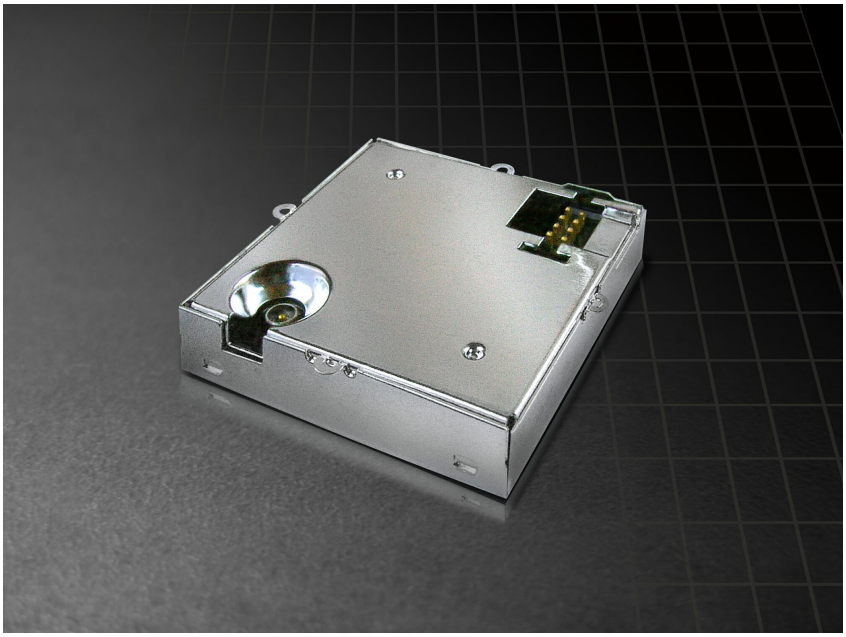

Softwarebeschreibung der Auswertung des NMEA-Protokolls mit dem Mikrocontroller Atmega128 von Atmel

von
Ralf Hochhausen
micro@ralf-hochhausen.de

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Nachdem ich in meinem letzten Artikel zum NMEA Data Link Layer beschrieben habe wie die NMEA-Daten vom GPS-Empfänger in den Mikrocontroller kommen dreht sich in diesem Artikel alles um die Auswertung der eigentlichen NMEA-Daten. D.h. ich werde beschreiben wie die benötigten Daten aus den NMEA-Sätzen herausgefiltert und wie sie von der Software interpretiert werden. Der Artikel beginnt dabei mit der Beschreibung der Anforderungen welche ich vor der Entwicklung an die Software gestellt habe. Diese bilden die Grundlage der Software und legen die Rahmenbedingungen für die Entwicklung fest. Daran schließt sich die Beschreibung des eigentlichen Softwaredesigns an in dem ich detailliert auf die Struktur und die Funktionsweise des Programms eingehe. Abgeschlossen wird der Artikel mit der Dokumentation der Anwendung der Software. Hier werde ich mit Hilfe von ein paar Beispielen erklären wie das Programm mit relativ wenig Aufwand an eine spezielle Anwendung angepasst werden kann.

2 Softwareanforderungen

Bevor ich mit der Entwicklung des NMEA Parsers begonnen habe, habe ich mir zunächst überlegt, welche besonderen Eigenschaften die Software am Ende besitzen, bzw. welche Anforderungen die Software erfüllen soll. Bei der Entwicklung habe ich die folgenden wesentlichen Anforderungen zugrunde gelegt:

- Modularität
- Flexibilität
- Leichte Erweiterbarkeit

Damit diese Schlagwörter nicht einfach so stehen bleiben werde ich die einzelnen Punkte im Folgenden etwas näher erläutern.

