University of Applied Sciences

GEORG-SIMON-OHM FACHHOCHSCHULE NÜRNBERG

FACHBEREICH elektrotechnik feinwerktechnik informationstechnik

# Praktikum Mikrocomputertechnik

## Realisierung eines Ortungssystems mit GPS

im WS 2006/07

## Gruppenmitglieder:

| | |
|---|---|
| Geißendörfer, Peter | 742902 |
| Leifer, Jens | 744202 |
| Uhl, Michael | 712560 |

## Betreuer:

Prof. Dr. Wackersreuther

# Inhaltsverzeichnis

# 1 Rahmenbedingungen

Die Studienpläne für Mikroelektronik und Nachrichtentechnik an der Georg-Simon-Ohm-Fachhochschule Nürnberg sehen für das 7. Semester ein Praktikum im Fach Mikrocomputertechnik vor. Ziel dieses Praktikums ist es, die in der Vorlesung Mikrocomputertechnik erlernten Fähigkeiten praktisch umzusetzen. Die Wahl der Aufgabenstellung bleibt dabei den Studenten überlassen. Unsere Gruppe entschied sich ein Ortungssystem mit GPS zu realisieren. Die Bearbeitung dieser Aufgabe geschah projektorientiert.

## 1.1 Beschreibung des Projekts

Ziel des Projektes ist es mit dem auf dem Lern- und Experimentierboard befindlichen RISC AVR-Mikrocontroller (ATmega8), einem 2-zeiligen Display und einem GPS-Empfänger ein funktionierendes Ortungssystem zu realisieren. Die Programmierung des Mikrocontrollers erfolgt dabei mit der Programmiersprache C.

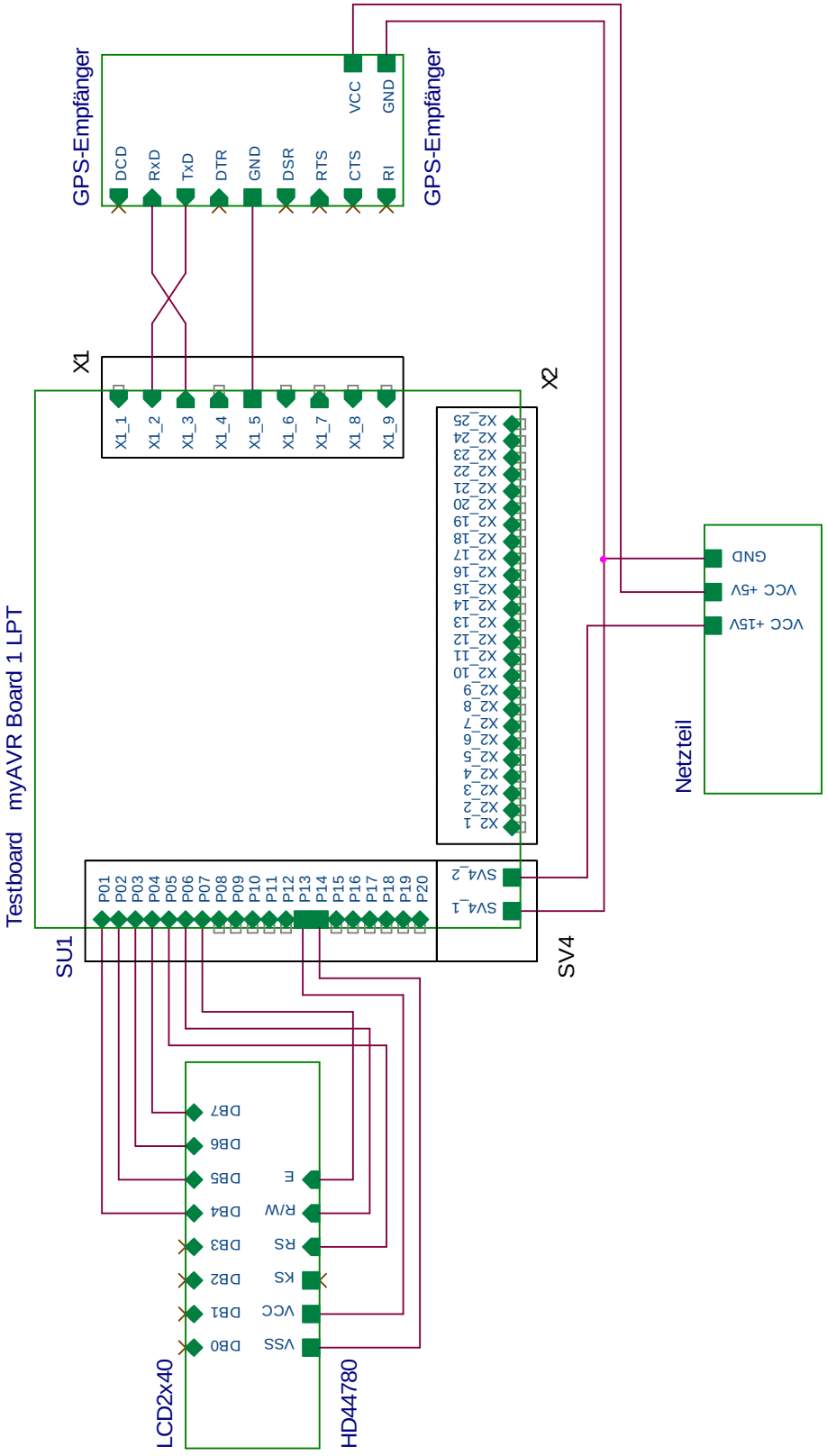
Das Projekt gliedert sich in verschiedene Phasen. In der Phase 1 stand die Informationsrecherche im Vordergrund (welche Komponenten werden benötigt, wie können die Komponenten angesprochen werden, wie kann die Programmierung des Mikrocontrollers erfolgen, …). In den Phasen 2 und 3 beschäftigten wir uns mit den einzelnen Komponenten und deren Zusammenspiel. Dabei wurde zunächst eine einfache Textausgabe auf dem Display erzeugt (Phase 2). Anschließend wurde der GPS-Empfänger in Betrieb genommen und damit die GPS-Zeit auf dem Display ausgegeben (Phase 3). Die eigentliche Realisierung des Ortungssystems fand in Phase 4 statt. Anschließend wurde noch ein Auswahlmenü verwirklicht, dass es erlaubt verschiedene Informationen des GPS-Signals darzustellen.

In Abbildung 1 sind die verwendeten Komponenten und ihre Verschaltung miteinander zu erkennen.

Das Display wird dabei an die 20-polige Sockelleiste SU1 angeschlossen. Von den 14 zur Verfügung stehenden Pins am Displaycontroller werden für die Erfüllung der Funktion allerdings nur 9 Pins benötigt.

An die 9-polige Buchse X1 wird der GPS-Empfänger angeschlossen. Das Anschlussschema entspricht hierbei dem der seriellen Schnittstelle RS232.

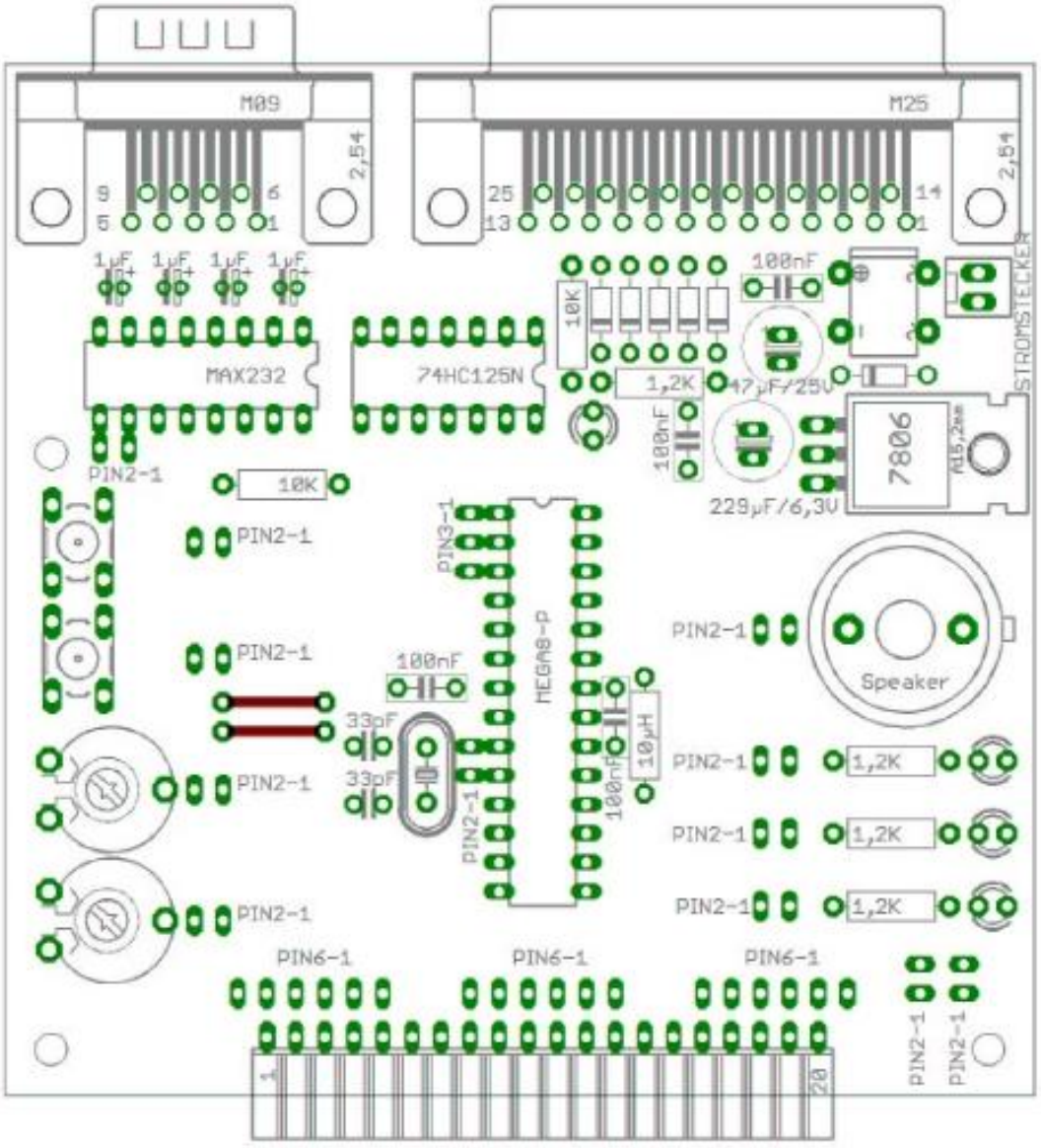
Die Spannungsversorgung erhält sowohl das myAVR Board 1 LPT als auch der GPS-Empfänger über ein externes Netzteil. Das myAVR Board 1 LPT wiederum speist das Display.



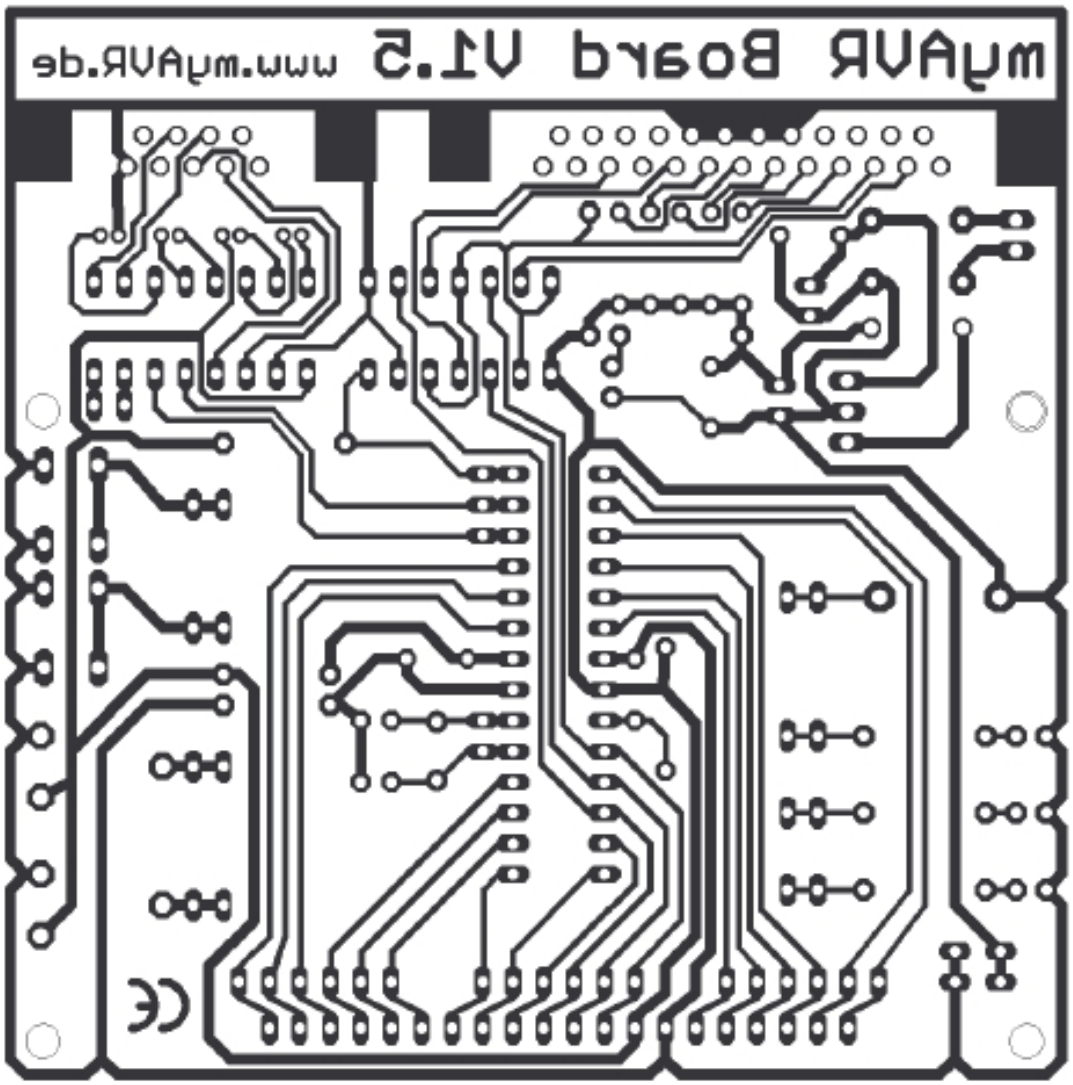

- Abbildung 1: Übersicht der verwendeten Komponenten-

## 1.2 Das Board myAVR Board 1 LPT

Das Lern- und Experimentierboard myAVR Board 1 LPT verfügt über einen RISC AVR-Mikrocontroller (ATmega8) der Firma ATMEL. Auf dem Board sind ein SP12 kompatibler LPT-Programmer und ein RS232 (COM/V24) Port integriert. Des Weiteren befinden sich bereits einige typische Ein- und Ausgabegeräte wie zum Beispiel Potentiometer, Schalter, Frequenzwandler und LEDs auf dem Board. Die für das Board vorgesehenen Controller gehört zur Reihe Mega-AVRs (ATmega8/48/88/168) und verfügen über alle wesentlichen Baugruppen.
Das Board wurde als Bausatz von der Firma Laser & Co. Solutions GmbH[1] bezogen und im Rahmen des Praktikums nach Abbildung 2 bestückt und verlötet.

