality	Latitude	Longitude	Altitude	Speed	Satellites
182630161	1	48° 20.898''	10° 52.387''	529 m	0 km/h	5
182630162	1	48° 20.898''	10° 52.385''	529 m	7 km/h	5
182630163	1	48° 20.899''	10° 52.382''	529 m	17 km/h	5
182630164	1	48° 20.900''	10° 52.376''	529 m	25 km/h	5
182630165	1	48° 20.902''	10° 52.369''	529 m	33 km/h	5
182630166	1	48° 20.904''	10° 52.361''	529 m	41 km/h	5
182630167	1	48° 20.907''	10° 52.351''	529 m	46 km/h	5
182630168	1	48° 20.910''	10° 52.340''	529 m	51 km/h	5
182630169	1	48° 20.912''	10° 52.328''	529 m	56 km/h	5
182630170	1	48° 20.915''	10° 52.315''	529 m	60 km/h	5
182630171	1	48° 20.918''	10° 52.301''	529 m	64 km/h	5
182630172	1	48° 20.921''	10° 52.286''	529 m	67 km/h	5
182630173	1	48° 20.924''	10° 52.271''	529 m	70 km/h	5
182630174	1	48° 20.927''	10° 52.255''	529 m	73 km/h	5
182630175	1	48° 20.930''	10° 52.239''	529 m	75 km/h	5
182630176	1	48° 20.934''	10° 52.222''	529 m	75 km/h	5
182630177	1	48° 20.938''	10° 52.206''	529 m	76 km/h	5
182630178	1	48° 20.943''	10° 52.190''	530 m	77 km/h	5
182630179	1	48° 20.947''	10° 52.174''	530 m	77 km/h	5
182630180	1	48° 20.952''	10° 52.158''	530 m	77 km/h	5
182630181	1	48° 20.957''	10° 52.142''	530 m	77 km/h	5
182630182	1	48° 20.962''	10° 52.126''	531 m	77 km/h	5
182630183	1	48° 20.966''	10° 52.110''	531 m	76 km/h	5
182630184	1	48° 20.971''	10° 52.094''	531 m	76 km/h	5
182630185	1	48° 20.976''	10° 52.078''	532 m	76 km/h	5
182630186	1	48° 20.980''	10° 52.062''	532 m	76 km/h	5
182630187	1	48° 20.985''	10° 52.046''	532 m	76 km/h	5
182630188	1	48° 20.990''	10° 52.030''	533 m	76 km/h	5

Abbildung 180 – STGView, Tabellenansicht

5.4.4. HÖHENPROFIL

Die nächste Ansicht ist das Höhenprofil, welches die Höhe in Abhängigkeit von der Zeit anzeigt:

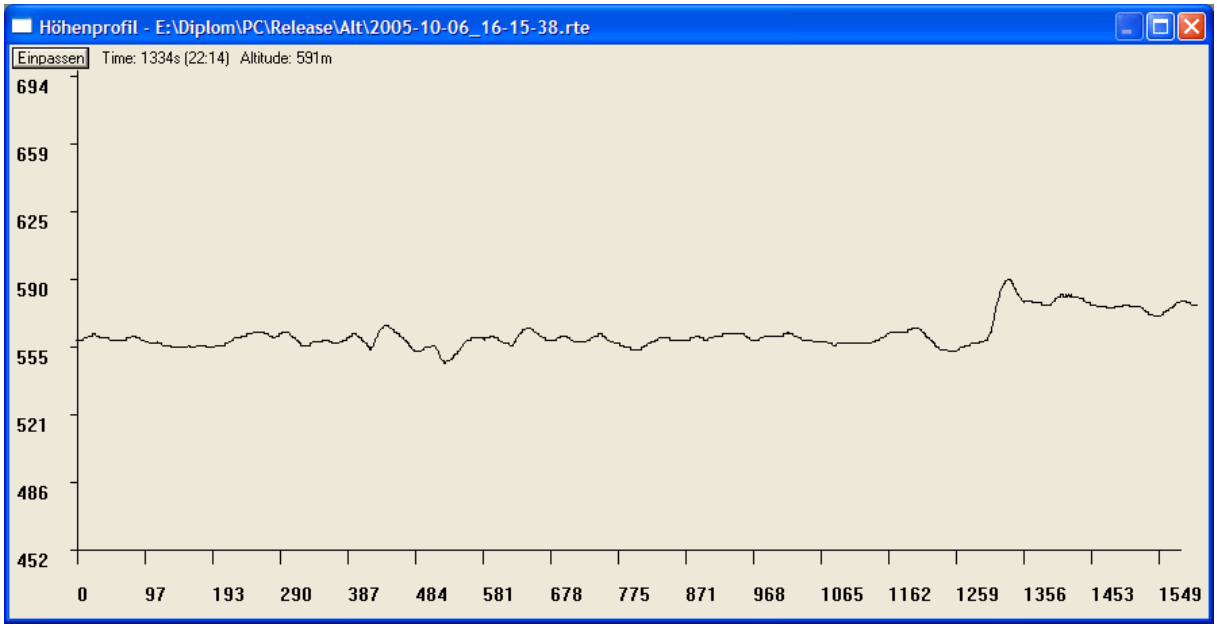


Abbildung 181 – STGView, Höhenprofil

Oben links befindet sich eine Schaltfläche, mit der man die gesamte Route optimal an die aktuell gewählte Fenstergröße anpassen kann. Rechts daneben befindet sich die Information zu der aktuellen Mausposition. Mit der linken Maustaste kann man das Diagramm verschieben und mit der rechten zoomen. Die linke Achse zeigt die Höhe in Metern an und die untere die vergangene Zeit seit Routenaufzeichnung in Sekunden.