2.1 Anforderung: Modularität

Beim Punkt ‚Modularität' geht es mir vor allem um das Design der Schnittstellen des Parsermoduls. Es sollen nur wenige definierte Schnittstellen zur Verfügung gestellt werden und diese sollen unabhängig von einer darüber liegenden Anwendung sein. Diesem Ansatz liegt zugrunde, dass ich nicht für jede Anwendung einen neuen Parser schreiben möchte. Stattdessen war mein Ziel, einen bestimmten Satz von Software als eine Art Bibliothek zu entwickeln. Diese Bibliothek soll unabhängig von der eigentlichen Anwendung sein und kann somit für die verschiedensten Zwecke verwendet werden. Damit ist es z.B. möglich einen einfachen Datenlogger, der einfach die empfangenen Positionsdaten auf einem Datenträger speichert, oder aber auch einen GPS-basierten Kilometerzähler, der die Positionsdaten numerisch weiterverarbeitet, mit dem gleichen Parser zu realisieren.

2.2 Anforderung: Flexibilität

Die Anforderung der Flexibilität ist sehr eng mit der Modularität verknüpft. Hierbei geht es jedoch mehr um die Anpassungsfähigkeit der Software bei der Interpretation der GPS-Daten. Wie ich oben schon erwähnt habe, möchte ich den NMEA-Parser für verschiedene Anwendungen verwenden können. So kann es zum Beispiel für einen einfachen Datenlogger ausreichen die NMEA-Daten als Strings zu interpretieren, wohingegen eine Entfernungsmessung eher mit numerischen Daten (z.B. im float-Format) arbeitet. Es muss also möglich sein, die Interpretation der Daten auf einfache Weise an eine spezielle Anwendung anzupassen.

2.3 Anforderung: Leichte Erweiterbarkeit

GPS-Empfänger stellen gewöhnlich verschiedene NMEA-Sätze zur Verfügung welche die verschiedensten Informationen enthalten. Da zu Beginn einer Entwicklung meist noch nicht feststeht welche Informationen der verschiedenen NMEA-Sätze am Ende benötigt werden, soll es möglich sein, den Parser einfach zu erweitern. Dies betrifft zum einen die Anzahl der ausgewerteten NMEA-Sätze selbst und zum anderen das Herausfiltern der benötigten Informationen aus einem speziellen NMEA-Satz.

3 Softwaredesign

Nachdem nun die wichtigsten Anforderungen geklärt sind, geht es in diesem Kapitel um das Design des NMEA-Parsers. Hierbei beginne ich zunächst mit der groben Softwarestruktur die, die Verknüpfung der einzelnen Module beschreibt. Weiter geht es dann mit der Abbildung der Daten durch den Parser. Abgeschlossen wird dieses Kapitel mit der Beschreibung des Verhaltens des Parsers zur Laufzeit des Programms. Die folgenden drei Abschnitte stellen damit die Softwarearchitektur in Form von drei Sichten dar.

3.1 Struktursicht

Die Entwicklung des Parsers habe ich mit dem Design einer groben Softwarestruktur begonnen. Diese Struktur beschreibt den statischen Aufbau und das Zusammenwirken der einzelnen Softwaremodule des NMEA-Parsers, sowie die Abhängigkeiten des Parsers von anderen Modulen. D.h. es wird nicht beschrieben, wie sich die Software zur Laufzeit verhält.

Bei der Festlegung der Softwarestruktur war es besonders wichtig, dass die obigen Anforderungen auch bereits mit in das Design einfließen. Aus den Anforderungen habe ich dann drei einzelne Softwaremodule abgeleitet, die durch jeweils eine C-Quelle abgebildet werden. Das sind:

- nmeaparser.c
- nmeaconv.c
- nmeaframecfg.c

Beim Modul ‚nmeaparser.c' handelt es sich um den eigentlichen Parser der die NMEA-Sätze interpretiert und die gewünschten Daten aus den Sätzen herausfiltert. Dieses Modul ist völlig unabhängig von den eigentlichen Daten und kann damit auch ohne Änderungen für die verschiedensten Anwendungen verwendet werden. Unter Berücksichtigung der Anforderungen ‚Flexibilität' und ‚Erweiterbarkeit' werden die applikationsspezifischen Anpassungen in den beiden Modulen ‚nmeaconv.c' und ‚nmeaframecfg.c' gekapselt. Das Modul ‚nmeaconv.c' stellt hierbei alle Funktionen zur Verfügung, welche für die Interpretation und Umwandlung der NMEA-Daten notwendig sind. Die eigentlichen Funktionstypen sind dabei allgemeingültig, sodass bei einer Anpassung der Dateninterpretation keine Anpassung des Hauptmoduls ‚nmeaparser.c' notwendig ist. Lediglich die Implementierung kann an die jeweiligen Erfordernisse der Anwendung angepasst werden. Welche NMEA-Sätze und welche in den Sätzen enthaltenen Daten herausgefiltert werden müssen ist in der Datei ‚nmeaframecfg.c' gekapselt. In dieser Datei erfolgt die Konfiguration des Parsers in Form von Tabellen. Das bedeutet, dass bei einer Erweiterung der ausgewerteten Daten lediglich eine Anpassung in einer Tabelle dieses Moduls erforderlich ist. Die anderen Module bleiben davon unberührt. Das Modul ‚nmeaframecfg.c' stellt damit die Schnittstelle zur eigentlichen Anwendung dar. Durch diese Struktur ist die Anforderung der Modularität erfüllt. Der NMEA-Parser benötigt keine Schnittstelle zur eigentlichen Anwendung. Die einzige erforderliche Schnittstelle ist zur unteren Schicht, dem NMEA Data Link Layer, gerichtet welcher die empfangenen NMEA-Da-

ten zur Verfügung stellt. Es wird hierbei vorrausgesetzt, dass jeder NMEA-Satz mit dem Identifier (z.B. GPGGA) beginnt und mit dem \0-Zeichen abgeschlossen wird.

In der folgenden Abbildung ist die Struktur des Parsers dargestellt:

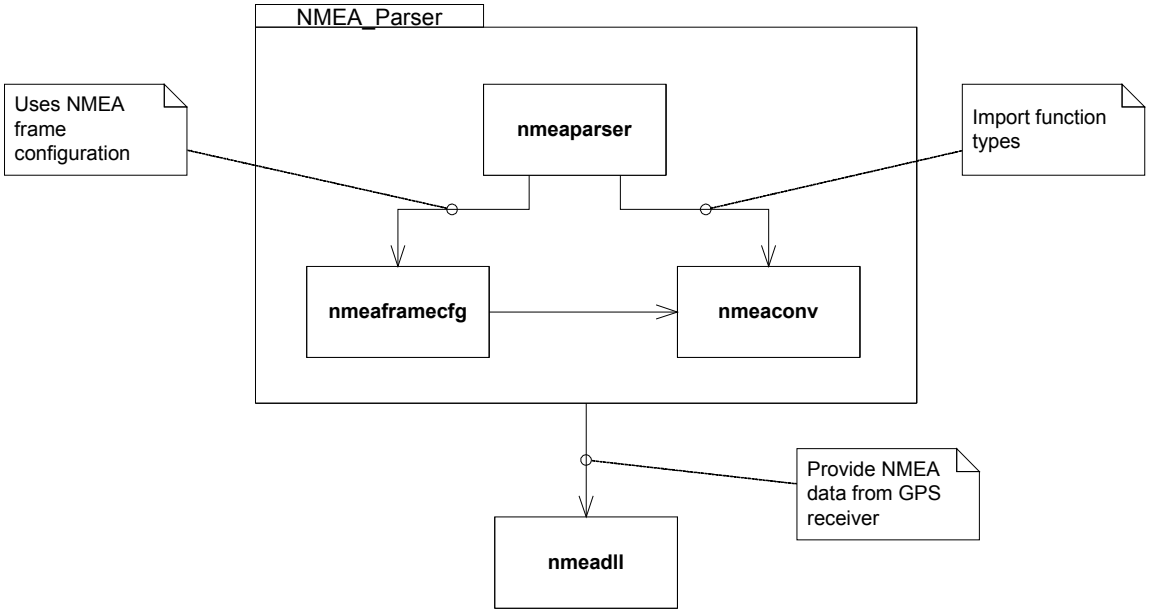

Abbildung 3-1: Struktursicht des NMEA Parsers

3.2 Datensicht

In diesem Kapitel beschreibe ich die Ableitung eines Datenmodells ausgehend von einer Analyse der NMEA-Daten. Das bedeutet, es dreht sich alles darum wie z.B. die in einem Satz enthaltenen Positionsdaten mit entsprechenden Zielvariablen verknüpft werden und woher der Parser weiß ob und wie ein empfangener NMEA-Satz zu parsen ist.

3.2.1 Datenanalyse

Die grobe Struktur der NMEA-Sätze habe ich bereits in meinem letzten Artikel beschrieben. Auf diesem Aufbau setze ich bei der jetzigen Analyse auf. Sieht man sich einen NMEA-Satz an, so fällt auf, dass man an den ersten fünf Zeichen eines Satzes erkennen kann, welche Daten dieser enthält und damit auch ob er verarbeitet werden muss. Weiterhin ist bekannt, dass die einzelnen Datenfelder durch Kommas getrennt sind und das jeder Satz im Puffer mit dem \0-Zeichen abgeschlossen wird. Diese Merkmale können für den prinzipiellen Parserprozess verwendet werden. Aus der Dokumentation des NMEA-Protokolls im Datenblatt des GPS-Empfängers kann man weiterhin erkennen, dass die Daten in insgesamt acht Datentypen unterteilt werden können. Das sind:

- Status
- Latitude
- Longitude
- Time
- Variable
- Fixed HEX
- Fixed Alpha
- Fixed Number

Hieraus lässt sich ableiten, dass es nicht notwendig ist, jedes Datenfeld für sich zu betrachten. Stattdessen ist es nur notwendig die verschiedenen Datentypen zu unterscheiden und für jeden Typen eine Behandlungsfunktion zu implementieren. Diese Funktionen können die Schnittstelle zwischen einem NMEA Datenfeld (aus dem Data Link Layer) und einer beliebigen Zielvariable entsprechenden Typs darstellen.

3.2.2 Datenmodell

Nach der Analyse kann nun das entsprechende Datenmodell entworfen werden. D.h. es geht nun um die eigentliche Interpretation der Datenfelder und die Konfiguration der zu unterstützenden NMEA-Sätze.

3.2.2.1 NMEA-Datentypen

Wie ich im Abschnitt Analyse schon beschrieben habe, können die NMEA-Daten durch acht verschiedene Datentypen abgebildet werden (evtl. sind auch weniger Typen möglich). Für jeden dieser Datentypen wird eine Konvertierungsfunktion implementiert welche die Daten einliest und in das entsprechende Zielformat umwandelt. Das Zielformat ist hierbei von der jew. Anwendung abhängig. Da das eigentliche Parsermodul die verschiedenen Datentypen nicht kennen soll, muss ein einheitlicher Funktionstyp definiert werden, der für alle Konvertierungsfunktionen gleich ist. Diesen Funktionstyp habe ich wie folgt festgelegt:

```
typedef uchar8 tNMEADataConverter(uchar8 *pucData_, void *pTargetVar_);
```

Alle Konvertierungsfunktionen besitzen dieses Format. Als Übergabeparameter gibt es einen Zeiger auf das jew. Datenfeld im NMEA-Satz und zum anderen einen Zeiger auf eine Zielvariable. Da grundsätzlich nicht bekannt ist welche Datentypen zu unterstützen sind, ist der Zeiger auf die Zielvariable als void-Zeiger definiert. Als Rückgabetyp habe ich einen unsigned char Typ festgelegt. Damit wäre es später auch möglich Fehlerzustände zu signalisieren.