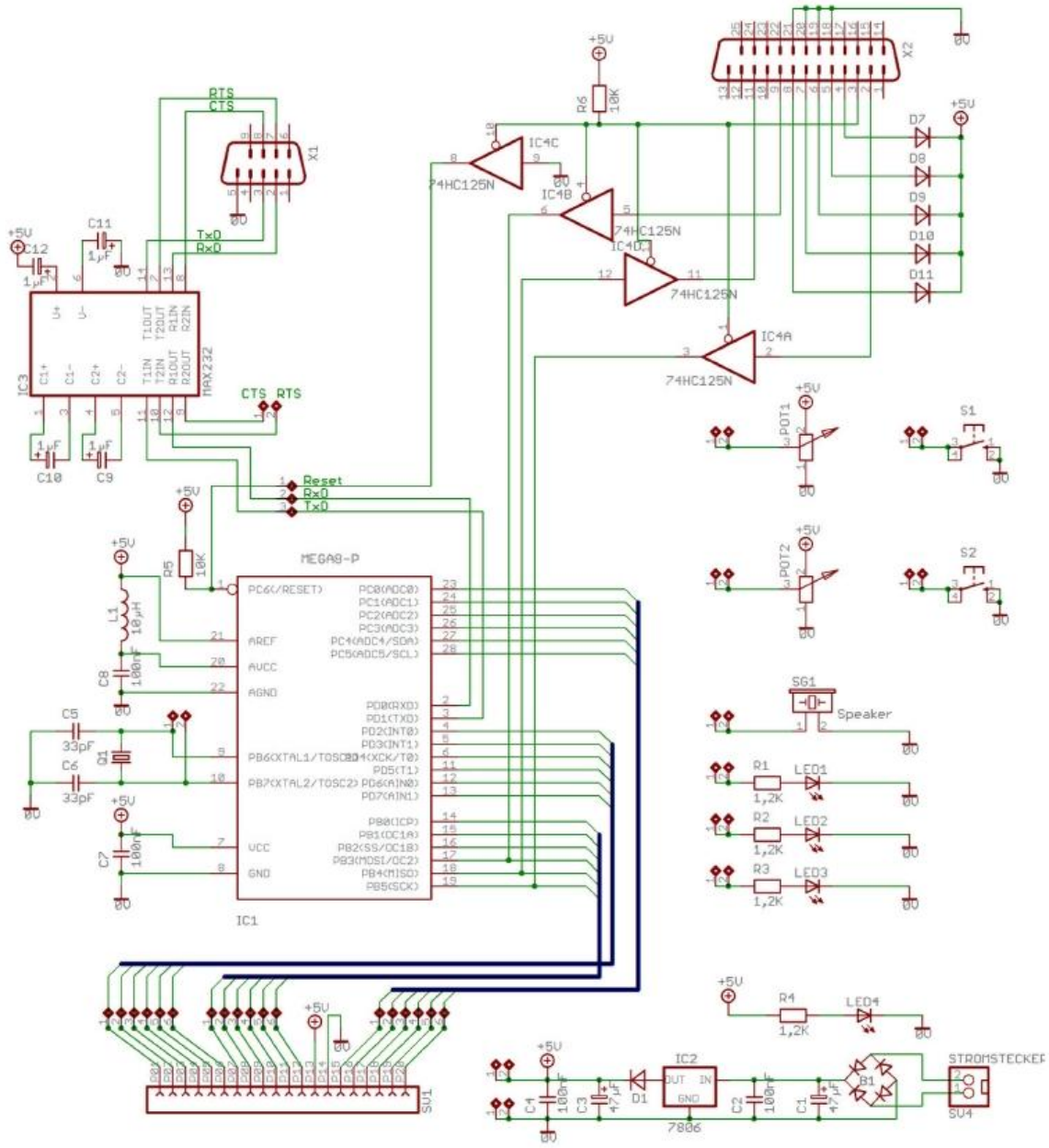



- Abbildung 2: Bestückungsplan myAVR Board 1 LPT -




- Abbildung 3: Layout myAVR Board 1 LPT -

Der Schaltplan (Abbildung 4) zeigt sowohl die interne Verschaltung auf dem Board, als auch die Schnittstellen des ATmega8L, die auf verschiedenen Schnittstellen nach Außen geführt werden. Die Schnittstelle X1 entspricht dabei der 9-poligen RS232-Schnittstelle (COM), die Schnittstelle X2 den 25-poligen ISP-Port (LPT), der unter anderem zum Programmieren des ATmega8L verwendet wird. Zur freien Verfügung (nicht Standardisiert) steht die Erweiterungsbuchse SU1. Auf dieser sind die Ports D, B und C des ATmega8L ausgeführt.

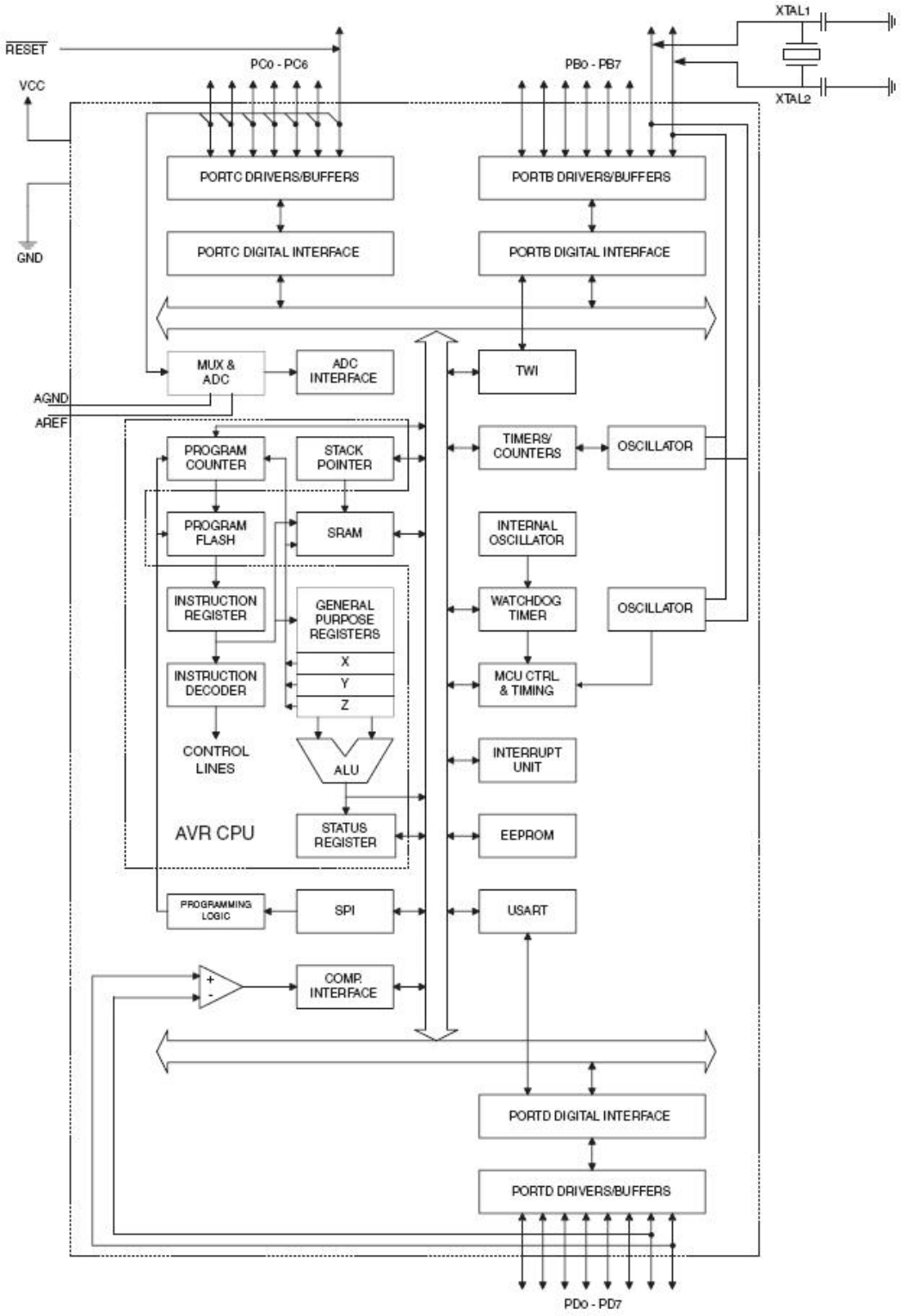



- Abbildung 4: Schaltplan myAVR Board 1 LPT -

### 1.2.1 Der Mikrocontroller ATmega8L

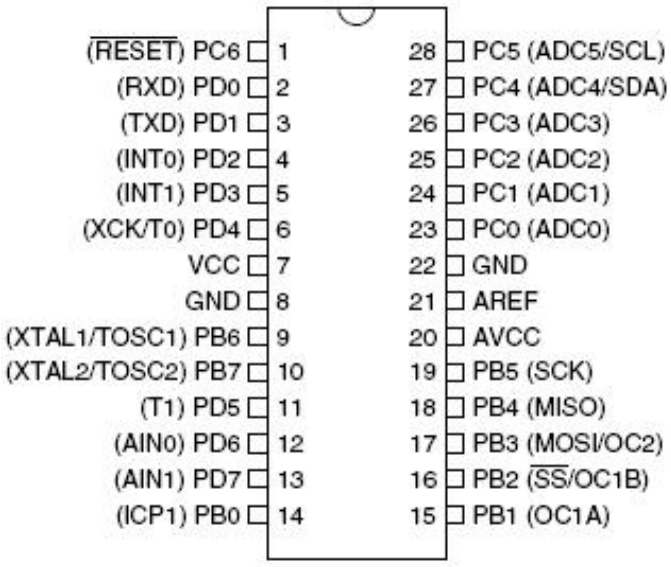
Kernstück des Boards ist der RISC AVR-Mikrocontroller ATmega8L 8PI. Hierbei handelt es sich um einen 8-bit Mikrokontroller mit max. 8MHz Taktfrequenz, 32 x 8-Bit breite Register und einem integrierten 2-Takt Multiplizierer. Als nichtflüchtiger Programmspeicher stehen 8 kByte Flashspeicher und ein 512 Byte großes EEPROM

zur Verfügung. Als Datenspeicher dient1 kByte SRAM. Den prinzipiellen Aufbau des Mikrocontrollers verdeutlicht das Blockdiagramm in Abbildung 5.




- Abbildung 5: Blockdiagramm des ATmega8L -

Die im Blockdiagramm enthaltenen Ein- und Ausgänge sind durch die Pins am IC-Baustein nach Außen geführt:




- Abbildung 6: Pin-Out des ATmega8L -

Die einzelnen Pins lassen sich anhand Ihrer Funktion beschreiben:

| Pin | Beschreibung |
|---|---|
| VCC (7) | Versorgungsspannung (digital) |
| GND (22) | Bezugspotential |
| Port B (PB7 … PB0) | 8-bit bidirektionaler I/O-Port |
| Port C (PC5 … PC0) | 7-bit bidirektionaler I/O-Port |
| PC6 | I/O-Pin (falls RSTDISBL programmiert), Reset (falls RSTDISBL nicht programmiert) |
| Port D (PD7 … PD0) | 8-bit bidirektionaler I/O-Port |
| AVCC (20) | Versorgungsspannung für AD-Wandler |
| AREF (21) | Analoge Referenzspannung für den AD-Wandler |

- Tabelle 1: Pinbelegung des ATmega8L -

Neben der eigentlichen Funktion als I/O-Ports für die Register des ATmega8L können die Ports B, C und D noch alternativ für allgemeine I/O-Aufgaben eingesetzt werden (entsprechende Programmierung vorausgesetzt):

| Port Pin | Alternative Funktion |
|---|---|
| PB7 | XTAL2 (Chip Clock Oscillator pin 2)<br>TOSC2 (Timer Oscillator pin 2) |
| PB6 | XTAL1 (Chip Clock Oscillator pin 1 or External clock input)<br>TOSC1 (Timer Oscillator pin 1) |
| PB5 | SCK (SPI Bus Master clock Input) |
| PB4 | MISO (SPI Bus Master Input/Slave Output) |
| PB3 | MOSI (SPI Bus Master Output/Slave Input)<br>OC2 (Timer/Counter2 Output Compare Match Output) |
| PB2 | SS (SPI Bus Master Slave select)<br>OC1B (Timer/Counter1 Output Compare Match B Output) |
| PB1 | OC1A (Timer/Counter1 Output Compare Match A Output) |
| PB0 | ICP1 (Timer/Counter1 Input Capture Pin) |

- Tabelle 2: Alternative Pinbelegung Port B -

| Port Pin | Alternative Funktion |
|---|---|
| PC6 | RESET (Reset pin) |
| PC5 | ADC5 (ADC Input Channel 5)<br>SCL (Two-wire Serial Bus Clock Line) |
| PC4 | ADC4 (ADC Input Channel 4)<br>SDA (Two-wire Serial Bus Data Input/Output Line) |
| PC3 | ADC3 (ADC Input Channel 3) |
| PC2 | ADC2 (ADC Input Channel 2) |
| PC1 | ADC1 (ADC Input Channel 1) |
| PC0 | ADC0 (ADC Input Channel 0) |

- Tabelle 3: Alternative Pinbelegung Port C -

| Port Pin | Alternative Funktion |
|---|---|
| PD7 | AIN1 (Analog Comparator Negative Input) |
| PD6 | AIN0 (Analog Comparator Positive Input) |
| PD5 | T1 (Timer/Counter 1 External Counter Input) |
| PD4 | XCK (USART External Clock Input/Output)<br>T0 (Timer/Counter 0 External Counter Input) |
| PD3 | INT1 (External Interrupt 1 Input) |
| PD2 | INT0 (External Interrupt 0 Input) |
| PD1 | TXD (USART Output Pin) |
| PD0 | RXD (USART Input Pin) |

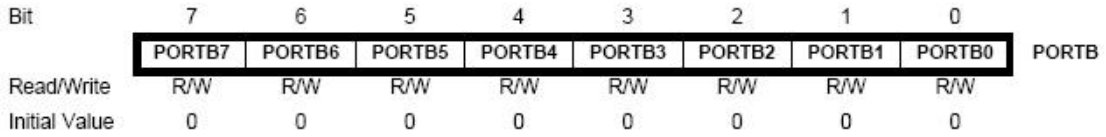
- Tabelle 3: Alternative Pinbelegung Port D -

Eine Ausführliche Beschreibung der oben genannten Pins und deren Programmierung ist im Datenblatt der Herstellerfirma Atmel Corporation zu finden[2]. Einige dieser alternativen Pinbelegungen werden im Rahmen dieses Projektes noch benötigt.