5.4.5. 2D-LANDKARTE

Die letzte und aufwendigste Ansicht ist die 2D-Landkarten Ansicht. In ihr wird, wie auf einer Landkarte, die Route eingezeichnet. Zusätzlich zu Breiten- und Längengrad wird aber auch noch die gefahrene Geschwindigkeit mit Hilfe der Farbe visualisiert.

Hier ein paar Beispielbilder:

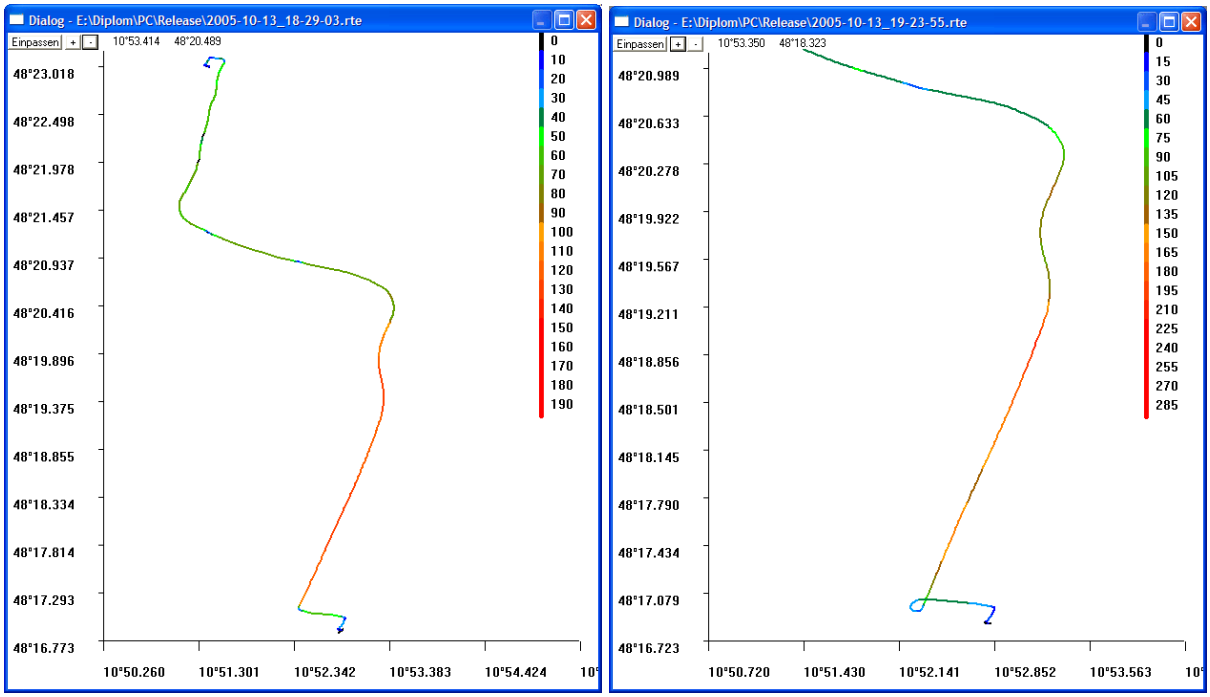


Abbildung 182 – STGView, 2D-Lankartenansicht 1, Hin- und vergrößerte Rückfahrt

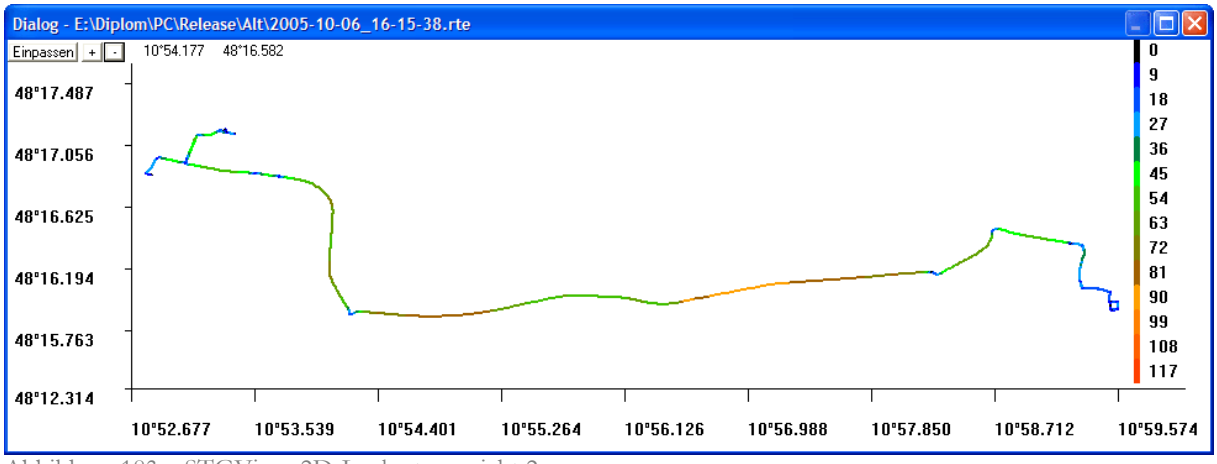


Abbildung 183 – STGView, 2D-Lankartenansicht 2

