DIPLOMARBEIT

Fehlverhalten von GPS-Satellitenempfängern

im Hinblick auf ihren Einsatz

im Forschungsprojekt SynUTC

verfaßt
am Institut für Automation
der Technischen Universität Wien

unter Anleitung von Univ. Doz. Dr. Ulrich Schmid
von
Dieter Höchtl

Dieter Höchtl
Am Kaisermühlendamm 85/21
A-1220 Wien

20. September 1996

Dieter Höchtl:
Diplomarbeit über das Fehlverhalten von GPS-Satellitenempfängern im Hinblick auf Ihren Einsatz im Forschungsprojekt SynUTC
Diplomarbeit am Institut für Automation (E183/1)
der Technischen Universität Wien
vom 20. September 1996

Diese Diplomarbeit wurde unter $\LaTeX\,2_\varepsilon$ mit deutschen Anpassungen gesetzt und mit dvips nach Postscript konvertiert.

Abstract

In January 1995 the research project *SynUTC* was started at the Departement of Automation at the Technical University of Vienna. This project is dedicated to fault-tolerant synchronization of computer-clocks in distributed real time systems. In contrast to former approaches which established only a global time base without any relationship to international accepted time scales (like UTC [Universal Time Coordinated]), the synchronicity to such time scales is a major issue in SynUTC.

This approach certainly needs an efficient access to the time scale. A highly attractive mean w.r.t. accuracy and availability is the satellite navigation system NavSTAR GPS (Navigation Satellite Timing and Ranging Global Positioning System), which provides worldwide and continously accurate position and time (accuracy depending on expense from 10 ns to 1 μs) relative to the time scale UTC[1]. The availability of cheap GPS-Receivers (some 1000 ATS) finally justifies why the GPS is considered the primary means for accessing the international time-scale UTC.

Fault-tolerance — a major issue in the project — requests a deeper study of the potential failure modes of GPS receivers. For this purpose 6 different GPS receivers (prices from 5,000 ATS to 60,000 ATS) were evaluated in a 2 months experiment. Among other things, it was interesting to find out whether the often propagated opinion that the time is provided by GPS receivers is highly reliable, is true. Surprisingly it turned out that only 3 of the 6 receivers worked satisfactory. Several different failure modes, including sporadic misbehaviour or an extremly offlying time signal (some μs), were observed for the 3 other receivers.

[1] Actually only relative to the real-time representation of UTC, namely UTC(USNO, MC) which is realized by the master clock of the *United States Naval Observatory (USNO)*.

Zusammenfassung

Im Jänner 1995 wurde am Institut für Automation der Technischen Universität Wien das Forschungsprojekt *SynUTC* gestartet. Dieses Projekt widmet sich der fehlertoleranten Synchronisation von Computeruhren in verteilten Echtzeitsystemen. Im Gegensatz zu früheren Ansätzen, in denen die durch geeignete Synchronisationsalgorithmen generierte Zeitbasis keinerlei Bezug zu einer international anerkannten Zeitskala (wie etwa UTC [Universal Time Coordinated, koordinierte Weltzeit]) hatte, ist die Synchronizität zu einer solchen Zeitskala grundsätzlicher Bestandteil des Projeks SynUTC.

Ein solcher Ansatz benötigt ein effizientes Mittel zur Reproduktion dieser internationalen Zeitskala. Ein betreffend Genauigkeit und Verfügbarkeit sehr attraktiv erscheinendes Mittel ist das Satellitennavigationssystem NavSTAR GPS, das weltweit und zu jeder Tageszeit, neben einer genauen Positionsbestimmung auch die genaue Zeitbestimmung (Genauigkeit je nach Aufwand einige 10 ns bis 1 μs) in Bezug auf die Zeitskala UTC[2] ermöglicht. Der weiters relativ günstige Preis von GPS-Satellitenempfängern (einige wenige 1000 ÖS) ist ein Grund dafür, daß im Projekt SynUTC das GPS-System als primäres Mittel für den Zugang zur internationalen UTC-Zeitskala angesehen wird.

Für die im Projekt geforderte Fehlertoleranz war es daher notwendig, sich über das mögliche Fehlverhalten von GPS-Empfängern Klarheit zu verschaffen. Zu diesem Zweck, wurde im Rahmen dieser Diplomarbeit eine 2 Monate dauernde, kontinuierliche experimentelle Evaluation von 6 verschiedenen GPS-Empfängern (mit Marktpreisen zwischen 5.000 ÖS und 60.000 ÖS) durchgeführt. Es galt unter anderem, herauszufinden, ob die öfters geäußerte Meinung, daß die von GPS-Empfängern gelieferte Zeit sehr zuverlässig sei, sich bewahrheiten würde. Überraschenderweise konnte im Rahmen dieser Evaluation festgestellt werden, daß nur 3 der 6 Empfänger zufriedenstellend funktionierten. Die restlichen 3 zeigten verschiedene Formen des Fehlverhaltens, von falschen Sekundenpulsen bis zu gravierenden „systematischen" Ablagen (mehrere μs) von der zu reproduzierenden Zeitskala.