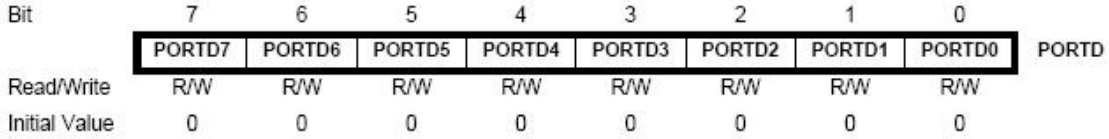
| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Page |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x3F (0x5F) | SREG | I | T | H | S | V | N | Z | C | 11 |
| 0x3E (0x5E) | SPH | – | – | – | – | – | SP10 | SP9 | SP8 | 13 |
| 0x3D (0x5D) | SPL | SP7 | SP6 | SP5 | SP4 | SP3 | SP2 | SP1 | SP0 | 13 |
| 0x3C (0x5C) | Reserved | | | | | | | | | |
| 0x3B (0x5B) | GICR | INT1 | INT0 | – | – | – | – | IVSEL | IVCE | 49, 67 |
| 0x3A (0x5A) | GIFR | INTF1 | INTF0 | – | – | – | – | – | – | 68 |
| 0x39 (0x59) | TIMSK | OCIE2 | TOIE2 | TICIE1 | OCIE1A | OCIE1B | TOIE1 | – | TOIE0 | 72, 102, 122 |
| 0x38 (0x58) | TIFR | OCF2 | TOV2 | ICF1 | OCF1A | OCF1B | TOV1 | – | TOV0 | 73, 103, 122 |
| 0x37 (0x57) | SPMCR | SPMIE | RWWSB | – | RWWSRE | BLBSET | PGWRT | PGERS | SPMEN | 213 |
| 0x36 (0x56) | TWCR | TWINT | TWEA | TWSTA | TWSTO | TWWC | TWEN | – | TWIE | 171 |
| 0x35 (0x55) | MCUCR | SE | SM2 | SM1 | SM0 | ISC11 | ISC10 | ISC01 | ISC00 | 33, 66 |
| 0x34 (0x54) | MCUCSR | – | – | – | – | WDRF | BORF | EXTRF | PORF | 41 |
| 0x33 (0x53) | TCCR0 | – | – | – | – | – | CS02 | CS01 | CS00 | 72 |
| 0x32 (0x52) | TCNT0 | Timer/Counter0 (8 Bits) | | | | | | | | 72 |
| 0x31 (0x51) | OSCCAL | Oscillator Calibration Register | | | | | | | | 31 |
| 0x30 (0x50) | SFIOR | – | – | – | – | ACME | PUD | PSR2 | PSR10 | 58, 75, 123, 193 |
| 0x2F (0x4F) | TCCR1A | COM1A1 | COM1A0 | COM1B1 | COM1B0 | FOC1A | FOC1B | WGM11 | WGM10 | 97 |
| 0x2E (0x4E) | TCCR1B | ICNC1 | ICES1 | – | WGM13 | WGM12 | CS12 | CS11 | CS10 | 100 |
| 0x2D (0x4D) | TCNT1H | Timer/Counter1 – Counter Register High byte | | | | | | | | 101 |
| 0x2C (0x4C) | TCNT1L | Timer/Counter1 – Counter Register Low byte | | | | | | | | 101 |
| 0x2B (0x4B) | OCR1AH | Timer/Counter1 – Output Compare Register A High byte | | | | | | | | 101 |
| 0x2A (0x4A) | OCR1AL | Timer/Counter1 – Output Compare Register A Low byte | | | | | | | | 101 |
| 0x29 (0x49) | OCR1BH | Timer/Counter1 – Output Compare Register B High byte | | | | | | | | 101 |
| 0x28 (0x48) | OCR1BL | Timer/Counter1 – Output Compare Register B Low byte | | | | | | | | 101 |
| 0x27 (0x47) | ICR1H | Timer/Counter1 – Input Capture Register High byte | | | | | | | | 102 |
| 0x26 (0x46) | ICR1L | Timer/Counter1 – Input Capture Register Low byte | | | | | | | | 102 |
| 0x25 (0x45) | TCCR2 | FOC2 | WGM20 | COM21 | COM20 | WGM21 | CS22 | CS21 | CS20 | 117 |
| 0x24 (0x44) | TCNT2 | Timer/Counter2 (8 Bits) | | | | | | | | 119 |
| 0x23 (0x43) | OCR2 | Timer/Counter2 Output Compare Register | | | | | | | | 119 |
| 0x22 (0x42) | ASSR | – | – | – | – | AS2 | TCN2UB | OCR2UB | TCR2UB | 119 |
| 0x21 (0x41) | WDTCR | – | – | – | WDCE | WDE | WDP2 | WDP1 | WDP0 | 43 |
| 0x20[1] (0x40)[1] | UBRRH | URSEL | – | – | – | UBRR[11:8] | | | | 158 |
| | UCSRC | URSEL | UMSEL | UPM1 | UPM0 | USBS | UCSZ1 | UCSZ0 | UCPOL | 156 |
| 0x1F (0x3F) | EEARH | – | – | – | – | – | – | – | EEAR8 | 20 |
| 0x1E (0x3E) | EEARL | EEAR7 | EEAR6 | EEAR5 | EEAR4 | EEAR3 | EEAR2 | EEAR1 | EEAR0 | 20 |
| 0x1D (0x3D) | EEDR | EEPROM Data Register | | | | | | | | 20 |
| 0x1C (0x3C) | EECR | – | – | – | – | EERIE | EEMWE | EEWE | EERE | 20 |
| 0x1B (0x3B) | Reserved | | | | | | | | | |
| 0x1A (0x3A) | Reserved | | | | | | | | | |
| 0x19 (0x39) | Reserved | | | | | | | | | |
| 0x18 (0x38) | PORTB | PORTB7 | PORTB6 | PORTB5 | PORTB4 | PORTB3 | PORTB2 | PORTB1 | PORTB0 | 65 |
| 0x17 (0x37) | DDRB | DDB7 | DDB6 | DDB5 | DDB4 | DDB3 | DDB2 | DDB1 | DDB0 | 65 |
| 0x16 (0x36) | PINB | PINB7 | PINB6 | PINB5 | PINB4 | PINB3 | PINB2 | PINB1 | PINB0 | 65 |
| 0x15 (0x35) | PORTC | – | PORTC6 | PORTC5 | PORTC4 | PORTC3 | PORTC2 | PORTC1 | PORTC0 | 65 |
| 0x14 (0x34) | DDRC | – | DDC6 | DDC5 | DDC4 | DDC3 | DDC2 | DDC1 | DDC0 | 65 |
| 0x13 (0x33) | PINC | – | PINC6 | PINC5 | PINC4 | PINC3 | PINC2 | PINC1 | PINC0 | 65 |
| 0x12 (0x32) | PORTD | PORTD7 | PORTD6 | PORTD5 | PORTD4 | PORTD3 | PORTD2 | PORTD1 | PORTD0 | 65 |
| 0x11 (0x31) | DDRD | DDD7 | DDD6 | DDD5 | DDD4 | DDD3 | DDD2 | DDD1 | DDD0 | 65 |
| 0x10 (0x30) | PIND | PIND7 | PIND6 | PIND5 | PIND4 | PIND3 | PIND2 | PIND1 | PIND0 | 65 |
| 0x0F (0x2F) | SPDR | SPI Data Register | | | | | | | | 131 |
| 0x0E (0x2E) | SPSR | SPIF | WCOL | – | – | – | – | – | SPI2X | 131 |
| 0x0D (0x2D) | SPCR | SPIE | SPE | DORD | MSTR | CPOL | CPHA | SPR1 | SPR0 | 129 |
| 0x0C (0x2C) | UDR | USART I/O Data Register | | | | | | | | 153 |
| 0x0B (0x2B) | UCSRA | RXC | TXC | UDRE | FE | DOR | PE | U2X | MPCM | 154 |
| 0x0A (0x2A) | UCSRB | RXCIE | TXCIE | UDRIE | RXEN | TXEN | UCSZ2 | RXB8 | TXB8 | 155 |
| 0x09 (0x29) | UBRRL | USART Baud Rate Register Low byte | | | | | | | | 158 |
| 0x08 (0x28) | ACSR | ACD | ACBG | ACO | ACI | ACIE | ACIC | ACIS1 | ACIS0 | 194 |
| 0x07 (0x27) | ADMUX | REFS1 | REFS0 | ADLAR | – | MUX3 | MUX2 | MUX1 | MUX0 | 205 |
| 0x06 (0x26) | ADCSRA | ADEN | ADSC | ADFR | ADIF | ADIE | ADPS2 | ADPS1 | ADPS0 | 207 |
| 0x05 (0x25) | ADCH | ADC Data Register High byte | | | | | | | | 208 |
| 0x04 (0x24) | ADCL | ADC Data Register Low byte | | | | | | | | 208 |
| 0x03 (0x23) | TWDR | Two-wire Serial Interface Data Register | | | | | | | | 173 |
| 0x02 (0x22) | TWAR | TWA6 | TWA5 | TWA4 | TWA3 | TWA2 | TWA1 | TWA0 | TWGCE | 174 |
| 0x01 (0x21) | TWSR | TWS7 | TWS6 | TWS5 | TWS4 | TWS3 | – | TWPS1 | TWPS0 | 173 |
| 0x00 (0x20) | TWBR | Two-wire Serial Interface Bit Rate Register | | | | | | | | 171 |

- Tabelle 4: Adressen der Register -

In Tabelle 4 sind sämtliche verfügbare Register des ATmega 8L mit der zugehörigen Startadresse vermerkt. Im Rahmen dieses Projektes sind die Register Port B (0x18 (Adresse im Speicherbereich: 0x38)) und Port D (0x12 (0x32)), das USART I/O Daten Register – UDR (0x0C (0x2C)) und die USART Kontroll- and Status Register – UCSR(A, B, C) (0x0B (0x2B), 0x0A (0x2A), (0x40)) von besonderem Interesse.



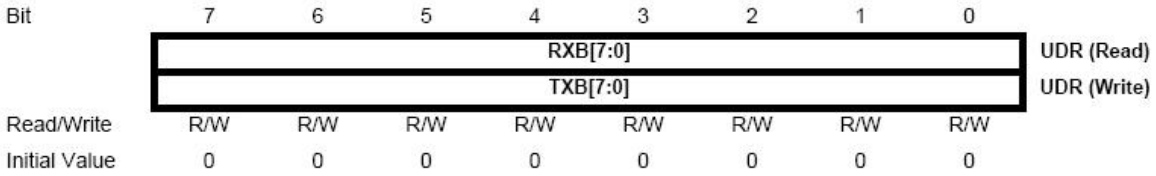

- Abbildung 7: Port B Datenregister 0x18 -



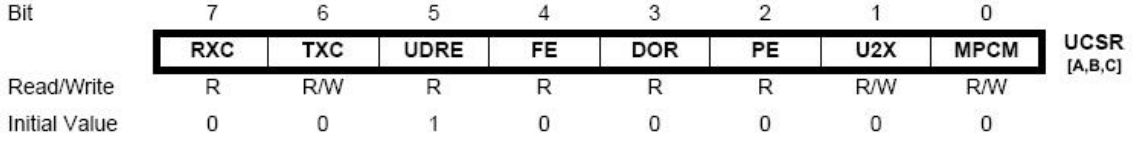

- Abbildung 8: Port D Datenregister 0x12 -

Die beiden Datenregister Port B und Port D werden im weiteren Verlauf als Schnittstelle zum LCD-Display verwendet.

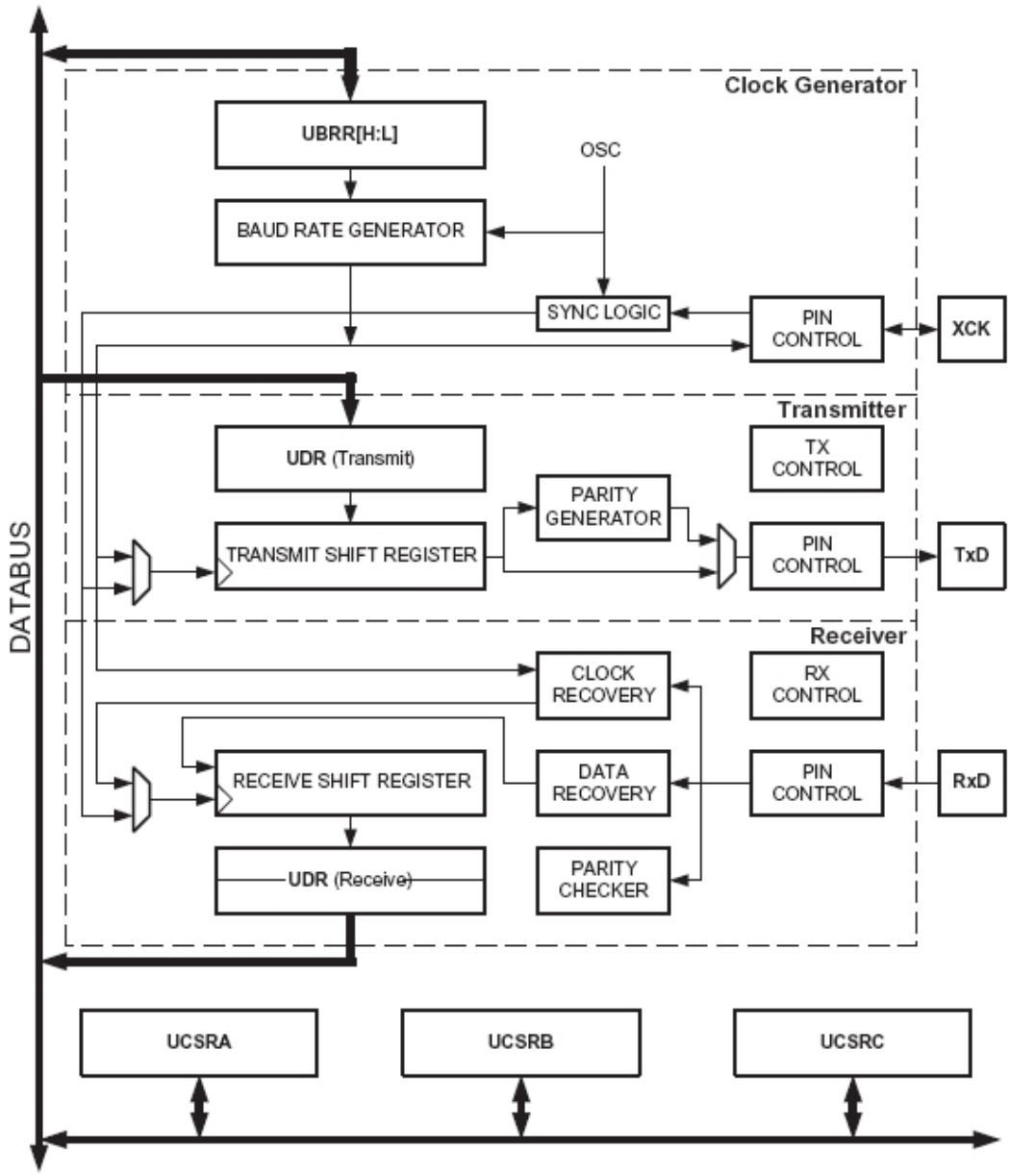
Der USART (Universal Synchronous Asynchronous Receiver Transmitter) dient als Sender und Empfänger, der die Datenumsetzung zwischen dem parallelen Bus des ATmega8 und der seriellen RS232-Schnittstelle zum GPS-Empfänger vornimmt.




- Abbildung 9: USART USART I/O Daten Register 0x0C -




- Abbildung 10: USART Kontroll- and Status Register -




- Abbildung 11: USART Blockdiagramm -

Innerhalb des USART wird der Takt für den Sender und Empfänger erzeugt. Hierzu steht ein eigener Taktgenerator zur Verfügung (siehe Abbildung 11). Der erzeugte Takt ist wichtig für die Übertragungsgeschwindigkeit („BAUD-Rate“) auf der seriellen Schnittstelle. Diese errechnet sich aus $BAUD = \frac{f_{OSC}}{16(UBRR+1)}$. $f_{OSC}$ und entspricht dabei der Taktfrequenz. $UBRR$ entspricht dem Inhalt des USART Baud Rate Registers.

In unserem Fall ist jedoch die Übertragungsgeschwindigkeit bereits vom GPS-Empfänger bzw. dessen RS232-Schnittstelle mit 4800 bit/s vorgegeben. Daher muss $UBRR = \frac{f_{OSC}}{16 \cdot BAUD} - 1$ berechnet werden. Bei der Programmierung bietet es sich dabei an, die Berechnung von $UBRR$ mit einem Makro zu gestalten:

```
#define UART_UBRR_CALC(BAUD_,FREQ_) ((FREQ_)/((BAUD_)*16L)-1)

#define UART_BAUD_RATE 4800
```

- Code 1: Makro zur UBRR-Berechnung -

## 1.3 Der GPS-Empfänger

Als GPS-Empfänger setzen wir den „Multi-Mode foldable GPS Receiver“ HI-303 $S^3$ von der Firma Haicom GPS ein. Hierbei handelt es sich um eine GPS-Maus mit interner Antenne. Als Schnittstellen stehen Compact Flash Ver 1.4 und die von uns verwendete RS232-Schnittstelle zur Verfügung. Von der RS232-Schnittstelle werden allerdings nur drei Leitungen benötigt (RxD, TxD und GND).
Die GPS-Maus verwendet den properitären Chipsatz Sirf 2e/LP. Von diesem ist jedoch keine Dokumentation vorhanden, so dass wir ihn als „Black Box“ betrachten müssen, der Datensätze nach dem Protokoll NMEA-0183 versendet.

### 1.3.1 Das Protokoll NMEA-0183

Die NMEA (National Marine Electronics Association, Nationale Vereinigung für Marineelektronik) engagiert sich für die Ausbildung und den Fortschritt der Marine-Elektronikindustrie und dem Markt, den diese bedient. Die NMEA hat unter anderem den Standard NMEA-1083 definiert um einen Datenaustausch zwischen verschiedenen Geräten aus der Marineelektronik zu ermöglichen. Die NMEA-0180,0182 und 0183 Standards (die ersten beiden sind nicht mehr von Bedeutung) sehen pro Netz ein Sendegerät und diverse Empfangsgeräte vor. Der Sender soll Daten nach dem RS-232-Standard ausgeben. Die Datenrate beträgt dabei 4800 bd (ca. 600 Bytes/Sekunde). In diesem Standard ist kein Stecker oder ähnliches definiert - die Anschlussart bleibt also dem Hersteller überlassen, weswegen viele Geräte unterschiedliche Stecker benötigen.
Übertragen werden die Daten im ASCII-Format. Dabei sind alle druckbaren Zeichen sowie Carriage-Return (CR, Waagenrücklauf) und Line-Feed (LF, Neue Zeile) erlaubt. Die Daten werden in der Form von Sätzen übertragen. Jeder dieser Sätze beginnt mir dem Zeichen „$“, einer zwei Zeichen langen Senderkennung, einer drei Zeichen langen Satzkennung und dann folgt eine Reihe von Datensätzen, die mit Kommas unterteilt werden. Schließlich wird der Satz mit einer optionalen Prüfsumme und einer CR/LF abgeschlossen. Jeder Satz kann inklusive des führenden „$“ und den beiden CR/LF bis zu 82 Zeichen enthalten. Ist ein Datenfeld in einem Satz zwar

vorgesehen aber nicht verfügbar, so wird er einfach weggelassen - das dazugehörige Komma zur Trennung der Datensätze wird aber ohne Leerzeichen beibehalten. Durch Zählen der Kommas kann ein Empfänger dann aus jeden Satz die entsprechenden Informationen richtig zuordnen. Die meist optionale Prüfsumme besteht aus einem „*“ und zwei Hexadezimalzahlen, die sich durch ein (bitweise) Exklusiv-Oder (1+1=0, 1+0=1, 0+0=0) aller Zeichen zwischen dem „$“ und dem „*“ berechnen. Bei manchen Sätzen ist die Prüfsumme notwendig.
Außerdem erlaubt der Standard den Herstellern proprietäre Satzformate. Diese fangen mit „$P“ an, gefolgt von der drei Buchstaben langen Herstellerkennung. Anschließend folgen die Daten (vgl. [4]).
Nach der NMEA-Spezifikation beginnen also sämtliche GPS-Signale mit dem Präfix „$GP“. Dies gestaltete die nachfolgende Programmierung der Interrupt Service Routine einfacher, da für die Auswahl der Datensätze erste das vierte Zeichen relevant ist.