Wie auch im Höhenprofil, läßt sich das Diagramm mit der linken Maustaste verschieben und mit der rechten zoomen. Zum Anpassen an den Geschwindigkeitsbereich sind die + und – Tasten gut. Damit läßt sich der Abstand zwischen den Farbschichten zwischen 1 $^{km}/_{h}$ bis 15 $^{km}/_{h}$ einstellen.

Wenn man nun diese Auswertung auf eine Straßenkarte legt, so erkennt man gut die Übereinstimmung. Hier wurde eine – von Map24 (www.map24.de) generierte – Landkarte als Hintergrund verwendet:

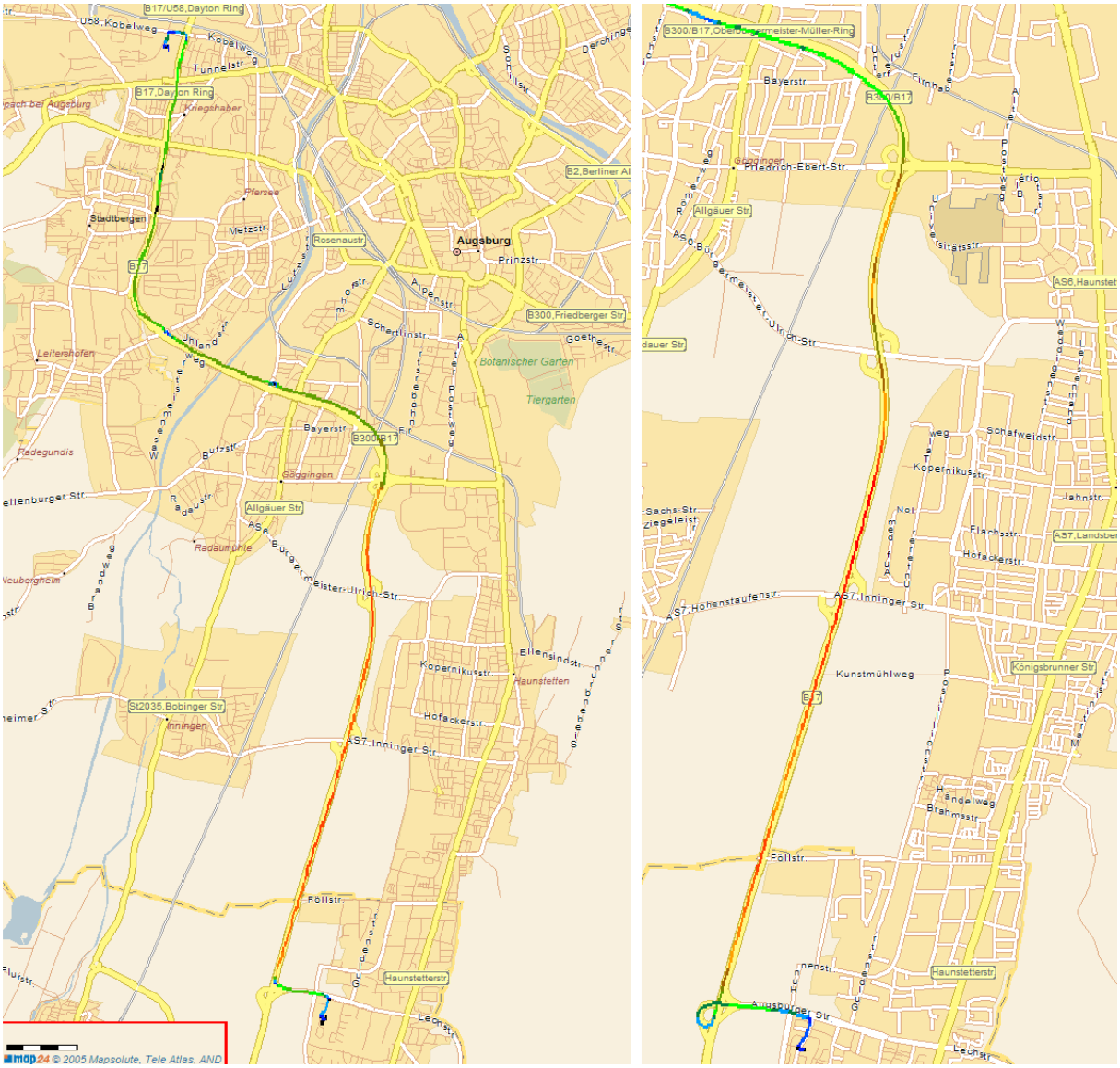


Abbildung 184 – STGView, 2D-Lankartenansicht 1, überlagert

Oder von der zweiten Fahrt:

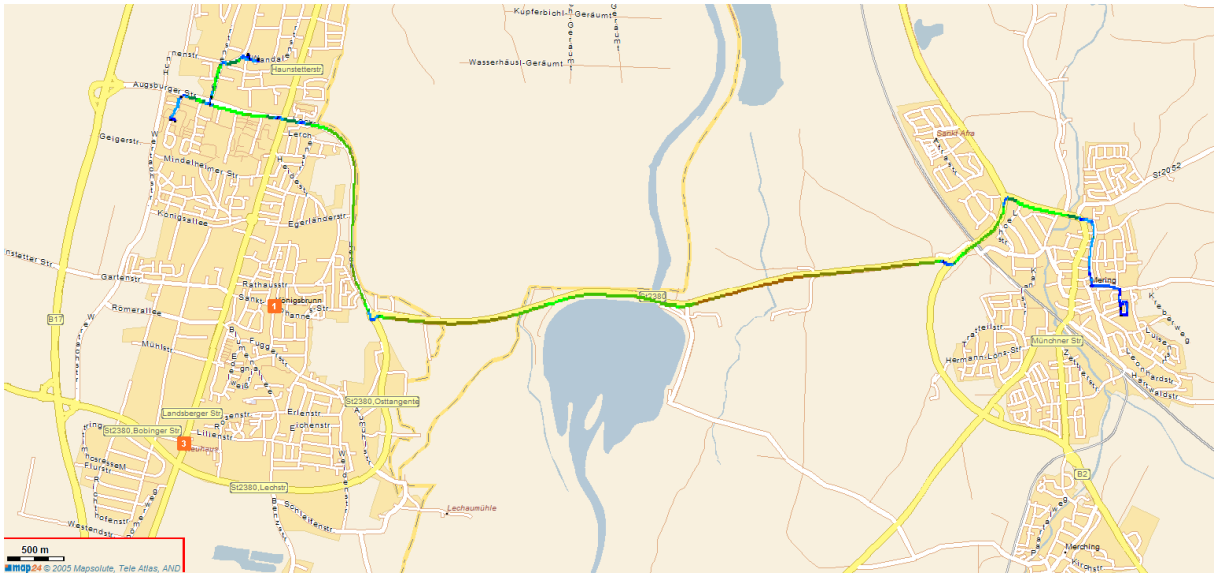


Abbildung 185 – STGView, 2D-Lankartenansicht 2, überlagert

6. ZUSAMMENFASSUNG

6.1. MÖGLICHE VERBESSERUNGEN

In diesem Kapitel sollen Verbesserungs- und Erweiterungsvorschläge für den Hardware- und den AVR-Software-Teil dieser Arbeit vorgetragen werden. Dabei wird keinerlei Rücksicht auf Machbarkeit beziehungsweise eventuell auftretende Schwierigkeiten genommen, es sind also reine Ideen.

6.1.1. HARDWARE

- **PWR-Modul:** Die Dioden am Eingang weglassen, um die minimale Eingangsspannung weiter zu senken. Somit könnte bei Batteriebetrieb die Spannung auf etwa 5.5V sinken, was eine längere Betriebszeit mit Batterien zur Folge hätte.

- **LCD-Modul:** Kontrasteinstellung vom Controller aus: Bisher muß der Kontrast extern über einen Spindelpoti eingestellt werden. Man könnte diese Einstellung entweder mit geeigneten Ausgleichsschaltungen komplett automatisieren oder zumindest digital aus dem Optionsmenü aus einstellbar machen. Dazu bräuchte man einen DA-Wandler zum Erzeugen der ausgewählten Spannung.

- **GPS-Modul:** Bisher kann man nur den Eingangskanal zwischen den drei Kanälen auswählen, die Ausgangssignale gehen aber an alle Ausgänge gleichzeitig. Man dürfte die Daten auch nur an dem zur Zeit selektierten Kanal ausgeben.

- **MMC-Modul:** Bisher basiert der Level-Converter-Teil der Schaltung auf Spannungsteilern, beziehungsweise auf der „Hoffung“, daß der AVR-Controller die 3.6V als HIGH einliest. Bei professionellem Einsatz müßte man auf jeden Fall einen richtigen Level-Converter verwenden, um die Pegel korrekt umzusetzen. Auch die Spannungsversorgung der MMC-Karte sollte über einen entsprechenden Spannungsregler oder einen Step-Down-Wandler geschehen.

- **IO-Modul:** Um Datenleitungen einzusparen, könnte man die Schalter- und LED-Zustände auch über ein 1- oder 2-Draht-Protokoll übertragen und müßte nicht für jede Komponente eine eigene Leitung verwenden.