Da das Parsermodul nicht weiß wie viele Datentypen zu unterstützen sind, werden alle Konvertierungsfunktionen in einer Tabelle abgelegt. Diese Tabelle enthält einen Zeiger auf jede einzelne Funktion. Der Zugriff erfolgt über einen gewöhnlichen Feldzugriff. Die Indexe der einzelnen Datentypen sind mit Hilfe einer Enumeration festgelegt. Das bedeutet, dass die einzelnen Enumerationselemente die Namen der Datentypen festlegen. Diese Namen werden später in der Konfigurationstabelle des Parsers statt der Funktionszeiger verwendet.
Nachdem die einzelnen Datentypen nun definiert sind müssen die Typen noch mit den Zielvariablen verbunden werden können. Um diese Verbindung zu ermöglichen habe ich eine spezielle Struktur angelegt welche dies ermöglicht. Die Definition sie wie folgt aus:

```
typedef struct NMEADataConnector
{
        tNMEADataType     eDataType;    /**< Connection to the converter function */
        void*             pTargetVar;   /**< Connection to the target variable */
}tNMEADataConnector;
```

Es ist erkennbar, dass jede Verbindung durch einen Datentyp und einen Zeiger auf die jew. Zielvariable beschrieben werden kann.

3.2.2.2 Konfigurationstabellen

Wie schon beschrieben erfolgt die Konfiguration der vom Parser zu unterstützenden NMEA-Sätze in der Datei ‚nmeaframecfg.c'. In dieser Datei sind verschiedene Tabellen definiert welche die zu unterstützenden NMEA-Sätze im Detail beschreiben.

Wie ich in der Datenanalyse schon beschrieben habe, kann jeder NMEA-Satz durch einen Identifier und eine variable Anzahl von Datenfeldern mit verschiedenen Datentypen beschrieben werden. Hieraus habe ich die folgende Struktur abgeleitet um die NMEA-Sätze zu beschreiben:

```
typedef struct NMEAFrameFormat
{
      uchar8                  ucMSG[5];
      uchar8                  ucNoOfDataFields;
      tNMEADataConnector      sDataConnection[];
}tNMEAFrameFormat;
```

Es ist erkennbar, dass zunächst der Identifier der NMEA-Datensätze codiert wird. Dann folgt die Codierung der Anzahl der im jew. Satz enthaltenen Datenfelder und eine flexible Anzahl von Verbindungsstrukturen welche die Zielvariablen mit den Konvertierungsfunktionen verknüpfen. Die Anzahl der definierten Verbindungsstrukturen muss dabei mit der Anzahl der zu erwartenden Datenfelder übereinstimmen. Die Konfiguration des NMEA-Satzes „GPGGA“ könnte damit wie folgt aussehen:

```
const tNMEAFrameFormat NCFG_sNMEAFrame_GGA PROGMEM =
{
      "GPGGA",
      14,
      {
            { NCONV_TypeTime,       DBG_ucTime },
            { NCONV_TypeLatitude,   DBG_ucLatitude.ucLatitude},
            { NCONV_TypeStatus,     DBG_ucLatitude.ucNS},
            { NCONV_TypeLongitude,  DBG_ucLongitude.ucLongitude},
            { NCONV_TypeStatus,     DBG_ucLongitude.ucEW},
            { NCONV_TypeUndefined,  NULL },
            { NCONV_TypeUndefined,  NULL },
            { NCONV_TypeUndefined,  NULL },
            { NCONV_TypeUndefined,  NULL },
            { NCONV_TypeUndefined,  NULL },
            { NCONV_TypeUndefined,  NULL },
            { NCONV_TypeUndefined,  NULL },
            { NCONV_TypeUndefined,  NULL },
            { NCONV_TypeUndefined,  NULL }
      }
};
```

Um die einzelnen NMEA-Konfigurationen dem zentralen Parsermodul bekannt zu machen werden diese in einer Tabelle zusammengefasst welche dem Parsermodul bekannt ist. Diese Tabelle enthält jew. einen Zeiger auf jede Konfigurationsstruktur.