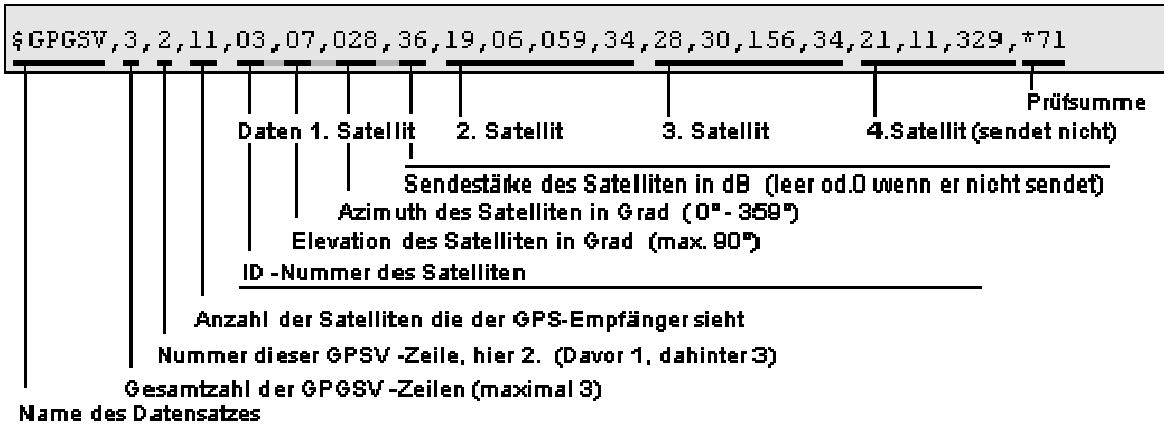
Der Datenstrom vom GPS-Empfänger lässt sich gut mit dem Hyper-Terminal von Windows XP analysieren:

```
$GPGSV,3,3,10,10,11,031,36,30,05,140,35*7B
$GPRMC,143626.943,A,4927.1555,N,01105.8498,E,0.00,,161106,,,A*7F
$GPVTG,,T,,M,0.00,N,0.0,K,A*13
$GPGGA,143627.943,4927.1555,N,01105.8498,E,1,09,0.9,390.9,M,,,,0000*03
$GPRMC,143627.943,A,4927.1555,N,01105.8498,E,0.00,,161106,,,A*7E
$GPVTG,,T,,M,0.00,N,0.0,K,A*13
$GPGGA,143628.943,4927.1570,N,01105.8525,E,1,09,0.9,371.0,M,,,,0000*0A
$GPRMC,143628.943,A,4927.1570,N,01105.8525,E,0.00,,161106,,,A*71
$GPVTG,,T,,M,0.00,N,0.0,K,A*13
$GPGGA,143629.943,4927.1576,N,01105.8536,E,1,09,0.9,360.2,M,,,,0000*0D
$GPRMC,143629.943,A,4927.1576,N,01105.8536,E,0.00,,161106,,,A*74
$GPVTG,,T,,M,0.00,N,0.0,K,A*13
$GPGGA,143630.943,4927.1574,N,01105.8513,E,1,09,0.9,343.1,M,,,,0000*02
$GPRMC,143630.943,A,4927.1574,N,01105.8513,E,0.00,,161106,,,A*79
$GPVTG,,T,,M,0.00,N,0.0,K,A*13
$GPGGA,143631.943,4927.1574,N,01105.8513,E,1,09,0.9,343.1,M,,,,0000*03
$GPGSA,A,3,07,18,31,10,16,03,21,25,06,,,,1.5,0.9,1.2*37
$GPGSV,3,1,10,07,83,330,30,21,74,134,32,16,57,295,30,06,39,086,45*79
```

- Aufzeichnung 1: Datensätze des GPS-Empfängers -

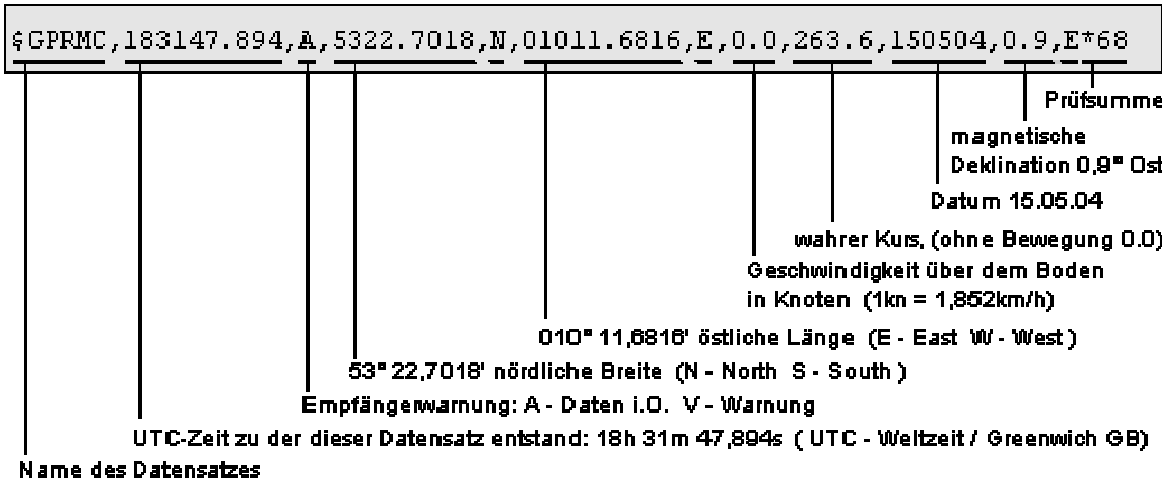
Aus Aufzeichnung 1 lassen sich gut die fünf Datensätze erkenne, die von unserem GPS-Empfänger HI-303 über die RS232-Schnittstelle gesendet werden.

Der GPGSV-Datensatz (SV, satellites in view, sichtbare Satelliten) enthält Informationen über die Satelliten, die empfangen werden. Hierbei werden neben der ID-Nummer der Satteliten auch die Position (Azimuth und Elevation) und die Signalstärke (in dB) angegeben. Da pro Satz nur die Informationen von vier Satelliten übertragen werden können (Beschränkung auf 82 Zeichen), kann es bis zu drei solche Datensätze geben.




- Abbildung 12: Datensatz GSV -

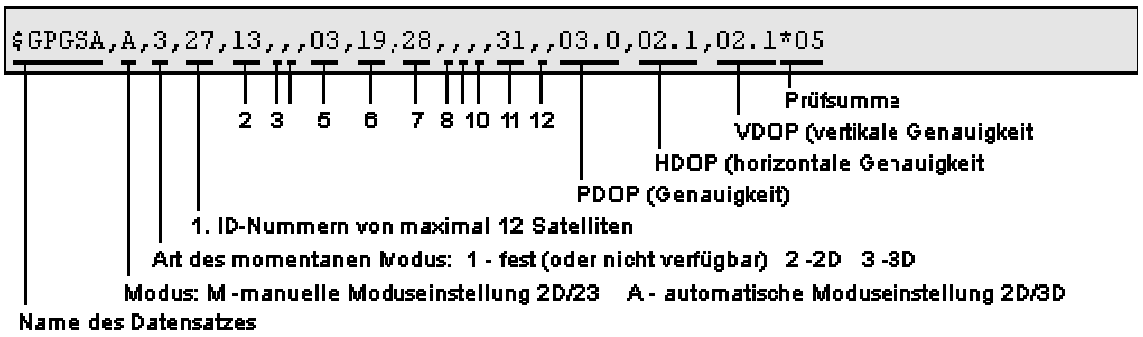
Der GPRMC-Datensatz (RMC, recommended minimum sentence C, empfohlener Minimumdatensatz) ist eine Empfehlung für das Minimum, was ein GPS-Empfänger ausgeben soll. Dieser enthält neben den Koordinaten (nördliche Breite (latitude - Nord entspricht positiven Zahlenwert) und Längengrad (longitude – Ost entsprich positiven Zahlenwerten)) und der UTC-Zeit, zu der der Datensatz gesendet wurde, auch die Geschwindigkeit des Satelliten in Knoten, den tatsächlichen Kurs (Bewegungsstarr), das Datum und die magnetische Deklination (Ortsmissweisung, die Abweichung zwischen geographischer und magnetischer Nordrichtung). Diese Daten stellen ein Minimum an Information dar, die ein Navigationsgerät benötigt um die Position und die Zielrichtung zu bestimmen, daher RMC.




- Abbildung 13: Datensatz RMC -

Der GPVTG-Datensatz (VTG, velocity track ground, Bewegungsgeschwindigkeit (des GPS-Empfängers)) enthält Informationen über die Bewegungsgeschwindigkeit und die Bewegungsrichtung. Eine graphische Darstellung dieses Datensatzes ist an dieser Stelle nicht möglich, da das Protokoll unter Laborbedingungen im statischen Zustand aufgenommen wurde.

Der GPGSA-Datensatz (SA, satellites active, aktive Satelliten) beinhaltet die PRN-Nummern (PRN, pseudorandom noise, pseudozufälliges Rauschen). Das Übertragungsverfahren der GPS-Signale entspricht CDMA (Code Division Multiple Access). Hierbei handelt es sich um Signale, die durch pseudozufälliges Rauschen aufgespreizt werden. Hierdurch wird eine geringere Störanfälligkeit gegenüber Interferenzen erreichet. Da jeder Satellit eine andere PRN-Sequenz (vgl. Goldcodes bei UMTS) verwendet, ist es möglich, dass mehrere Satelliten die gleiche Sendefrequenz benutzen und der Empfänger trotzdem die Signale unterscheiden kann.

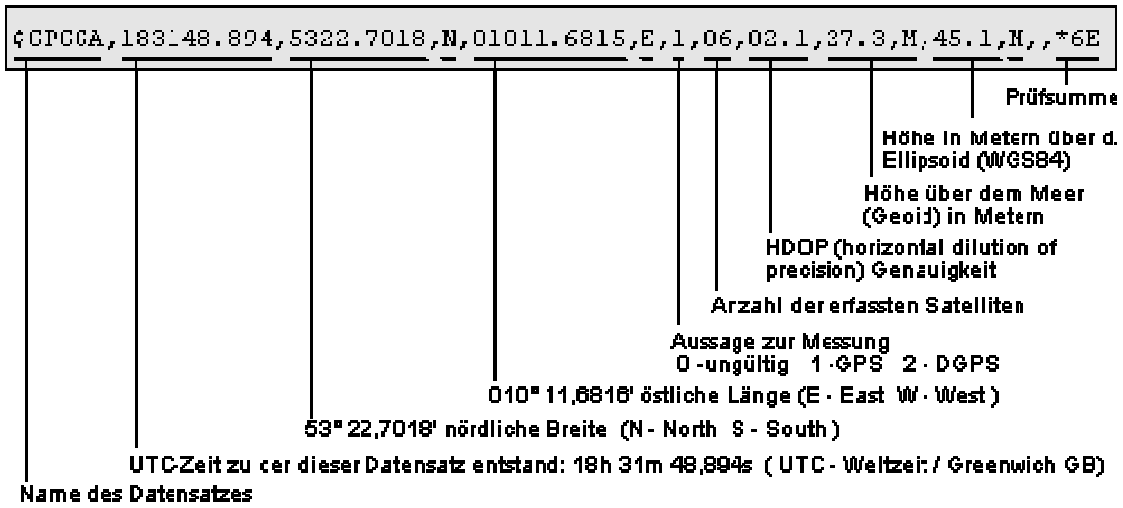



- Abbildung 14: Datensatz GSA -

Im Idealfall (unreell) werden alle 12 Satelliten mit ihren PRN-Nummern dargestellt. Anschließend wird die Genauigkeit der Messung mit den drei folgenden Werten angegeben (sog. DOP-Werte, Güte der Satellitengeometrie). PDOP bezeichnet die allgemeine Genauigkeit (in Abhängigkeit der Anzahl der empfangenen Satelliten), HDOP bezeichnet die horizontale Genauigkeit (HDOP-Werte unter 4 werden als sehr gut bewertet – Werte über 8 als schlecht. Die HDOP Werte werden desto schlechter, je höher (am Himmel) sich die empfangenen Satelliten befinden) und VDP die vertikale Genauigkeit (die Bewertung findet wie bei den HDOP-Werten statt – jedoch werden die Werte schlechter je näher sich die Satelliten am Horizont befinden).

Der GGA-Datensatz enthält die wichtigsten Daten zur GPS-Position und Genauigkeit. Daher wird dieser Datensatz maßgeblich für unser Projekt ausgewertet. Er enthält die UTC-Zeit, zu der der Datensatz entstand, die Koordinaten in nördlicher Breite und östlicher Länge (in Grad und Minuten (Minuten bis zur vierten Kommastelle)). Für die Genauigkeit bezüglicher der Messung und Navigation wird die die Anzahl der zur Messung empfangenen Satelliten und die DOP-Werte HDOP und VCOP übertragen.

Die HDOP-Werte kennzeichnen in diesem Fall Angaben über die Höhe N.N. (entspricht Geoid – Eine Bezugsfläche im Schwerefeld der Erde zur Vermessung und Beschreibung der Erdfigur. In guter Näherung wird das Geoid durch den mittleren Meeresspiegel der Weltmeere repräsentiert und ist damit in seiner Form außerhalb der Landmassen sichtbar). Die VDOP-Werte wiederum beschreiben das Ellipsoid (Ellipsoide dienen als Referenzfläche um die Lage und Höhe von Objekten auf der Erdoberfläche anzugeben).




- Abbildung 15: Datensatz GGA -

## 1.4 Das Display LCD 2x40

Bei dem eingesetzten Display handelt es sich um ein Punkt-Matrix LCD-Display mit integriertem Controller. Die Anzeige besteht aus 2 Zeilen, in denen je 40 Zeichen dargestellt werden können. Der Displaycontroller ist der HD44780.[5]
Zur Kommunikation zwischen dem Display und dem Mikrocontroller muss also der Displaycontroller näher betrachtet werden.

Dieser besteht aus:

- Zwei 8-bit breiten Register (Instruction Register zum speichern von Instruktionen wie „clear display“, „shift cursor“ und zum speichern von Adressinformationen des RAM - Data Register wird für die temporäre Speicherung der Daten während des Datenaustausches mit dem Mikrocontroller verwendet).

- Dem Busy Flag um dem Mikrocontroller mit einer logischen 1 zu signalisieren, dass der Displaycontroller einen Befehl ausführt und keine Anweisungen angenommen werden können.

- Dem Display Data RAM, in dem die mit dem Display darstellbaren 80 8-bit Zeichen codiert gespeichert sind.

- Dem Character Generator RAM, in dem acht beliebige, vom Anwender zu programmierende, Zeichen gespeichert werden können.

- Dem Character Generator ROM, in dem sämtliche Bitmuster des Zeichenvorrates gespeichert sind.

- Dem Adress Counter, der die Adressen des Display Data RAM und des Character Data RAM generiert.

- Dem Timing Generator, der den internen Takt zur Verfügung stellt.

- Dem Cursor/Blink Controller, der den Cursor oder das Blinksignal am Display erzeugt.