6.1.2. SOFTWARE

- **LCD:** Bisher wird die Funktionalität des Displays nicht komplett ausgenutzt und man kann nur Daten zum Display senden. Dieses unterstützt aber auch das Einlesen der Daten aus dem Displayspeicher. Man könnte dann auch Scrollen ohne Hardwareunterstützung ermöglichen oder unbenutzte Teile vom doch recht großen RAM des Displaycontrollers für andere Zwecke verwenden.

- **LCD:** Auch wird bis jetzt nur der Textmodus unterstützt. Das Display kann aber auch in einem graphischen Modus betrieben werden, mit dem man sehr viel bessere Darstellungsmöglichkeiten hätte (vgl. Kompaß, Menü, usw…).

- **GPS:** Eine große Schwachstelle ist, daß sich die Daten aus GPS_DATA jederzeit durch einen Interrupt ändern können. Hier sollte man den neuen Datensatz beim Empfang zuerst in einen Zwischenpuffer einlesen und erst beim Aufruf der Funktion GPS_NewDataAvailable(…) in die globale Variable kopieren. Fehler treten zwar beim jetzigen Design nicht auf, man kann allerdings dem Datensatz theoretisch nicht eindeutig zuordnen, ob und welche Daten zum aktuellen oder schon zum nächsten gehören. Somit gibt es jetzt eine theoretische Ungenauigkeit von maximal einer Sekunde. In der Praxis allerdings ist der Prozessor so schnell, daß wirklich nur die jetzigen Daten verwendet werden; er ist also mit der Abspeicherung schon fertig, bevor Teile der neuen Daten eintreffen. Das kann natürlich – speziell auch bei Erweiterungen – nicht garantiert werden.

- **GPS:** Man sollte den NMEA-Parser auch mit anderen GPS-Empfängern testen. Zu dieser Arbeit stand nur ein Empfänger zur Verfügung. Es könnte sein, daß andere Empfänger teilweise kleine Änderungen in der Ausgabe haben, die noch nicht berücksichtigt sind.

- **MU:** Um den Stromverbrauch weiter zu senken, könnte man die Sleep-Modi des Controllers verwenden. Die dazu notwendigen Hardwarevoraussetzungen sind schon eingebaut, allerdings noch nicht verwendet. Beim Empfang eines Zeichens tritt ein Interrupt auf. Der Powerbutton und das Signal `IO_INT` liegen zwar schon auf den Interruptleitungen `INT0` und `INT1` vom Controller, müßten aber noch entsprechend konfiguriert werden.

- **PWR:** Bisher muß die Funktion `PWR_DoCyclicPowerManagement(…)` regelmäßig aufgerufen werden. Man könnte das eventuell durch einen Interrupt automatisieren, müßte sich dann aber andererseits auch wieder Gedanken machen, wie man die aktive Applikation davon in Kenntnis setzen könnte. Wahrscheinlich wieder über eine Abfragefunktion, die das Problem nur verlagern würde. Man könnte einen einfachen Scheduler zur Programmsteuerung einbauen und könnte dann ausschließen, daß sich Applikationen unbemerkt aufhängen oder auf das ShutDown-Signal nicht reagieren. Das würde aber eine Menge anderer Probleme und Schwierigkeiten, vor allem bei der Prozesskommunikation, bringen.

- **Storage:** Bisher existieren nur vier feste Sektoren für die Routenliste. Somit sind maximal 256 Routen anzulegen. Nachdem der Startsektor der Routenliste bereits variabel im Bootsektor gespeichert wird, könnte man die Anzahl der Sektoren auch variabel machen.

- **Storage:** Bisher können nur neue Routen angelegt, alte aber nicht gelöscht werden. Das Löschen an sich ist nicht das Problem sondern eher, daß der Speicher-Zwischenraum nicht mehr genutzt werden kann, da das Storage-Dateisystem keine Segmentierung der Routen vorsieht. Diese müssen zur Zeit streng seriell liegen.

- **Storage:** Einige Sektoren werden beim Storage-Dateisystem extrem oft neu beschrieben, untern anderen am häufigsten der `NextFreeSektor`-Sektor, der eigentlich bei jeder Änderung aktualisiert werden muß. Da gerade Flash-Speichermedien eine beschränkte Wiederbeschreibbarkeit besitzen, müßte man hier über eine andere Lösung nachdenken. Neue MMC-Karten sollen allerdings intern im Controller eine Logik besitzen, daß Schreibzugriffe auf gleiche Sektoren auf immer andere physikalische Sektoren gespeichert werden. Dieses müßte jedoch noch einmal evaluiert werden.

- **Record:** Die eigentliche Aufzeichnung könnte auch im Hintergrund geschehen, so daß man im Vordergrund andere Applikationen ausführen kann, trotzdem aber aufnimmt.

- **Record:** In gebirgigen Gegenden werden Tunnels häufiger auftreten. Bisher sieht man nur den letzten Punkt vor dem Tunnel und dann den ersten Punkt nach dem Tunnel, der durch eine einfache Linie mit der Geschwindigkeit Null verbunden ist. Eventuell könnte man die Zwischenstrecke – zumindest teilweise – automatisch berechnen lassen.