3.2.2.3 Abschlussübersicht

In den folgenden beiden Abbildungen ist sind die grundlegenden Datenstrukturen dargestellt:

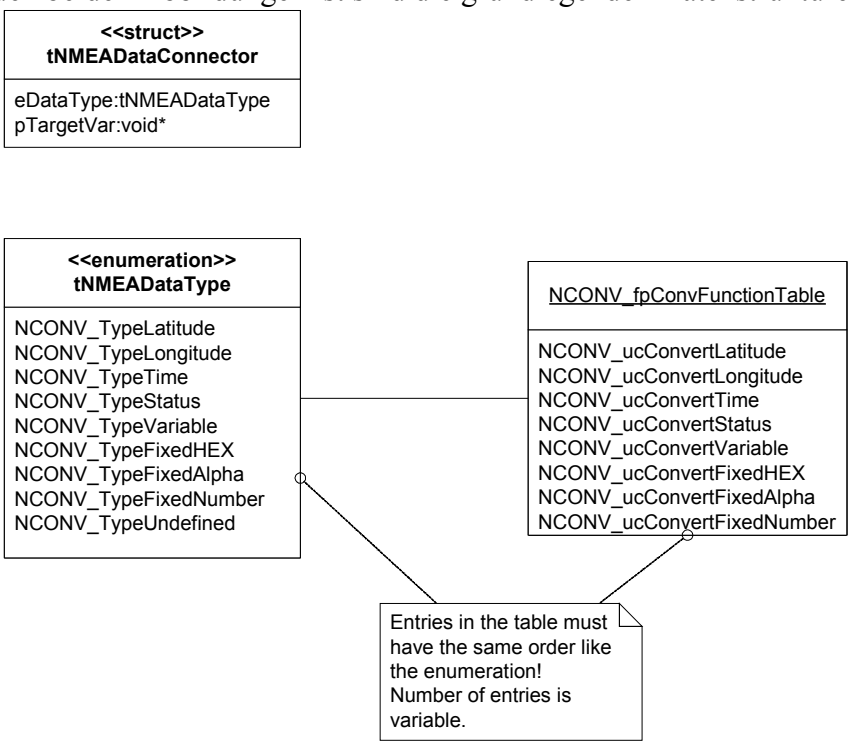

Abbildung 3-2: Aufbau der NMEA Datentypen

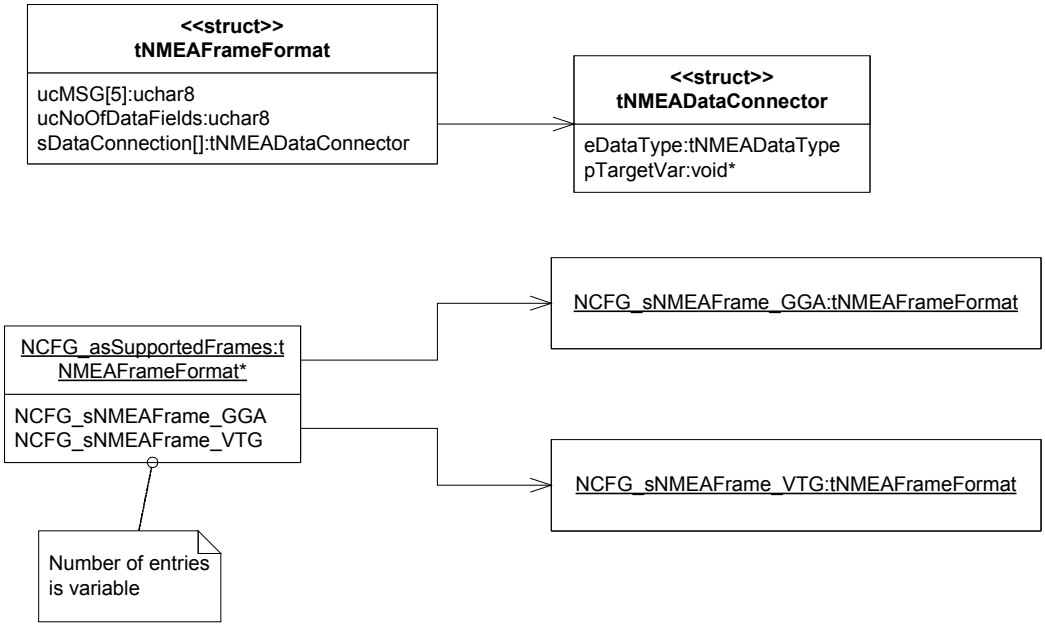

Abbildung 3-3: Aufbau der NMEA Konfigurationsstruktur

3.3 Laufzeitsicht

Nachdem nun der statische Aufbau, sowie die Datenstrukturen des NMEA Parsers beschrieben sind, geht es nun um die Beschreibung der eigentlichen Parserfunktion. D.h. ich werde beschreiben wie das Parsermodul zur Laufzeit genau vorgeht.

Das Parsermodul besitzt als zentrale Funktion einen Taskhandler. Dieser Handler muss zyklisch in der Mainloop der Software aufgerufen werden. Nach dem Aufruf prüft diese Funkti-

on zunächst ob NMEA-Daten vom Data Link Layer empfangen wurden. Sollte dies der Fall sein, so kann ein Parserprozess gestartet werden. D.h., dass der empfangene NMEA-Satz zunächst beim Data Link Layer abgeholt wird. Danach wird überprüft ob der empfangene Satz überhaupt zu parsen ist. Dies erfolgt durch Vergleich des empfangenen NMEA Identifiers mit den Einträgen in der Konfigurationstabelle. Sollte der empfangene Identifier gefunden werden, so wird der Satz verarbeitet und abschließend aus dem DLL Empfangspuffer entfernt. Sollte kein entsprechender Eintrag gefunden werden so wird der Satz gleich verworfen. Nachdem klar ist, dass ein NMEA-Satz verarbeitet werden soll, wird zunächst noch eine Plausibilitätsprüfung durchgeführt. Bei dieser Prüfung wird die Anzahl der Datenfelder im empfangenen NMEA-Satz mit dem Eintrag in der Konfigurationstabelle verglichen. Sollte sich hier ein Unterschied ergeben, so wird davon ausgegangen, dass die NMEA-Daten ungültig sind und der Satz wird verworfen. Im anderen Fall wird der Parserprozess gestartet.