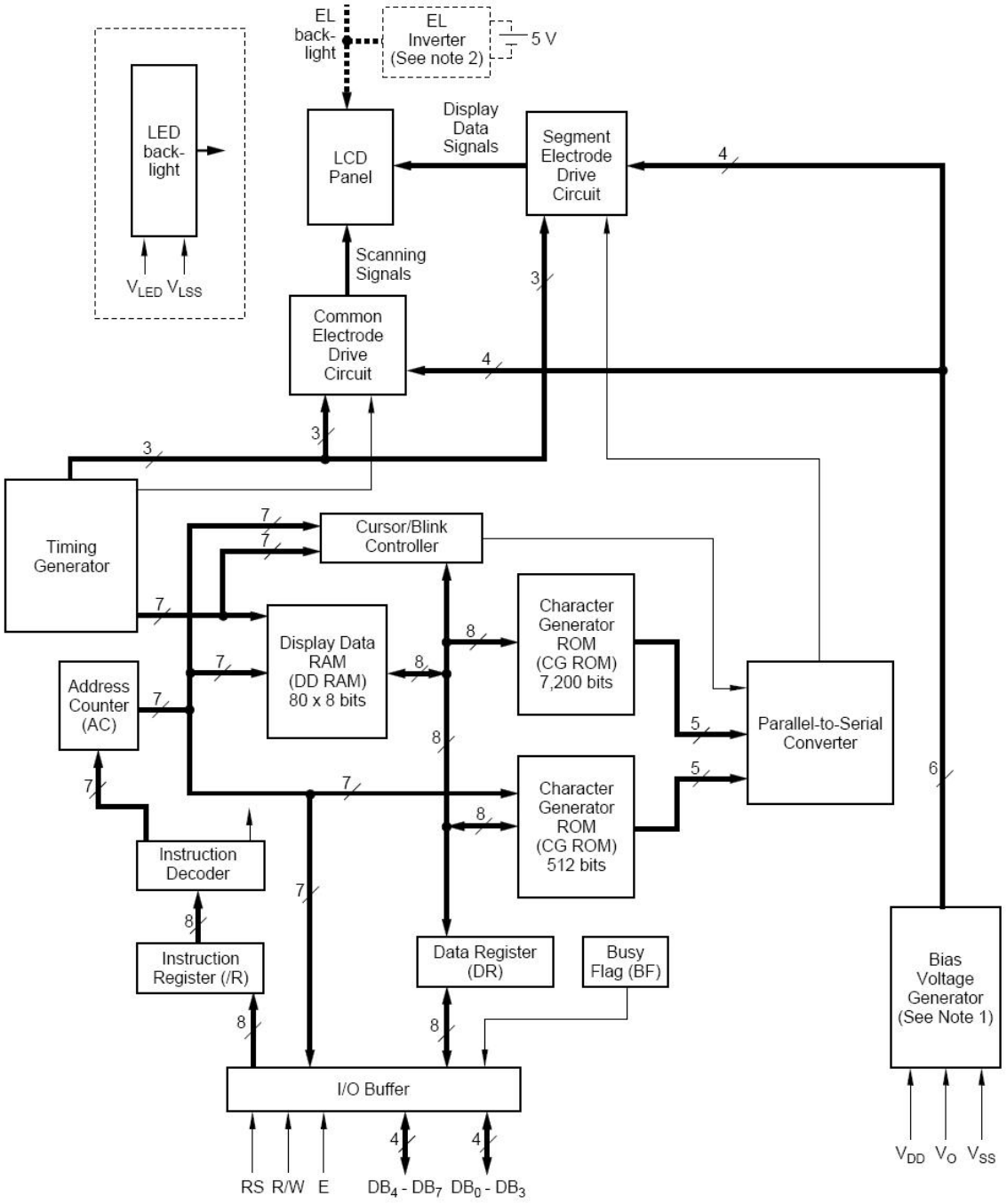
- Dem Parallel-to-Serial Converter, der den parallelen Datenstrom vom Character Generator ROM / RAM in einen seriellen Datenstrom, der vom Display Treiber benötigt wird, wandelt.

- Dem Bias Voltage Generator, mit dem eine Temperaturkompensation der Displayspannung erreicht werden kann.

- Dem LCD-Treiber, der die Displaydaten, die Timing-Singale und die Arbeitspunktspannung erhält und daraus Signale für das LCD bereitstellt.

- Dem eigentlichen Flüssigkristall LCD-Panel mit seinen 2 Zeilen, zu je 40 Zeichen.

Die einzelnen Komponenten des Displays sind im Blockdiagramm in Abbildung 16 zu sehen. Die Beschreibung der Schnittstellensignale findet in Tabelle 5 statt.




- Abbildung 16: Blockdiagramm Displaycontroller -

| SIGNAL NAME | INPUT/OUTPUT | EXTERNAL CONNECTION | FUNCTION |
|---|---|---|---|
| RS | Input | P05 (PD6) | Register select signal<br>"0": Instruction register (when writing)<br>Busy flag and address counter (when reading)<br>"1": Data register (when writing and reading) |
| R/W | Input | P06 (PD7) | Read/write select signal: "0": Writing; "1": Reading |
| E | Input | P07 (PB0) | Operation (data read/write) enable signal |
| $DB_4$ - $DB_7$ | Input/Output | P01 - P04 (PD2 - PD5) | High-order lines of data bus with three-state, bidirctional function for use in data transactions with the MPU. $DB_7$ may also be used to check the busy flag. |
| $DB_0$ - $DB_3$ | Input/Output | n.A. | Low-order lines of data bus with three-state, bidirectional function for use in data transactions with the MPU.<br>These lines are not used when interfacing with a 4-bit microprocessor. |
| $V_{DD}$, $V_{SS}$ | Power | P13, P14 | $V_{DD}$: +5 V, $V_{SS}$: GND |
| $V_0$ | Power | n.A. | Contrast adjustment voltage |

- Tabelle 5: Schnittstellensignale des Displays -

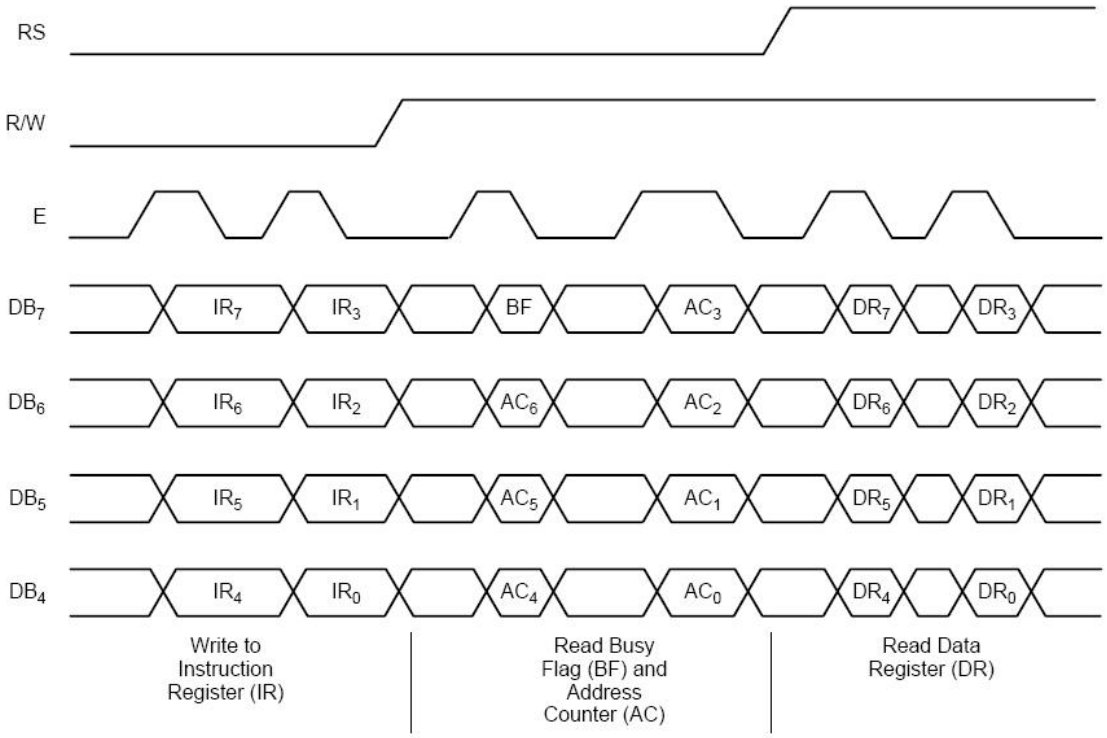
In Spalte 3 (External Connection) der Tabelle sind die die Pins der Schnittstelle SU1 auf dem Board myAVR Board 1 LPT (siehe Abbildung 4) vermerkt. In Klammern sind hierzu die Schnittstellen des ATmega8 zu sehen. Der Displaycontroller ist also folgendermaßen mit dem ATmega8 verbunden:

| Signalname | Port [Bit] des ATmega8 |
|---|---|
| Data Bus DB4 | Register Port D [2] |
| Data Bus DB5 | Register Port D [3] |
| Data Bus DB6 | Register Port D [4] |
| Data Bus DB7 | Register Port D [5] |
| Signal RS | Register Port D [6] |
| Signal R/W | Register Port D [7] |
| Signal E | Register Port B [0] |

- Tabelle 6: Anschluss der ext. Signale an ATmega8 -

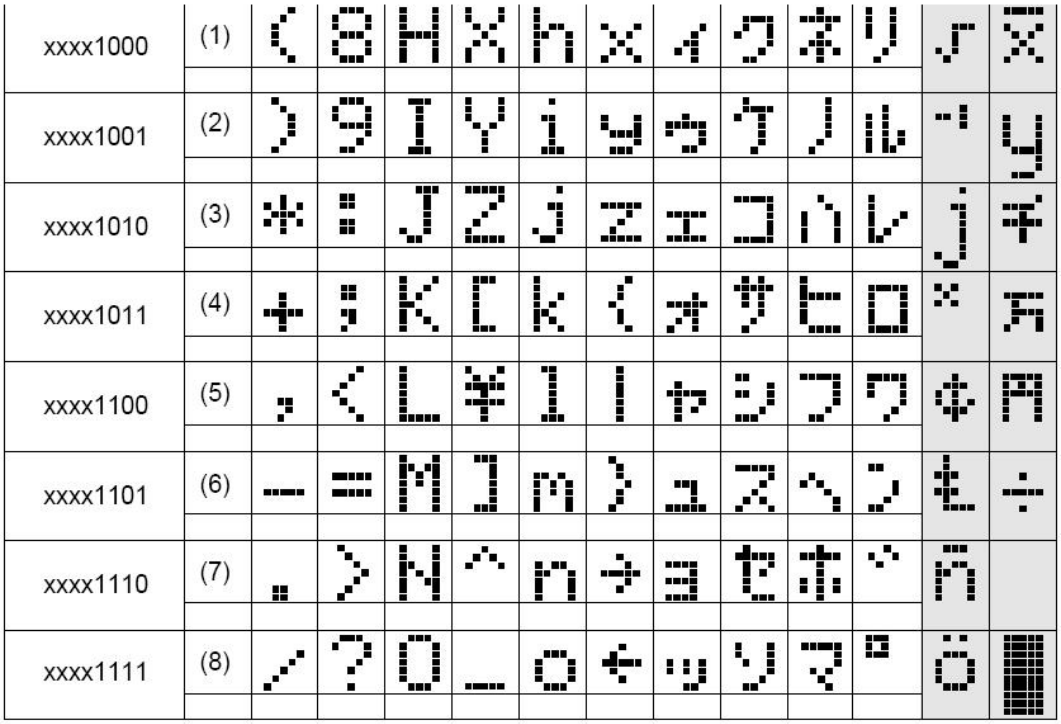
Die Versorgungsspannung von +5V (VDD) und die Masse VSS erhält der Displaycontroller von Pin 13 (+5V) bzw. von Pin 14 (GND) des Boards.

Die unteren Signale des Datenbuses (DB0 – DB3) bleiben unbelegt, da die 8-bit Zeichen des Displays auch mit nur einen 4-bit breitem Datensignal dargestellt werden können. Hierzu werden die 8-bit breiten Zeichen von Mikrocontroller in zwei 4-bit Datenübertragungen zum Display gesendet. Dieses Verfahren hat den Vorteil, dass anstatt 8 Bit der Register vom ATmega8 nur 4 Bit benötigt werden und trotzdem der gesamte Zeichensatz des Displays dargestellt werden kann (Abbildung 18). Die schematische Datenübertragung hierzu ist in Abbildung 17 dargestellt.




- Abbildung 17: 4-Bit Datenübertragung

| HIGH-ORDER 4 BIT / LOW-ORDER 4 BIT | 0000 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| xxxx0000 | CG RAM (1) | | | | | | | | | | | | |
| xxxx0001 | (2) | | | | | | | | | | | | |
| xxxx0010 | (3) | | | | | | | | | | | | |
| xxxx0011 | (4) | | | | | | | | | | | | |
| xxxx0100 | (5) | | | | | | | | | | | | |
| xxx0101 | (6) | | | | | | | | | | | | |
| xxx0110 | (7) | | | | | | | | | | | | |
| xxxx0111 | (8) | | | | | | | | | | | | |




- Abbildung 18: Zeichensatz des Displays -

## 1.5 Die Entwicklungsumgebung WinAVR

Die Programmierung des ATmega8 erfolgt mit dem WinAVR-Paket[6]. Neben der Windows Distribution des C-Compilers AVR-GCC für AVR-Mikrocontroller enthält das Paket noch die C-Bibliothek avr-libc die Programmiersoftware avrdude (zum schreiben des Programms in den Mikrocontroller) und den Editor Programmer's-Notepad.

Die C-Programme wurden mit dem Editor Programmer's-Notepad erstellet, mit dem AVR-GCC kompiliert und anschließend mit avrdude in den Mikrocontroller geschrieben. Um die Programmierung des Mikrocontrollers letztendlich durchführen zu können, war es noch notwendig den giveio-Treiber für den Parallelport des PCs zu installieren. Dieser ist ebenfalls im WinAVR-Paket enthalten und ist nach der Installation im Verzeichnis %Program Files%\WinAVR\bin\install_giveio.bat zu finden.

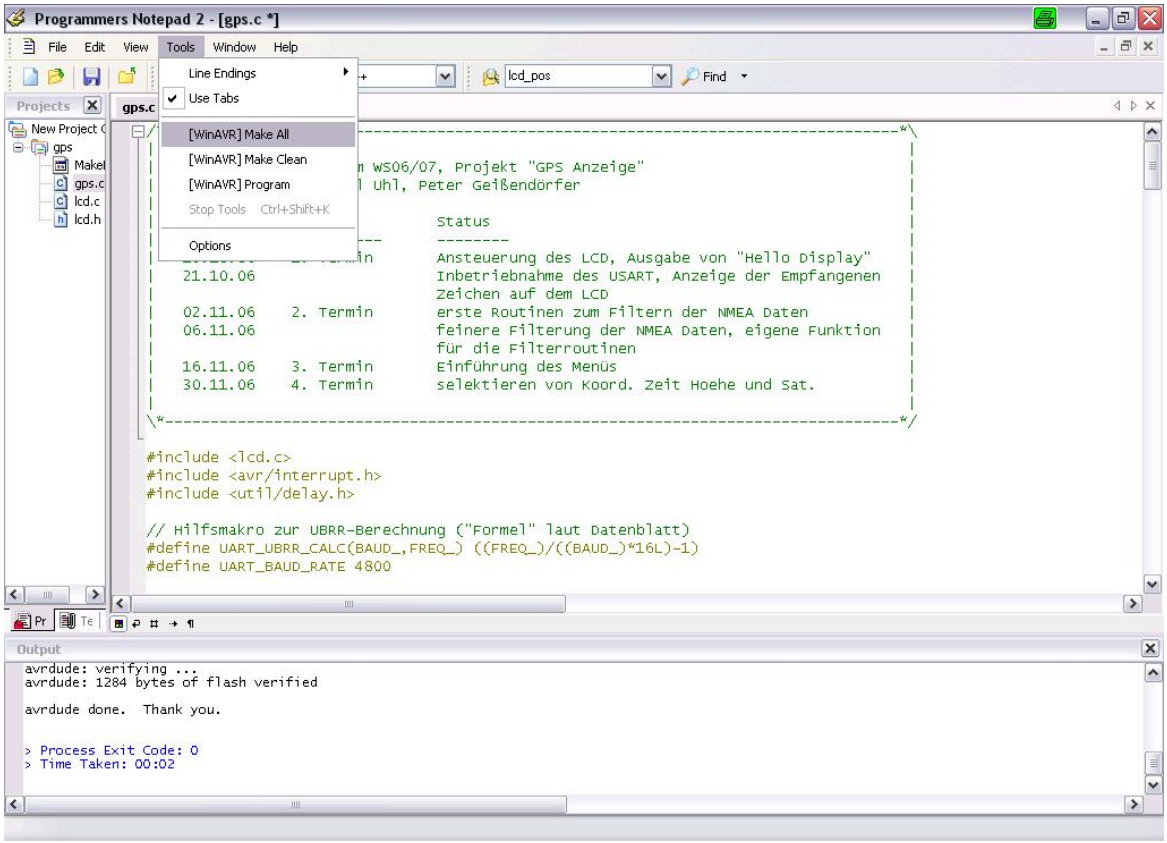
Da der AVR-GCC mit Makefiles arbeitet, war es noch notwendig vor dem kompilieren der Programme einmalig ein Makefile für unser Projekt zu erstellen. Hierzu wurde die im WinAVR-Paket enthaltenen Vorlage (%Program Files%\WinAVR\sample\Makefile) angepasst.

```
MCU = atmega8                   //Controller Typ
F_CPU = 1000000                 //Frequenz in Hz
TARGET = gps                    //Name der Programmdatei (ohne Endung)
AVRDUDE_PROGRAMMER = sp12       //welcher Programmieradapter
AVRDUDE_PORT = lpt1             //Anschluss am Rechner
```

- Code 2: Makefile des Projekts (Ausschnitt) -

Code 2 stellt die Zeilen im Makefile dar, die geändert wurden.