6.2. Fazit

Zuerst einmal die Erleichterung: Alle von mir am Anfang gesteckten Ziele sind erfolgreich erreicht worden. Diese sind jedoch – wie wahrscheinlich bei den Meisten im ersten größeren Projekt – viel zu hoch gesteckt worden. Am Anfang so leicht geglaubte Teile stellen sich später als riesiges Problem heraus und gefährdeten den Zeitplan. Auch tauchten immer wieder neue, unvorhergesehene Ereignisse auf, die man – trotz intensiver Planung – doch nicht bedacht hatte. Manchmal ist weniger vielleicht doch mehr.

Zudem gab es auch viel Neues zu Lernen: Angefangen von den Techniken und Möglichkeiten ein Gehäuse herzustellen, bis hin zu der kompletten Entwicklung der Hardware. Teile der Elektrotechnik, in denen man bisher immer mit der Meinung „wird schon funktionieren“ durchkam, mußte ich in der Diplomarbeit endlich fundiert hinterfragen und erlernen.

Softwareseitig muß auf solch limitierten Embedded-Systems ein ganz anderer Programmierstil verwendet werden. Im Gegensatz zu den objektorientierten Modellen auf dem PC, bei denen Effizienz und Speicherverbrauch weitgehend uninteressant sind, geht es auf kleinen Mikrocontrollern fast nur um folgende Eigenschaften: Programmgröße, Speichergröße und Performance. Hier spielen 10 Bytes mehr oder weniger eine entscheidende Rolle. Die größte Herausforderung war es, die ganze Arbeit zusätzlich möglichst flexibel, verständlich und wiederverwendbar zu gestalten.

Um einen guten Kompressionsalgorithmus zu finden, mußten viele Routen analysiert und die zur Verfügung stehenden Möglichkeiten genau überdacht werden. Es sollte ja letztendlich eine so einfache und zugleich effektive Komprimierung gefunden werden, damit diese auch mit den beschränkten Ressourcen des Mikrocontrollers funktioniert.

Die Ergebnisse übertrafen meine Einschätzungen zu Beginn der Arbeit bei weitem. Allein die Tatsache, daß auf eine – zur Zeit schon sehr günstige – 512 MB Speicherkarte unvorstellbare 10 Jahre Daueraufzeichnung passen, ermöglicht ganz neue Einsatzgebiete.

Auch die minimale Gerätegröße könnte auf ein Miniaturformat reduzierbar sein. So bräuchte ein reines Aufzeichnungsgerät keinerlei Interaktivität und käme mit nur extrem wenigen Teilen aus.

Es müßte eine externe, stabilisierte 3.6V- und eine 5V-Stromversorgung vorhanden sein; mit den 5V wird der GPS-Empfänger gespeist. Dessen CMOS-Ausgangssignale werden über einen Spannungsteiler auf 3.6V gebracht und direkt auf den UART-Eingang geschalten. Der Mikrocontroller selbst und die MMC-Karte werden mit 3.6V betrieben und erfordern keine zusätzlichen Bauteile zur Beschaltung. Man benötigt also folgende Teile: Eine MMC-Fassung, am besten Reduced-MMC, einen AVR-Controller im MLF-Gehäuseformat (*Micro Lead Frame Package*), welches nur 7x7mm groß ist und zwei Widerstände. Zusätzlich jeweils einen Mini-Stecker für den Empfänger und die Stromversorgung, wobei diese den meisten Platz beanspruchen würden.

Fazit: Ein 10-Jahres-GPS-Logger in Streichholzschachtelgröße!

7. Copyright und Lizenz

creative commons

COMMONS DEED

Namensnennung-NichtKommerziell-Weitergabe unter gleichen Bedingungen 2.0 Deutschland

Sie dürfen:

- den Inhalt vervielfältigen, verbreiten und öffentlich aufführen
- Bearbeitungen anfertigen

Zu den folgenden Bedingungen:

Namensnennung. Sie müssen den Namen des Autors/Rechtsinhabers nennen.

Keine kommerzielle Nutzung. Dieser Inhalt darf nicht für kommerzielle Zwecke verwendet werden.

Weitergabe unter gleichen Bedingungen. Wenn Sie diesen Inhalt bearbeiten oder in anderer Weise umgestalten, verändern oder als Grundlage für einen anderen Inhalt verwenden, dann dürfen Sie den neu entstandenen Inhalt nur unter Verwendung identischer Lizenzbedingungen weitergeben.

- Im Falle einer Verbreitung müssen Sie anderen die Lizenzbedingungen, unter die dieser Inhalt fällt, mitteilen.
- Jede dieser Bedingungen kann nach schriftlicher Einwilligung des Rechtsinhabers aufgehoben werden.

Die gesetzlichen Schranken des Urheberrechts bleiben hiervon unberührt.

Das Commons Deed ist eine Zusammenfassung des Lizenzvertrags in allgemeinverständlicher Sprache.