Der eigentliche Parserprozess ist relativ einfach aufgebaut. Die Parserfunktion durchläuft den gesamten empfangenen NMEA-Satz bis zum abschließenden \0-Zeichen. Immer dann wenn ein Komma gefunden wird beginnt ein neues Datenfeld. An diesem Punkt entnimmt die Parserfunktion einen Zeiger auf die Datentypfunktion sowie auf die Zielvariable aus den Konfigurationstabellen und ruft die Typfunktion mit einem Zeiger auf die aktuelle Stelle im NMEA-Satz und auf die Zielvariable auf. Die Datentypfunktion übernimmt dann die Umsetzung der Daten. Ist der komplette NMEA-Satz durchaufen, so endet der Parserprozess.

4 Verwendung der Software

Wie ich schon angedeutet habe, kann die NMEA Parsersoftware frei konfiguriert werden. Dies erfolgt mit Hilfe der schon beschriebenen Datenstrukturen. Die Konfiguration der Software lässt sich in zwei Hauptpunkte unterteilen:

- Anlegen eines neuen Datentyps
- Anlegen und Bearbeiten von unterstützten NMEA-Sätzen

4.1 Anlegen eines neuen Datentyps

Soll ein neuer Datentyp unterstützt werden so sind nur die beiden Dateien nmeaconv.c und nmeaconv.h zu modifizieren. Hierbei ist zunächst eine neue Funktion anzulegen, welche dem oben definierten Typ entspricht. Für den Datentyp Latitude sieht die entsprechende Funktion z.B. wie folgt aus.

```
uchar8 NCONV_ucConvertLatitude(uchar8 *pucData_, void *pTargetVar_)
```

Nach der Implementierung der Funktionalität der Funktion kann diese in der Datentypentabelle ‚NCONV_fpConvFunctionTable' (nmeaconv.c) eingetragen werden. Danach muss noch die Enumeration , tNMEADataType' (nmeaconv.h) ergänzt werden. Bei der Ergänzung ist es sehr wichtig, dass die Enumeration an der gleichen Stelle ergänzt wird wie die Funktionszeigertabelle da die Enumerationen als Index in der Zeigertabelle dienen!