Das nun für unser Projekt angepasst Makefile wurde anschließend in das Projektverzeichnis (in dem ebenfalls der C Source Code und die Header Dateien zu finden sind) abgelegt.




- Abbildung 19: Screenshot Programmer's-Notepad. -

# 2 Inbetriebnahme des Displays

Um uns mit der Programmierung des ATmega8 vertraut zu machen, war unser Ziel in der 1. Phase die Ausgabe eines Textes auf dem Display zu realisieren.

Die erforderlichen Dateien lcd.c und lcd.h enthielt die LCD-Library[7] für HD44780-Displaycontroller. In der Datei lcd.c sind Standartroutinen für den Displaycontroller als Funktionen implementiert. Die Datei lcd.h enthält die zugehörige Headerdatei.

Die Headerdatei lcd.h mussten wir noch für unser Projekt angepasst werden:

```
#define XTAL              1000000      /**< clock frequency in Hz, for delay timer */
#define LCD_LINES         2            /**< number of visible lines of the display *
#define LCD_DISP_LENGTH   40           /**< visibles characters per line of the display */
#define LCD_PORT          PORTD        /**< port for the LCD lines   */
#define LCD_DATA0_PORT    LCD_PORT     /**< port for 4bit data bit 0 */
#define LCD_DATA1_PORT    LCD_PORT     /**< port for 4bit data bit 1 */
#define LCD_DATA2_PORT    LCD_PORT     /**< port for 4bit data bit 2 */
#define LCD_DATA3_PORT    LCD_PORT     /**< port for 4bit data bit 3 */
#define LCD_DATA0_PIN     2            /**< pin for 4bit data bit 0  */
#define LCD_DATA1_PIN     3            /**< pin for 4bit data bit 1  */
#define LCD_DATA2_PIN     4            /**< pin for 4bit data bit 2  */
#define LCD_DATA3_PIN     5            /**< pin for 4bit data bit 3  */
#define LCD_RS_PORT       LCD_PORT     /**< port for RS line         */
#define LCD_RS_PIN        6            /**< pin  for RS line         */
#define LCD_RW_PORT       LCD_PORT     /**< port for RW line         */
#define LCD_RW_PIN        7            /**< pin  for RW line         */
#define LCD_E_PORT        PORTB        /**< port for Enable line     */
#define LCD_E_PIN         0            /**< pin  for Enable line     */
```

- Code 3: lcd.h (Ausschnitt) -

Neben der Frequenz des Mikrocontrollers (XTAL) und der Größe des Displays (LCD_LINES, LCD_DISP_LENGHT) musste angegeben werden, auf welchen Port / Pin des Mikrocontrollers die Signalleitungen des Displaycontrollers liegen. Die Zuordnung geschah, wie in Abschnitt 1.4, Tabelle 6 beschrieben. Code 3 zeigt wiederum nur die angepassten Zeilen aus lcd.h.

Nachdem sämtlich benötigten Dateien vorhanden waren (Makefile, lcd.c, lcd.h) schrieben wir das Hauptprogramm gps.c:

```
#include <lcd.c>

int main(void)
{
       lcd_init(LCD_DISP_ON);
       lcd_clrscr();
       lcd_puts("Hello Display");
}
```

- Code 4: Hauptprogramm gps.c zur Displayinbetriebnahme-

Im Projektverzeichnis befanden sich nun alle notwendigen Dateien (Makefile, gps.c, lcd.c, lcd.h) um den ersten Kompilierversuch zu wagen. Dieser gelang uns gleich

aufs erste Mal, so dass wir die entstandene Datei gps.hex mit dem Programm avrdude in den ATmega8 schreiben konnten. Hierzu schlossen wir das mit dem myAVR Board 1 LPT mitgelieferten Nullmodemkabel an die COM-Schnittstelle des PCs und an die Steckerleiste X2 des Boards an. Nun mussten wir nur noch auf die Schaltfläche „Programmieren“ unter avrdude klicken (nach Auswahl des Schnittstelle am PC und einstellen der Datenübertragungsrate), und schon wurde das Programm geschrieben.

Nach einem Reset des ATmega8 mit dem Schalter S1 auf dem Board wurde nun das Programm ausgeführt und auf dem Display war „Hello Display“ zu lesen.

# 3 Anzeige der GPS-Zeit

Nachdem wir das Display erfolgreich in Betrieb genommen haben, und die ersten Zeichen ausgeben konnten, kam auch der GPS-Empfänger ins Spiel. Um uns mit diesen vertraut zu machen, nahmen wir uns vor die GPS-Zeit auszugeben.

Die Datensätze, die der GPS-Empfänger liefert sind nach dem, wie in Abschnitt 1.3.1 beschrieben, NMEA-0183 Protokoll kodiert. Zunächst schlossen wir den GPS-Empfänger an den PC über die RS232-Schnittstelle an. Anschließend starteten wir das Hyper-Terminal von Windows XP, wählten die richtige Verbindung aus und ließen uns somit den Datenstrom anzeigen. Anhand der Beschreibungen aus den Quellen 3 und 4 analysierten wir den Datenstrom.

```
$GPGSV,3,3,10,10,11,031,36,30,05,140,35*7B
$GPRMC,143626.943,A,4927.1555,N,01105.8498,E,0.00,,161106,,,A*7F
$GPVTG,,T,,M,0.00,N,0.0,K,A*13
$GPGGA,143627.943,4927.1555,N,01105.8498,E,1,09,0.9,390.9,M,,,,0000*03
$GPRMC,143627.943,A,4927.1555,N,01105.8498,E,0.00,,161106,,,A*7E
$GPVTG,,T,,M,0.00,N,0.0,K,A*13
$GPGGA,143628.943,4927.1570,N,01105.8525,E,1,09,0.9,371.0,M,,,,0000*0A
$GPRMC,143628.943,A,4927.1570,N,01105.8525,E,0.00,,161106,,,A*71
$GPVTG,,T,,M,0.00,N,0.0,K,A*13
$GPGGA,143629.943,4927.1576,N,01105.8536,E,1,09,0.9,360.2,M,,,,0000*0D
$GPRMC,143629.943,A,4927.1576,N,01105.8536,E,0.00,,161106,,,A*74
$GPVTG,,T,,M,0.00,N,0.0,K,A*13
$GPGGA,143630.943,4927.1574,N,01105.8513,E,1,09,0.9,343.1,M,,,,0000*02
$GPRMC,143630.943,A,4927.1574,N,01105.8513,E,0.00,,161106,,,A*79
$GPVTG,,T,,M,0.00,N,0.0,K,A*13
$GPGGA,143631.943,4927.1574,N,01105.8513,E,1,09,0.9,343.1,M,,,,0000*03
$GPGSA,A,3,07,18,31,10,16,03,21,25,06,,,,1.5,0.9,1.2*37
$GPGSV,3,1,10,07,83,330,30,21,74,134,32,16,57,295,30,06,39,086,45*79
```

- Aufzeichnung 2: Datensätze des GPS-Empfängers -

Um unser Projekt realisieren zu können benötigen wir den, wie bereits in Abschnitt 1.3.1, Abbildung 15 erwähnt, GPGGA-Datensatz. In diesem Datensatz sind die UTC-Zeit, die Koordinaten der nördlichen Breite und der östlichen Länge sowie weitere Informationen enthalten.

```
#include <lcd.c>
#include <avr/interrupt.h>

// Hilfsmakro zur UBRR-Berechnung ("Formel" laut Datenblatt)
#define UART_UBRR_CALC(BAUD_,FREQ_) ((FREQ_)/((BAUD_)*16L)-1)

#define UART_BAUD_RATE 4800


volatile uint8_t komma_count=0,zeichen_count=0,last_true=0;


ISR(USART_RXC_vect)                                    // USART receive complete
{
        uint8_t udr_temp;
```

```
        uint8_t tmp_sreg;                                       //  temporaerer  Speicher  fuer  das
                                                                    Statusregister
    tmp_sreg = SREG;                                            // Statusregister sichern
    cli();                                                      // Interrupts global deaktivieren
        udr_temp = UDR;
        if (udr_temp == '$')
        {
               //lcd_putc(udr_temp);
               //lcd_puts("$erkannt, ");
               zeichen_count=0;
               komma_count=0;
        }
        if (zeichen_count==4 && udr_temp=='G')                  // wenn das vierte Zeichen G ist...
        {
               last_true = 1;
               //lcd_puts("G erkannt, ");
        }
        if (zeichen_count==5 && udr_temp=='A' && last_true == 1)
        {
               last_true=2;
               lcd_clrscr();
               //lcd_puts("A erkannt, ");
        }
        if (last_true==2 && zeichen_count>5 && zeichen_count<15)
        {
               lcd_putc(udr_temp);                              // Inhalt des USART
                                                                   Empfangsregisters  auf  dem  LCD
                                                                   ausgeben
        }

        zeichen_count++;
        SREG = tmp_sreg;                                        // Status-Register wieder
                                                                   herstellen
        sei();                                                  // Interrupts global aktivieren
}

int main(void)
{
        // UART Baudrate, 16bit Register, aufgeteilt auf High und Low
        UBRRH = (uint8_t)( UART_UBRR_CALC( UART_BAUD_RATE, F_CPU ) >> 8 );
        UBRRL = (uint8_t)UART_UBRR_CALC( UART_BAUD_RATE, F_CPU );
        UCSRB |= ( 1 << RXEN )|( 1 << RXCIE );                      // UART RX und RX-Interrupt
                                                                    einschalten
        UCSRC |= ( 1 << URSEL )|( 1 << UCSZ1 )|( 1 << UCSZ0);       // Asynchron 8N1

        lcd_init(LCD_DISP_ON_CURSOR);                               // LCD initialisieren
        lcd_clrscr();                                               // LCD löschen

        sei();                                                      // Interrupts global
                                                                    aktivieren
}
```

- Code 5: Hauptprogramm gps.c zur Ausgabe der GPS-Zeit-

Die Auswahl des benötigten Datensatzes GPGGA erfolgte mittels einer Interrupt Service Routine (ISR). Als Auslöser für den Interrupt diente das Ereignis „UART Zeichen empfangen“ (`ISR(USART_RXC_vect)`). Das heißt, immer wenn der GPS-Empfänger ein Zeichen sendet wird die ISR ausgeführt. Innerhalb der ISR erfolgt die Auswahl des Datensatzes über mehrere if-Anweisungen. Falls der Anfang eines Datensatzes erkannt wurde („$“) werden die Zähler (Variablen) `zeichen_count` und `komma_count` auf Null gesetzt. Die beiden Variablen wurden dabei global als `volatile` deklariert, um den Compiler darauf hinzuweisen, dass die Variablen ihren Wert ändern können, auch ohne dass eine Zuweisung oder ein anderweitiger

Programmzugriff erfolgt. Der Compiler wird in diesem Fall also von zu aggressiver Optimierung abgehalten.
Die nächsten zwei Zeichen sind in jedem Datensatz gleich und sind deshalb bei der Auswahl des Datensatzes ohne belang. Der Zähler `zeichen_count` wird jedoch bei jeder Abarbeitung der ISR, also bei jedem empfangenen Zeichen inkrementiert.
Die nächste Abfrage erfolgt beim vierten empfangenen Zeichen (`(zeichen_count==4 && udr_temp=='G')`). Dort wird der Zähler `last_true` auf 1 gesetzt.
Falls das sechste empfangene Zeichen eines Datensatzes „A“ ist, der Zähler `zeichen_count` gleich 5 ist und der Zähler `last_true` gleich 1 ist, befindet man sich im für uns relevanten Datensatz GPGGA. Nun wird der Zähler `last_true` auf 2 gesetzt, das Display gelöscht und der Cursor auf die Position (0,0) gesetzt.
Zwischen dem sechsten und dem 15ten Zeichen werden mit der nächsten Abfrage (`last_true==2 && zeichen_count>5 && zeichen_count<15`) sämtliche empfangene Zeichen auf dem Display ausgegeben (`lcd_putc(udr_temp)`).
Am Anfang der ISR wurden noch die beiden temporären Variablen `udr_temp` (als Speicher für das USART-Empfangsregister (UDR)) und `tmp_sreg` (als Speicher für das Stausregister) deklariert. `tmp_sreg` wird benötigt um das Statusregister nach der Abarbeitung der ISR wieder herzustellen. `udr_temp` dient als Buffer für das USART-Empfangsregister. Dieser Buffer ist notwendig geworden, da vom Empfangsregister erst mit einem Zeichen Verspätung gelesen werden konnte und somit die Zähler nicht mehr mit der tatsächlichen Zeichenzahl übereinstimmten.
Während der ISR wurden sämtliche Interrupts global deaktiviert (`cli()` – ab diesem Zeitpunkt konnte kein Interrupt mehr ausgelöst werden) um zu gewährleisten, dass die ISR ohne Unterbrechung ablaufen kann. Am Ende der ISR wurden die Interrupts wieder global aktiviert (`sei()`).

Im Hauptprogramm wurde die bereits in Abschnitt 1.2.1 erwähnte Baudrate für den USART in das in High- und Lowteil unterteilte USART Baud Rate Registers geschrieben. Die Berechnung der Baudrate erfolgte mit dem Makro `#define UART_UBRR_CALC(BAUD_,FREQ_) ((FREQ_)/((BAUD_)*16L)-1)`. Anschließend wurde noch der Port USART RX und der dazugehörige Interrupt RXCIE eingeschalten und bekannt gemacht, dass die Daten an den RX-Port asynchron gelangen, das Display initialisiert, das Display gelöscht und die Interrupts wurden global aktiviert.

# 4 Anzeige der GPS-Koordinaten

Nachdem die Anzeige der GPS-Zeit funktionierte wagten wir uns an die eigentliche Funktion, nämlich der Anzeige der GPS-Koordinaten.