8. BEGLEIT-CD

Auf der Begleit-CD der Diplomarbeit befinden sich folgende Hauptverzeichnisse:

Verzeichnis	Beschreibung
/	Root-Verzeichnis: Diplomarbeit im PDF-Format.
/DA	Text der Diplomarbeit im Word-Format, zusätzlich in den Unterverzeichnissen alle verwendeten Abbildungen.
/Electronic	Alle benutzten Datenblätter, Dokumente und Informationen zu den elektronischen Bauteilen.
/Hardware	Schaltpläne und Gehäusezeichnungen sowohl im Quellformat, als auch in gerenderten Bildern.
/Licence	Texte, Informationen und Abbildungen sowohl zur Creative-Commons-Lizenz, als auch zur GPL-Lizenz.
/PC	Quellcode und Binaries der PC-Auswertsoftware.
/Photos	Alle im Zusammenhang mit der Diplomarbeit entstandenen Fotos (über 500).
/Source	Quelltext und Binaries der Software für das GPS-Gerät.
/Utils	Verwendete Hilfsprogramme.

Tabelle 65 – Begleit-CD

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LISTINGVERZEICHNIS

GLOSSAR

AD-Wandler	Analog-Digital-Umsetzer: Wandelt analoge Eingangsdaten in digitale Ausgangsdaten um.
ASCII	American Standard Code for Information Interchange: Standardzeichensatz.
Baud	Frühe Einheit für Datenübertragungsgeschwindigkeit bei Modems in Bit/s.
Checksumme	Prüfsumme: Algorithmus zur Gewährleistung der Datenintegrität.
CMOS-Level	Darstellung der Bits: 0 = 0V; 1 = 5V.
Connector	Physikalisches Element zum Verbinden von Signalleitungen.
DC/DC-Wandler	Elektronisches Bauteil, das aus einer bestimmten Eingangsspannung eine bestimmte Ausgangsspannung erzeugt.
Debuggen	Schrittweises Ausführen von Quellcode zum Testen und Auffinden von Fehlern.
DGPS	Differential GPS, Differential Global Positioning System: Verfahren zur Verbesserung der Genauigkeit von GPS.
DIP	Dual in-line package: Gehäuseform mit zwei parallelen Reihen von Anschlüssen.
DOP	Dilution Of Precision: Qualitätsangabe bei GPS-Empfang.
EEPROM	Electrically Erasable Programmable Read-Only Memory: Elektrisch löschbarer Nur-Lese-Speicher.
EL-Folie	Eine Folie zur Hintergrundbeleuchtung von LCD-Displays. Leuchtet beim Anlegen von Wechsel-Hochspannung.
EL-Inverter	Bauteil zum Umwandeln einer Eingangsspannung in eine Wechsel-Hochspannung zur Ansteuerung von EL-Folien.
GPS	Global Positioning System: Satellitengestütztes System zur weltweiten Positionsbestimmung.
GPS-Maus	Kompletter GPS-Empfänger in einem Gehäuse mit einfacher Ansteuerung (meist serielle Schnittstelle).
GPS-Modul	Eine Platine im Projekt als Zwischenschicht zwischen AVR-Controller und dem GPS-Empfänger.
HEX	Hexadezimale Darstellung, Darstellung im 16er-Zahlensystem.
IC	Integrated Circuit, Integrierter Schaltkreis: Zusammenfassung elektrischer Bauteile mit Beschaltung auf einem Träger.
ID	Identifikationsbezeichnung / Identifikationsnummer
IO	Input / Output: Ein- / Ausgabe
IO-Modul	Eine Platine im Projekt, auf der die LEDs zur Ausgabe und die Taster zur Eingabe befestigt sind und direkt an die Frontplatte montiert wird.
JTAG	Joint Test Action Group: Standardisiertes Verfahren zum Debuggen von Hardware.
LCD	Liquid Crystal Display: Flüssigkristallbildschirm
LCD-Modul	Eine Platine im Projekt als Zwischenschicht zwischen AVR-Controller und dem LCD-Display.
Level-Konverter	Elektronisches Bauteil zum Wandeln einer Eingangsspannung in eine Ausgangsspannung.
LoBat	Low Battery: Signal zur Anzeige eines niedrigen Batteriestands.
Low-Voltage	Bausteine mit einer niedrigeren Versorgungsspannung als üblich.
MMC	Multimedia Card: Flash Speicherkarte zur nichtflüchtigen Speicherung von