Damit ist ein neuer Datentyp auch schon angelegt und kann in der Konfiguration der NMEA-Sätze verwendet werden.

4.2 Anlegen und Bearbeiten von unterstützten NMEA-Sätzen

Soll ein neuer NMEA-Satz unterstützt werden, so ist zunächst eine neue Struktur vom Typ , tNMEAFrameFormat' in der Datei ‚nmeaframecfg.c' anzulegen. Diese Struktur muss im Programmspeicher des Atmel Controllers angelegt werden. Für den NMEA-Satz ‚GPGGA' habe ich bereits ein Beispiel im Kapitel 3.2.2.2 dargestellt. Die Konfiguration beginnt mit der Festlegung des Identifiers, im Beispiel wäre das „GPGGA". Anschließend wird die Anzahl der zu erwartenden Datenfelder konfiguriert. Für den Satz GPGGA sind 14 Datenfelder zu

erwarten. Daran anschließend muss für jedes Datenfeld ein Datentyp und eine Zielvariable festgelegt werden. Bei 14 Datenfeldern sind damit auch 14 Einträge zu konfigurieren. Die Reihenfolge der entsprechenden Einträge entspricht genau der Reihenfolge wie sie im NMEA-Satz auftreten. Im Beispiel hat eine Konfiguration z.B. folgendes Aussehen:

```
{ NCONV_TypeTime,       DBG_ucTime }
```

Was sagt nun diese Einstellung aus? Nun, zunächst wird mit „NCONV_TypeTime“ der Datentyp festgelegt. Dies ist ein Eintrag aus der schon bekannten Enumeration in der Datei ‚nmeaconv.h’ und bewirkt, dass dieses Datenfeld mir der Konvertierungsfunktion „uchar8 NCONV_ucConvertTime(uchar8 *pucData_, void *pTargetVar_)“ verknüpft wird. Bei dem Eintrag „DBG_ucTime“ handelt es sich um einen Zeiger auf die Zielvariable. Das bedeutet, dass die Funktion NCONV_ucConvertTime die Im Datenfeld enthaltene Information in der Variablen DBG_ucTime abspeichert.

Eine Besonderheit ergibt sich für Datenfelder welche nicht benötigt werden. Diese Felder sind mit folgender Einstellung zu berücksichtigen:

```
{ NCONV_TypeUndefined, NULL }
```

Ist der NMEA-Satz komplett konfiguriert so ist dieser abschließend in der Tabelle „NCFG_asSupportedFrames“ (nmeaframecfg.c) einzutragen. Dieser Eintrag ist in Form eines Zeigers vorzunehmen. Für o.g. NMEA-Satz sieht der Eintrag wie folgt aus (rot):

```
const tNMEAFrameFormat *NCFG_asSupportedFrames [] PROGMEM =
{
      &NCFG_sNMEAFrame_GGA,
      &NCFG_sNMEAFrame_VTG
};
```

Damit der Parser dann auch noch weiß, dass sich die Anzahl der zu unterstützenden NMEA-Sätze geändert hat ist zu aller letzt das Literal „NCFG_SUPPORTED_FRAME_NO“ in der Datei ‚nmeaframecfg.h’ um eins zu erhöhen.

Damit ist dann die Erweiterung abgeschlossen…

5 Quellenangaben

Abschließen möchte ich diesen Artikel mit ein paar interessanten Quellen zum Thema GPS, NMEA und allem was dazugehört:

- http://www.kowoma.de/gps/
- http://www.gs-enduro.de/