Als Grundgerüst des Programms diente das Programm zur Anzeige der GPS-Zeit. Eigentlich musste darin lediglich die Zeichenausgabe verschoben werden: statt `last_true==2 && zeichen_count>5 && zeichen_count<15` sollte die Abfrage `last_true==2 && zeichen_count>15 && zeichen_count<40` genügen. Um die formatierte Ausgabe der Koordinaten (nördliche Breite und östliche Länge) zu erreichen musste jedoch noch die Hilfsvariable `komma_count` eingeführt werden. Somit wurde es außerdem sinnvoller die Ausgabe der Zeichen nicht über die Zeichenanzahl (`zeichen_count>15 && zeichen_count<40`) sondern über die das Auftreten von Kommas (`komma_count >= 2 && komma_count <= 5`) zu realisieren. Das in der Ausgabe enthaltene Komma wurde dabei entfernt und durch ein Leerzeichen ersetzt.

```
#include <lcd.c>
#include <avr/interrupt.h>

// Hilfsmakro zur UBRR-Berechnung ("Formel" laut Datenblatt)
#define UART_UBRR_CALC(BAUD_,FREQ_) ((FREQ_)/((BAUD_)*16L)-1)
#define UART_BAUD_RATE 4800

// Bits für die "GPS Statusvariable" (GPSSV)
#define LCF 0  // last char found
#define INM 1  // in message
#define OOD 2  // out on display
#define MPO 4  // (und 5) Menü Position
#define MSE 6  // (und 7) Menü selected


volatile uint8_t komma_count=0,zeichen_count=0,GPSSV=0;

void handle_char(uint8_t udr_temp)   // Zeichen auswerten
{
       if (udr_temp == '$')
       {
              zeichen_count=0;
              komma_count=0;
              GPSSV &= ~(1<<LCF) & ~(1<<INM) & ~(1<<OOD);
       }

       // wenn das vierte Zeichen = G ist...
       if (zeichen_count==4 && udr_temp=='G')
       {
              GPSSV |= (1<<LCF);    // last char found = true
       }

       // wenn fünftes Zeichen = A und last char found = true ist...
       if (zeichen_count==5 && udr_temp=='A' && (GPSSV & (1<<LCF)))
       {
              GPSSV |= (1<<INM);    // in message = true
              lcd_gotoxy(0,0);
       }

       // wenn man innerhalb der Nachricht ist und das Zeichen ein Komma ist...
       if ((GPSSV & (1<<INM)) && udr_temp == ',')
       {
              komma_count++;
```

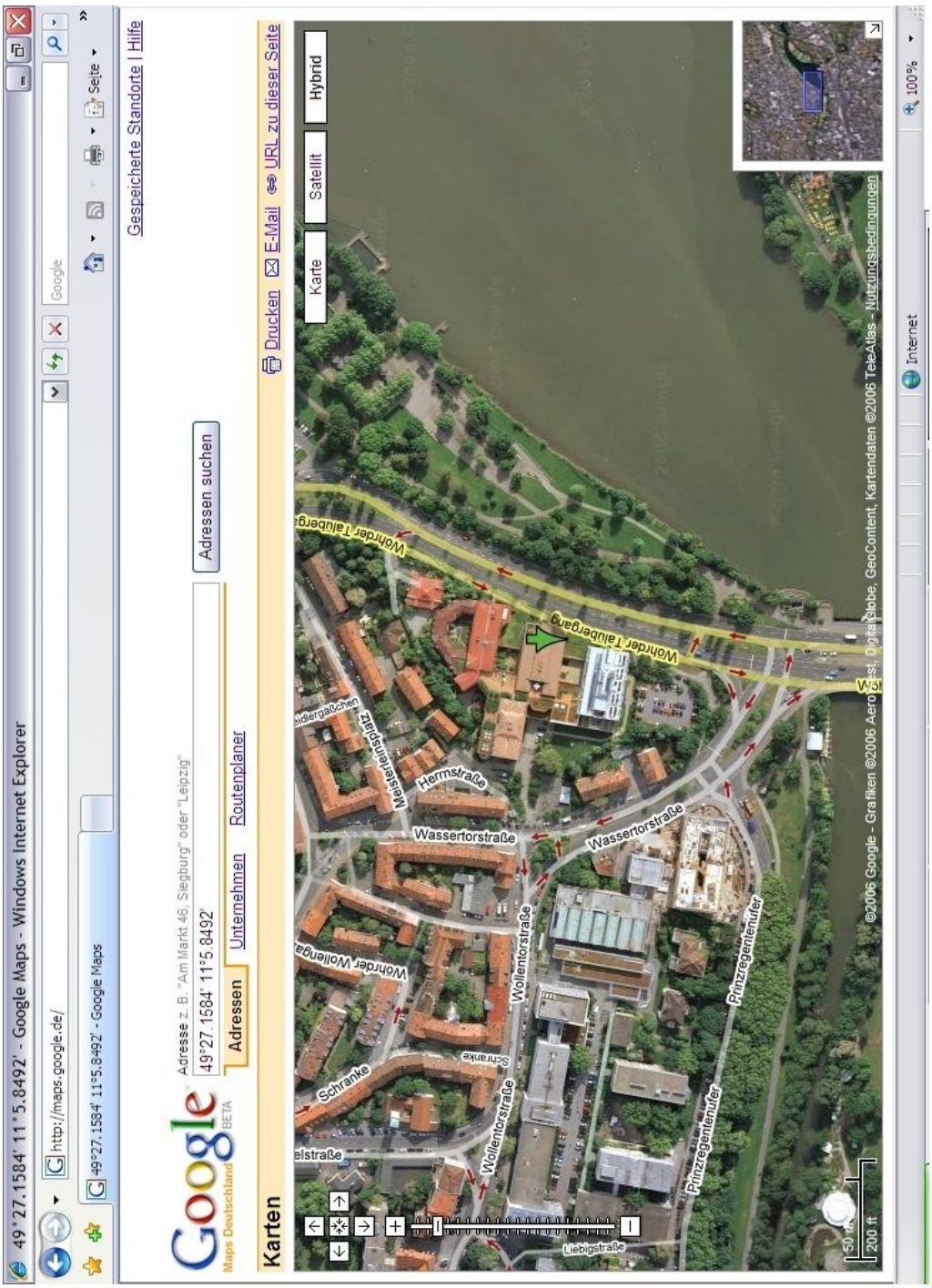
```
        }

        // wenn man sich zwischen dem 2. und 5. Komma befindet (Koordinaten)...
        if (komma_count >= 2 && komma_count <= 5)
        {
                GPSSV |= (1<<OOD);    // out on display = true
        }
        else
        {
                GPSSV &= ~(1<<OOD);   // out on display = false
        }

        // Leerzeichen nach dem Längengrad einfügen
        if (komma_count == 4 && udr_temp == ',')
        {
                lcd_putc(' ');
        }

        // wenn out on display = true ist und das Zeichen kein Komma ist...
        if ((GPSSV & (1<<OOD)) && udr_temp != ',')
        {
                lcd_putc(udr_temp);   // aktuelles Zeichen auf dem LCD ausgeben
        }

        zeichen_count++;
}

ISR(USART_RXC_vect)                   // USART receive complete
{
        uint8_t tmp_sreg;             // temporaerer Speicher fuer das Statusregister
        tmp_sreg = SREG;                      // Statusregister sichern
        cli();                        // Interrupts global deaktivieren

        handle_char(UDR);             // empfangenes Zeichen auswerten

        SREG = tmp_sreg;              // Status-Register wieder herstellen
        sei();                        // Interrupts global aktivieren
}

int main(void)
{
        // UART Baudrate, 16bit Register, aufgeteilt auf High und Low
        UBRRH = (uint8_t)( UART_UBRR_CALC( UART_BAUD_RATE, F_CPU ) >> 8 );
        UBRRL = (uint8_t)UART_UBRR_CALC( UART_BAUD_RATE, F_CPU );

        // UART RX und RX-Interrupt einschalten
        UCSRB |= ( 1 << RXEN )|( 1 << RXCIE );

        // Asynchron 8N1
        UCSRC |= ( 1 << URSEL )|( 1 << UCSZ1 )|( 1 << UCSZ0);


        lcd_init(LCD_DISP_ON);        // LCD initialisieren
        lcd_clrscr();                 // LCD löschen

        sei();                        // Interrupts global aktivieren
}
```

- Code 6: Hauptprogramm gps.c zur Ausgabe der GPS-Koordinaten-

Um das Programm universeller und lesbarer zu gestallten wurden die Zeichenfilterung und die damit verbundene Ausgabe aus der ISR in die Funktion `handle_char` verlagert. In der ISR erfolgt somit ausschließlich der Funktionsaufruf `handle_char(UDR)`. Zusätzlich wurde die Statusvariable `GPSSV` eingeführt um die Speicherzugriffe zu optimieren. Der Wert 0 (dezimal, 00000000 binär) der

Statusvariable GPSSV dient als Indikator für letzte abgefragte Zeichen (LCF, last char found), 1 (dezimal, 0000001 binär) gibt an, ob man sich in der interessierenden Nachricht befindet (INM, in message), 2 (dezimal, 00000010 binär) dient als „Schalter“ für die Displayausgabe (OOD, Out on Display).
Die ersten drei if-Abfragen entsprechen denen des im Abschnitt 3 vorgestellten Programms. Die vierte if-Abfrage realisiert den Kommazähler (`if ((GPSSV & (1<<INM)) && udr_temp == ',')`). Um Ressourcen zu sparen wird dieser nur erhöht, falls man sich in der interessierenden Nachricht befindet. Falls man sich nun zwischen dem zweiten und dem fünften Komma befindet (GPS-Koordinaten) (`if (komma_count >= 2 && komma_count <= 5)`), wird mit der Anweisung `GPSSV |= (1<<OOD);` die Displayausgabe aktiviert. Die eigentliche Ausgabe wird mit der letzten if-Anweisung (`if ((GPSSV & (1<<OOD)) && udr_temp != ',')`) durchgeführt (`lcd_putc(udr_temp);`). Um das Komma zwischen den dem Latitude und dem Longitude zu entfernen wird das vierte Komma im GPGGA-Datensatz durch ein Leerzeichen ersetzt (`if (komma_count >= 2 && komma_count <= 5) { GPSSV |= (1<<OOD);}`).

Nach dem kompilieren und der Programmierung des ATmega8 wurden am Display die Koordinaten 4927.1584N 01105.8429E angezeigt. Zur Kontrolle wurden diese mit Google Maps überprüft[8] (Abbildung 20).




- Abbildung 20: Überprüfung der ausgegebenen GPS-Koordinaten -

# 5 Anzeige diverser GPS-Daten (Auswahlmenü)

Neben der reinen Ausgabe der GPS-Zeit und der GPS-Koordinaten wurden nun mehrere Informationen aus dem GPGGA-Datensatz ausgewertet und angezeigt. Da das Display für formatierte Anzeige nicht ausreicht, wurde ein Auswahlmenü implementiert. Als anzeigbare Elemente stehen die Koordinaten (Menüeintrag 1), die GPS-Zeit (Menüeintrag 2), die Höhe über N.N. (Menüeintrag 3) und die Anzahl der bei der Messung zur Verfügung stehenden Satelliten (Menüeintrag 4) zur Verfügung. Mittels des Schalters S2 auf dem Board lassen sich die einzelnen Menüeinträge auswählen.

Für die Darstellung des Menüs ist die Funktion `void reset_menu(uint8_t pos1, uint8_t pos2)` zuständig. Darin wird zum einen die Anzeige der Menüeinträge realisiert (`lcd_gotoxy(0,1); lcd_puts(" Koord. Zeit Hoehe Sat. ");`) und zum anderen wird die Kennzeichnung der aktuellen Menüeintragsselektion mittels „<Menüeintrag>“ (`lcd_gotoxy(pos1,1); lcd_putc('>'); lcd_gotoxy(pos2,1); lcd_putc('<');`) verwirklicht. Da es sich bei dem Schalter S2 um einen mechanischen Taster (mit allen „negativen“ Effekten) handelt, wurde dieser mit der Funktion `uint8_t get_key(void)` entprellt. Dies wurde einfachkeitshalber mit zwei Verzögerungszeiten bei Pegeländerung an PINB/PB1 gelöst (`_delay_ms(100);`).

Die Funktion `void handle_char(uint8_t udr_temp)` zur Auswahl des GPGGA-Datensatzes musste aufgrund der Selektionsmöglichkeit innerhalb des GPGGA-Datensatzes erweitert werden. Hierzu wurde die `switch (GPSSV>>MPO)` -Anweisung eingeführt. Diese wertet die Menüposition aus und übergibt der Funktion `void display(uint8_t von, uint8_t bis)` die Parameter `von` und `bis`. Innerhalb von Case 0 (Anzeige der Koordinaten) wird, wie bereits in Abschnitt 4 realisiert, das Komma zwischen den Koordinaten durch ein Leerzeichen ersetzet. Um den Punkt und die anschließenden Zeichen innerhalb des GPS-Zeit-Datensatzes (Case 1) zu entfernen (die Zeichen nach dem Punkt innerhalb der GPS-Zeit kennzeichnen die aktuelle Differenz zwischen der GPS-Zeit und der tatsächlichen UTC-Zeit (die Zeitabweichung durch die Erdrotation wird innerhalb der GPS-Zeit nicht berücksichtigt)) musste die Funktion `void display(uint8_t von, uint8_t bis)` implementiert werden (sonstige Punkte innerhalb des GPGGA- Datensatzes sollen abgebildet werden). Die generelle Ausgabe erfolgt wie gehabt mit der Funtion `lcd_putc(udr_temp)`.

In der Funktion `int main (void)` musste noch der der Schalter S2 auf PINB/PB1 „gelegt“ werden und der zugehörige Pull-Up-Widerstand an diesem Pin aktiviert werden (`DDRB &= ~(1<<DDB1); PORTB |= (1<<PB1);`). Außerdem wurde noch die formatierte Ausgabe der Menüeinträge mit den verschiedenen Übergabewerten für die Funktionen implementiert.

Der C-Code (mit ausführlichen Kommentaren) ist in Code 6 zu sehen.

```
#include <lcd.c>
#include <avr/interrupt.h>
#include <util/delay.h>

// Hilfsmakro zur UBRR-Berechnung ("Formel" laut Datenblatt)
#define UART_UBRR_CALC(BAUD_,FREQ_) ((FREQ_)/((BAUD_)*16L)-1)
#define UART_BAUD_RATE 4800

// Bits für die "GPS Statusvariable" (GPSSV)
#define LCF 0         // last char found
#define INM 1         // in message
#define OOD 2         // out on display
#define PKT 3         // Punkt war da (für die Zeitanzeige)
#define MPO 4         // (und 5) Menü Position


volatile uint8_t komma_count=0,zeichen_count=0,GPSSV=0,lcd_pos=0;

uint8_t get_key(void)                       // Entprellung der Taste an PINB/PB1 (low aktiv)
{
       if (!(PINB&(1<<PB1)))                // wenn der Taster gedrückt...
       {
              _delay_ms(100);               // 100ms warten
              if (PINB&(1<<PB1))            // wenn der Taster nicht mehr gedrückt...
              {
                     _delay_ms(100);        // 100ms warten
                     return 1;
              }
       }
       return 0;
}

void reset_menu(uint8_t pos1, uint8_t pos2) // das Menü zurücksetzen
{
       uint8_t tmp_sreg;                    // temporaerer Speicher fuer das Statusregister
       tmp_sreg = SREG;                     // Statusregister sichern
       cli();                               // Interrupts global deaktivieren

       lcd_clrscr();                        // LCD löschen
       lcd_pos=0;

       lcd_gotoxy(0,1);

       //01234567890123456789012345678 (Position auf dem LCD)
       lcd_puts(" Koord.  Zeit  Hoehe  Sat. ");

       // Zeiger für die aktuelle Menüposition
       lcd_gotoxy(pos1,1);
       lcd_putc('>');
       lcd_gotoxy(pos2,1);
       lcd_putc('<');

       lcd_gotoxy(0,0);
       lcd_pos=0;

       SREG = tmp_sreg;                     // Status-Register wieder herstellen
       sei();                               // Interrupts global aktivieren
}

void display(uint8_t von, uint8_t bis)      // Ausgabe von Komma "von" bis Komma "bis"
{
       if (komma_count >= von && komma_count <= bis)
       {
              if (!(GPSSV & (1<<PKT)))      // wenn vor dem Punkt in der Nachricht "Zeit"...
                     GPSSV |= (1<<OOD);     // out on display = true
              else
                     GPSSV &= ~(1<<OOD);    // out on display = false
       }
       else
       {
```