	Daten. Zugriff über das SPI-Protokoll.
MMC-Modul	Eine Platine im Projekt als Zwischenschicht zwischen AVR-Controller und der MMC-Karte.
MU	Main Unit: Auf ihr befindet sich der Controller; alle anderen Platinen werden mit ihr verbunden.
Multiplexer	Selektionsschaltnetz zur Auswahl eines Signals aus mehreren verfügbaren Eingangssignalen.
MUX	Siehe Multiplexer.
NMEA	National Marine Electronics Association: Übertragungsstandard
PDIP	Plastic DIP: Gehäuseform aus Plastik (siehe DIP).
Port	Hardware-Schnittstelle
Programmer	Hardware zum Einspielen von Software in das zu programmierende Gerät.
Pull-Down-Widerstand	Verbinden einer Datenleitung mit Masse über einen Widerstand, um einen definierten Aus-Zustand zu erreichen.
Pull-Up-Widerstand	Verbinden einer Datenleitung mit +VDD über einen Widerstand, um einen definierten Aus-Zustand zu erreichen.
PWM	Pulsweitenmodulation, Modulation des Tastverhältnisses eines Rechtecksignals bei konstanter Frequenz.
PWR-Modul	Eine Platine im Projekt, zuständig für die Stromversorgung aller anderen Teile.
RISC	Reduced Instruction Set Computing: Prozessor mit reduziertem Befehlssatz, meist einfache aber dafür sehr schnelle Befehle.
RS-232	Heute EIA-232-Schnittstelle, serielle Spannungsschnittstelle.
RS-232-Level	Darstellung der Bits: 0 = +3V … +12V; 1 = -3V … -12V.
RTC	Real Time Clock: Echtzeituhr zur Messung der physikalischen Zeit.
Schottky	Sehr schnelle Diode mit niedriger Diffusionsspannung.
Sink	(Gegenteil: Source): LOW-Ausgang: Legen einer IO-Leitung auf Masse.
SMD	Surface Mounted Device: Direkt auf der Oberfläche montiertes Bauteil.
SOT-32	Spezielle Gehäuseform mit drei Anschlüssen, meist für Transistoren mittlerer Größe.
Source	(Gegenteil: Sink): HIGH-Ausgang: Legen einer IO-Leitung auf die Versorgungsspannung.
Source	Quelltext
SPI	Serial Peripheral Interface: Synchroner serieller Datenbus mit 4 Busleitungen.
Spindeltrimmer	Spezielle Bauform eines Potentiometers mit einem Einstellungsbereich von einigen Umdrehungen.
SRAM	Static Random Access Memory: Flüchtiger, schneller Speicher mit direktem Zugriff.
STK500	Komplettes Starter-Kit und Entwicklungssystem für AVR-Mikroprozessoren.
Tasterprellung	Kurze Kontaktunterbrechungen beim Schließen von Schaltern, hervorgerufen durch das Aufeinanderprallen von Metall.
Timer	Zeitgeber: Periodisches Auslösen eines Interrupts in fest definierten Zeitabständen.
TO-92	Spezielle Gehäuseform mit drei Anschlüssen, meist für kleine Transistoren.
TWI	Two-Wire-Serial-Interface: Einfaches, aber mächtiges Kommunikations-interface mit nur zwei Busleitungen.
UART	Universal Asynchronous Receiver Transmitter: Serielle Übertragung.
Watchdog	Komponente zur Beobachtung des Systems. Führt im Fehlerfall einen Reset aus.

LITERATURVERZEICHNIS

[4539] **Datasheet:**
Motorola Semiconductor:
MC14539B
http://www.datasheetarchive.com/ (Datasheet)
http://www.national.com/

[AVRM32] **Datasheet:**
Atmel Corporation:
ATmega32(L) Complete (344 pages, revision H, updated 03/05)
http://www.atmel.com

[BC548] **Datasheet:**
Motorola Semiconductor:
BC546, B; BC547, A, B, C; BC548, A, B, C
http://www.datasheetarchive.com/ (Datasheet)
http://www.motorola.com/

[BC558] **Datasheet:**
Motorola Semiconductor:
BC556, B BC557, A, B, C BC558B
http://www.datasheetarchive.com/ (Datasheet)
http://www.motorola.com/

[BD439] **Datasheet:**
SGS-Thomson Microelectronics (ST):
BD439/BD440/BD441/BD442
http://www.datasheetarchive.com/ (Datasheet)
http://www.st.com/

[GM210] **Datasheet:**
Holux Technology Inc.:
GM-210 GPS Receiver
http://www.holux.com

[GPS] **Website:**
Wikipedia (Verschiedene Autoren):
Global Positioning System
http://de.wikipedia.org/wiki/Global_Positioning_System

[IA32OPT] **Datasheet:**
Intel Corporation:
IA-32 Intel® Architecture Optimization
http://developer.intel.com/

[LCD] **Datasheet:**
Hitachi Semiconductor:
LMG638X Family
http://www.hitachi.com/

[LCDCTL] **Datasheet:**
Hitachi Semiconductor:
HD61830/HD61830B LCDC (LCD Timing Controller)
http://www.hitachi.com/

[LM2575] **Datasheet:**
National Semiconductors:
LM1575/LM2575/LM2575HV
http://www.datasheetarchive.com/

[MAX232] **Datasheet:**
Texas Instruments Incorporated:
MAX232
http://www.datasheetarchive.com/ (Datasheet)
http://www.ti.com/

[MMC] **Datasheet:**
Hitachi Semiconductor:
HB28E016MM2/… (MultiMediaCard)
http://www.hitachi.com/

[NMEA] **Website:**
Peter Bennett:
The NMEA FAQ, Version 6.1 Sept. 15, 1997
ftp://sundae.triumf.ca/pub/peter/index.html
bennett@triumf.ca (E-Mail)

[STK500] **Datasheet:**
Atmel Corporation:
AVR STK500 User Guide
http://www.atmel.com

[WINAVR] **Website:**
Eric B. Weddington: (arcanum@users.sf.net)
WinAVR User Manual - 20050214
http://sourceforge.net/projects/winavr
http://winavr.sourceforge.net/

INDEX