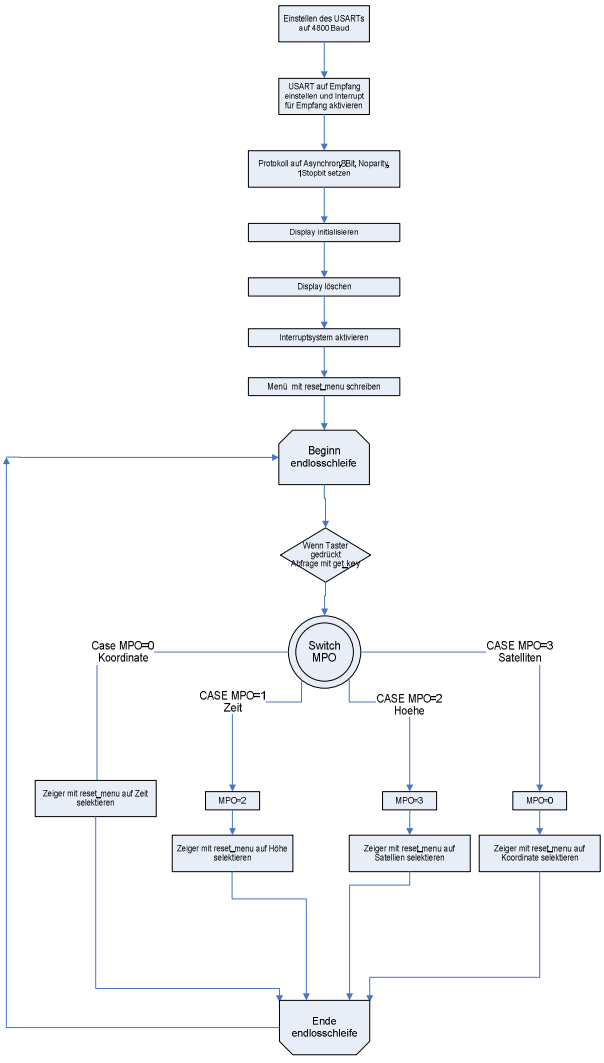
```
                GPSSV &= ~(1<<OOD);          // out on display = false
                GPSSV &= ~(1<<PKT);          // Punkt zurücksetzen
        }
}

void handle_char(uint8_t udr_temp)          // Zeichen auswerten
{
        lcd_gotoxy(lcd_pos,0);              // Cursorposition wiederherstellen

        // wenn Anfang einer neuen GPS Nachricht...
        if (udr_temp == '$')
        {
                zeichen_count=0;
                komma_count=0;
                GPSSV &= ~(1<<LCF) & ~(1<<INM) & ~(1<<OOD);
        }

        // wenn das vierte Zeichen = G ist...
        if (zeichen_count==4 && udr_temp=='G')
        {
                GPSSV |= (1<<LCF);           // last char found = true
        }

        // wenn fünftes Zeichen = A und last char found = true ist...
        if (zeichen_count==5 && udr_temp=='A' && (GPSSV & (1<<LCF)))
        {
                GPSSV |= (1<<INM);           // in message = true
                lcd_gotoxy(0,0);
                lcd_pos=0;
        }

        // wenn man innerhalb der Nachricht ist und das Zeichen ein Komma ist...
        if ((GPSSV & (1<<INM)) && udr_temp == ',')
        {
                komma_count++;
        }

        switch (GPSSV>>MPO)
        {
                case 0:                                          // Koord. ausgewählt...
                {
                        display(2,5);  //von Komma 2 bis Komma 5
                        if (komma_count == 4 && udr_temp == ',')    //   Leerzeichen   nach   dem
                                                                    Längengrad einfügen
                        {
                                lcd_putc(' ');                      // Leerzeichen ausgeben
                                lcd_pos++;                          // Positionsspeicher erhöhen
                        }
                }
                break;

                case 1:                                          // Zeit ausgewählt...
                        if (komma_count == 1 && udr_temp == '.')    //  wenn  in  der  Nachricht
                                                                    Zeit der Punkt erreicht...
                                GPSSV |= (1<<PKT);                  // PKT = true setzen
                        display(1,1);                               // von Komma 1 bis Komma 1
                break;

                case 2:                                          // Hoehe ausgewählt...
                        display(9,10);                              // von Komma 9 bis Komma 10
                break;

                case 3:                                          // Sat. ausgewählt...
                        display(7,7);                               // von Komma 7 bis Komma 7
                break;
        }


        // wenn out on display = true ist und das Zeichen kein Komma ist...
        if ((GPSSV & (1<<OOD)) && udr_temp != ',')
```

```
	{
		lcd_putc(udr_temp);		// aktuelles Zeichen auf dem LCD ausgeben
		lcd_pos++;			// Positionsspeicher erhöhen
	}

	zeichen_count++;
}

ISR(USART_RXC_vect)			// USART receive complete
{
	uint8_t tmp_sreg;		// temporaerer Speicher fuer das Statusregister
    tmp_sreg = SREG;			// Statusregister sichern
    cli();				// Interrupts global deaktivieren

	handle_char(UDR);		// empfangenes Zeichen auswerten

	SREG = tmp_sreg;		// Status-Register wieder herstellen
	sei();				// Interrupts global aktivieren
}

int main(void)
{
	// UART Baudrate, 16bit Register, aufgeteilt auf High und Low
	UBRRH = (uint8_t)( UART_UBRR_CALC( UART_BAUD_RATE, F_CPU ) >> 8 );
	UBRRL = (uint8_t)UART_UBRR_CALC( UART_BAUD_RATE, F_CPU );

	// UART RX und RX-Interrupt einschalten
	UCSRB |= ( 1 << RXEN )|( 1 << RXCIE );

	// Asynchron 8N1
	UCSRC |= ( 1 << URSEL )|( 1 << UCSZ1 )|( 1 << UCSZ0);

	DDRB &= ~(1<<DDB1);		// PortB, Pin1 als Eingang (S.53)
	PORTB |= (1<<PB1);		// Pullup aktivieren

	lcd_init(LCD_DISP_ON);		// LCD initialisieren
	lcd_clrscr();			// LCD löschen

	sei();				// Interrupts global aktivieren

	reset_menu(0,7);		// das Menü zurücksetzen

	for(;;)
	{
		if (get_key())				// Wenn Taster gedrückt
		{
			switch (GPSSV>>MPO)		// Registerinhalt als Integerwert
			{
				case 0:			// wenn auf "Koord."...
				{
					GPSSV &= ~(3<<MPO);	// alte Werte löschen
					GPSSV |= (1<<MPO);	// gehe auf "Zeit"
					reset_menu(8,13);	// Zeiger am LCD setzen
				}
				break;
				case 1:			// wenn auf "Zeit"...
				{
					GPSSV &= ~(3<<MPO);	// alte Werte löschen
					GPSSV |= (2<<MPO);	// gehe auf "Hoehe"
					reset_menu(14,20);	// Zeiger am LCD setzen
				}
				break;
				case 2:			// wenn auf "Hoehe"...
				{
					GPSSV &= ~(3<<MPO);	// alte Werte löschen
					GPSSV |= (3<<MPO);	// gehe auf "Sat."
					reset_menu(21,26);	// Zeiger am LCD setzen
				}
				break;
				case 3:			// wenn auf "Sat."...
```

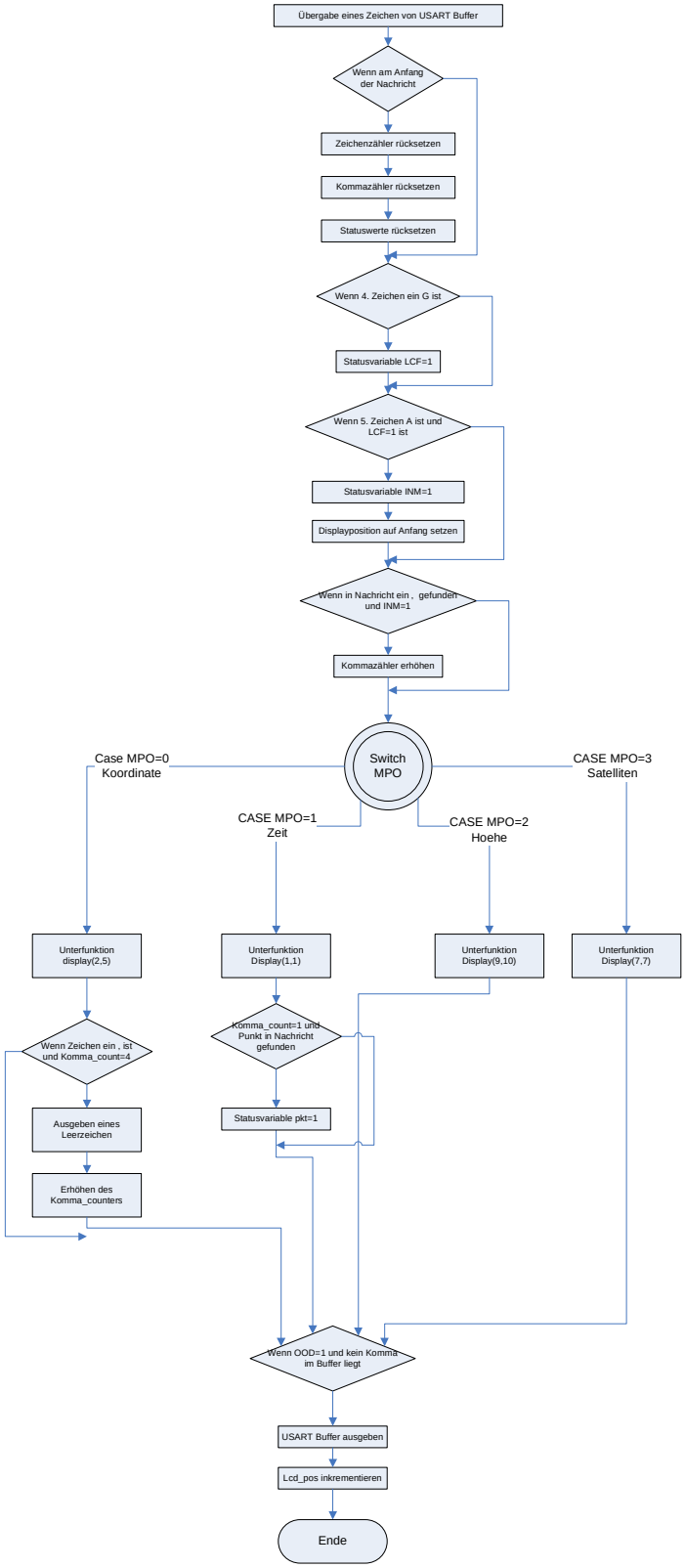
```
                        {
                                GPSSV &= ~(3<<MPO);    // gehe auf "Koord."
                                reset_menu(0,7);       // Zeiger am LCD setzen
                        }
                        break;
                        default:                       // sollte nie erreicht werden
                        {
                                GPSSV &= ~(3<<MPO);    // gehe auf "Koord."
                                reset_menu(0,7);
                        }
                        break;
                }
        }
    }
}
```

- Code 7: Hauptprogramm gps.c zur Ausgabe des GPGGA-Datensatzes-

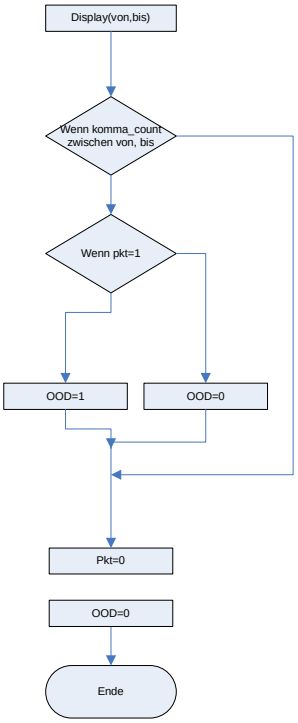
Der Ablauf des Programms wird zur Verdeutlichung in nachfolgenden Flussdiagrammen dargestellt:



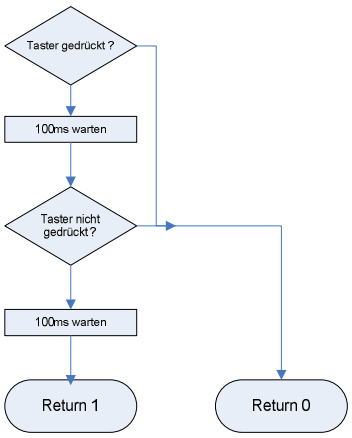

- Abbildung 21: Flussdiagramm zu int main (void) -



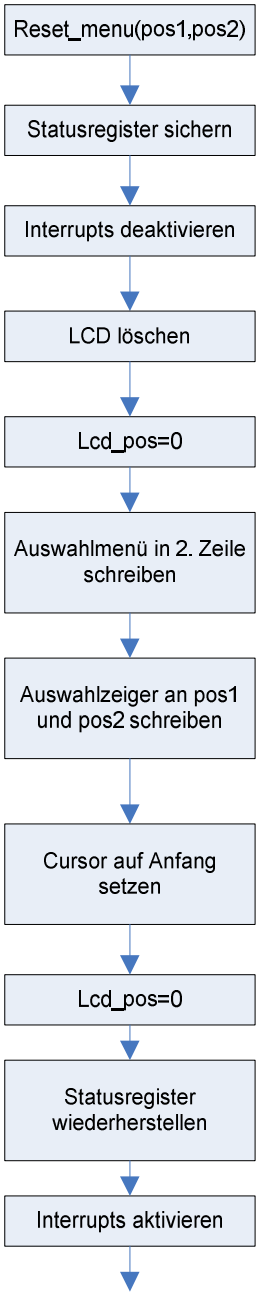

- Abbildung 22: Flussdiagramm zu void handle_char(uint8_t udr_temp) -



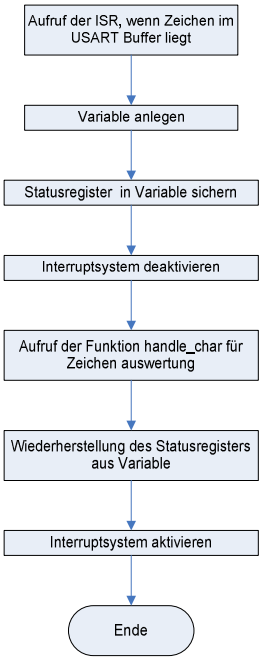

- Abbildung 23: Flussdiagramm zu void display(uint8_t von, uint8_t bis) -




- Abbildung 24: Flussdiagramm zu uint8_t get_key(void) -

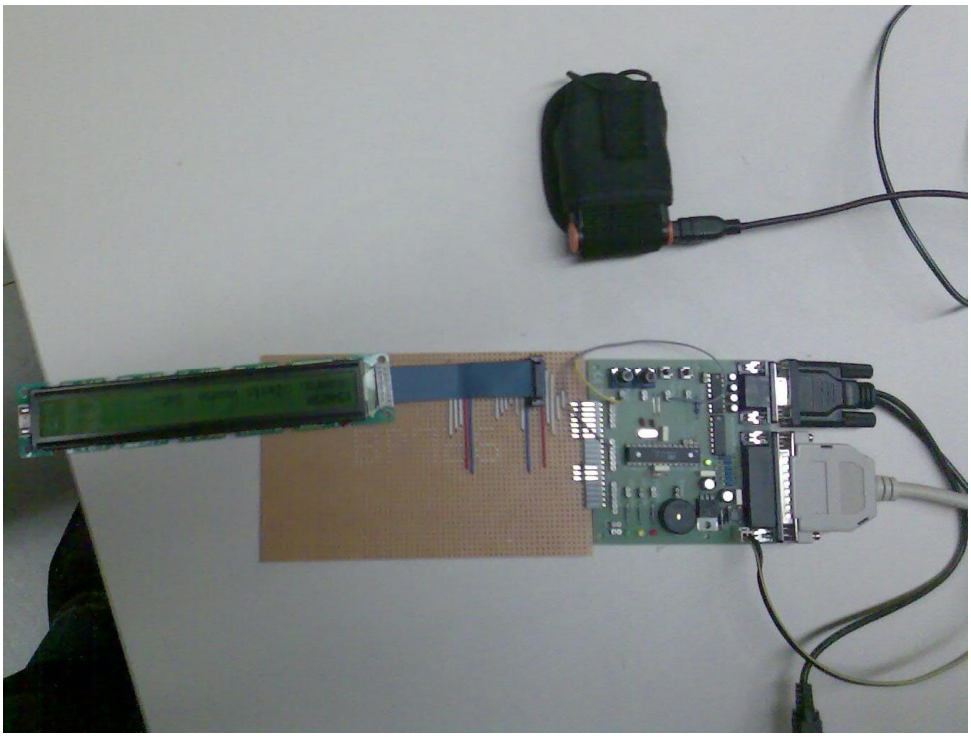



- Abbildung 25: Flussdiagramm zu void reset_menu(uint8_t pos1, uint8_t pos2) -




- Abbildung 26: Flussdiagramm zu ISR(USART_RXC_vect) -

In Abbildung 27 ist der Versuchsaufbau mit sämtlichen Komponenten als Photographie zu sehen.



- Abbildung 27: Versuchsaufbau. -

# 6 Quellenverzeichnis

[1] http://www.myavr.de/
[2] http://www.atmel.com/dyn/resources/prod_documents/doc2486.pdf
[3] http://www.haicom-gps.de/downloads/HI-303S.pdf
[4] http://www.kowoma.de/gps/zusatzerklaerungen/NMEA.htm
[5] http://bray.velenje.cx/avr/PDFs/lcd-spec.pdf
[6] http://winavr.sourceforge.net/
[7] http://homepage.hispeed.ch/peterfleury/lcdlibrary.zip
[8]
http://maps.google.de/maps?f=q&hl=de&q=49%C2%B027.1584%27+11%C2%B005.8429%27&ie=UTF8&z=17&ll=49.452866,11.098434&spn=0.003292,0.01075&t=h&om=1