[2] Eigentlich nur bezüglich einer Echzeitdarstellung derselben, nämlich UTC(USNO,MC), die durch die Hauptuhr des *United States Naval Observatory (USNO)* definiert wird.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Teil I

Einleitung

1 Einleitung

Gegenstand der vorliegenden Diplomarbeit ist die Untersuchung des potentiellen Fehlverhaltens des Satellitensystems *NavSTAR GPS* und der dazu kommerziell erhältlichen GPS-Empfänger für den Einsatz im Forschungsprojekt *SynUTC*. Beginnend mit einer kurzen Einführung in dieses Projekt werden im folgenden die im Zusammenhang mit dem *NavSTAR GPS* auftretenden Fragestellung besprochen. Im Anschluß daran sind die konkreten Aufgabenstellungen dieser Diplomarbeit dargestellt.

1.1 Projekt SynUTC

1.1.1 Allgemeines

SynUTC (Synchronized UTC) ist ein im Jänner 1995 am Institut für Automation der Technischen Universität Wien gestartetes Forschungsprojekt, das sich der Entwicklung eines Systems zur hochgenauen, fehlertoleranten Synchronisation lokaler Computeruhren mit der *koordinierten Weltzeit (Universal Time Coordinated, UTC)* in großräumig *verteilten Echtzeitsystemen (distributed real-time systems)* widmet.

Im Gegensatz zum Problem der *internen* Synchronisation eines verteilten Echtzeitsystems zielt das Projekt SynUTC auf das Problem der *externen* Synchronisation ab. Es ist hierbei nicht nur die Synchronisation aller lokalen Uhren untereinander von Interesse, sondern auch die Synchronisation dieser Uhren mit der *koordinierten Weltzeit (UTC)* [1]. Ziel des Ganzen ist, daß die lokalen Uhren UTC mit einer gewissen Genauigkeit repräsentieren. Dadurch ist es erst möglich, die Zeitinformation extern datierter Ereignisse *intern* bzw. intern datierter Ereignisse *extern* nutzen und verwerten zu können.

Es hat sich erwiesen, daß — sofern Fehlertoleranz gefordert wird — die für die interne Synchronisation eines Netzwerks entwickelten Konzepte nicht ohne weiteres auf die externe Synchronisation übertragen werden können (siehe dazu Schmid [1994]). Das Projekt SynUTC verfolgt daher als eines der Ziele die Entwicklung geeigneter Konzepte und Algorithmen für die fehlertolerante externe Synchronisation. Daneben soll auch ein funktionierendes System in Form eines Prototyps entwickelt werden, dessen grober Aufbau im folgenden beschrieben sei:

[1] Die Bedeutung von UTC liegt nicht nur in ihrer globalen Anerkennung. (Sie ist in fast allen Ländern Grundlage der gesetzlichen Zeit.) Auch ihr Skalenmaß ist, mit der zum jeweiligen Stand der Technik erreichbaren Genauigkeit, gleich der gesetzlich definierten Sekunde. Dies deshalb, da UTC ein Hybrid der Internationalen Atomzeit TAI und der für das bürgerliche Leben wichtigen Weltzeit UT ist.

1.1.2 Aufbau des Systems

Wie aus Abbildung 1.1 ersichtlich, besteht das System aus mehreren *Synchronisationssubnetzen (SSN)*, die über sogenannte *Gatewayknoten* miteinander verbunden sind.

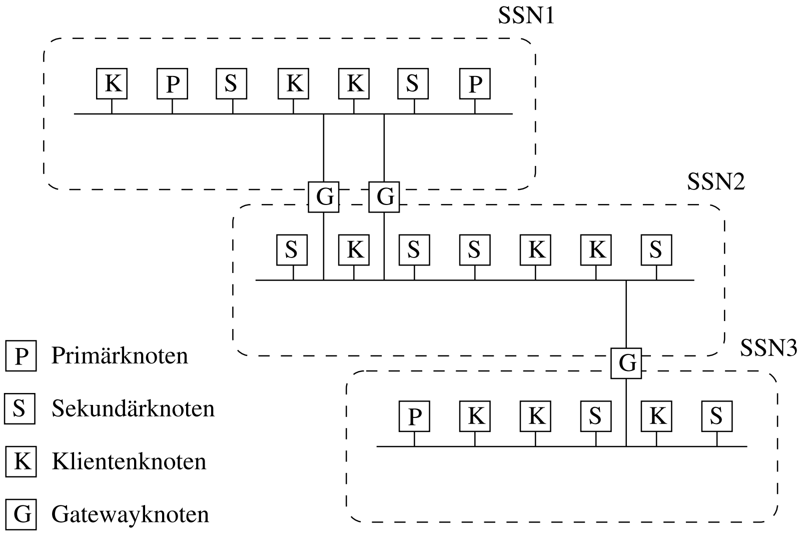

Abbildung 1.1: Modell des Kommunikationsnetzwerks in SynUTC

Die in einem solchen Netzwerk auftretenden Knoten — diese repräsentieren die lokalen Rechner bzw. Uhren — können von folgender Art sein:

- **Gatewayknoten.** Deren Aufgabe ist es, einzelne Synchronisationssubnetze miteinander zu verbinden.
- **Primärknoten.** Dies sind Knoten, die eine direkte Verbindung mit einer der UTC-Quellen besitzen.
- **Sekundärknoten.** Knoten dieser Art besitzen keine Verbindung zu einer UTC-Quelle, nehmen aber mit ihren lokalen Uhren in aktiver Weise am Synchronisationsverfahren teil.
- **Klientenknoten.** Diese Knoten hören Nachrichten am Kommunikationsnetzwerk ab, um sich auf diese einfache Weise mit der globalen Zeit zu synchronisieren.

Mit einem geeigneten Algorithmus soll nun eine globale Zeit eingerichtet werden, die folgende wichtige Eigenschaften besitzt:

- Fehlertolerant gegenüber Kommunikationsfehlern im Netzwerk
- Fehlertolerant gegenüber Knotenausfällen
- Fehlertolerant gegenüber Fehlverhalten von UTC-Quellen (Primärknoten)
- Synchron zu UTC (über die Primärknoten)

Die Anbindung an die UTC-Zeitskala erfolgt ausschließlich über die Primärknoten. Diese unterscheiden sich von den Sekundärknoten lediglich durch deren Kopplung mit einer UTC-Quelle.

Anhand von Abbildung 1.2 soll der Aufbau eines Knotens im Kommunikationsnetzwerk näher beschrieben werden: Im Zentrum steht die sogenannte *UTCSU* (*UTC Synchronization Unit*), die für die Zeithaltung zu sorgen hat. Über einen Systembus ist sie mit der *CPU*, dem lokalen Speicher und der Kommunikationseinheit *COMCO (Communication Coprozessor)* des Knotens verbunden. Die Kommunikationseinheit ermöglicht den Zugang zum Netzwerk, über das die Nachrichten, die zur Synchronisation der Uhren in verschiedenen Knoten erforderlich sind, übermittelt werden können.

Steht eine UTC-Quelle — etwa ein GPS-Empfänger — zur Verfügung, so kann ein Knoten auch als Primärknoten betrieben werden. Für diesen Fall muß die UTCSU geeignete Schnittstellen zur Verarbeitung der Zeitsignale einer solchen Quelle zur Verfügung stellen. Die in der UTCSU gehaltene Zeit soll weiters über standardisierte Schnittstellen *(NTPA-Bus)* nach außen geliefert werden und den auf dem Knoten laufenden Anwendungen effiziente Zugriffsmöglichkeiten auf die lokale Zeitskala ermöglichen. Die Spezifikation der UTCSU ist in Schossmaier und Schmid [1995] zu finden.

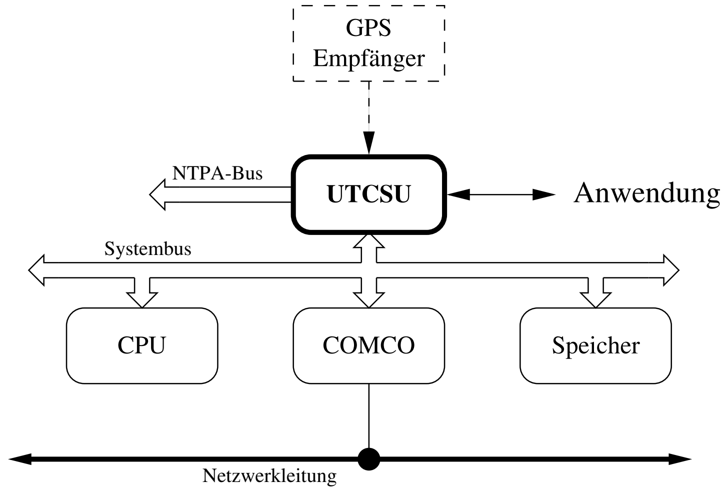

Abbildung 1.2: Detailaufbau eines Knotens in SynUTC

1.1.3 Die UTC-Quelle NavSTAR GPS

Für das Projekt SynUTC ist ein geeignetes Mittel zur Synchronisation mit UTC erforderlich. Für solche Zwecke wurden Zeitzeichensender installiert, die für ein beschränktes Sendegebiet die genaue Zeit per Funkaussendungen zur Verfügung stellen, etwa der Zeitzeichensender DCF77 in Mainflingen, Deutschland. Für SynUTC reichen aber sowohl die Genauigkeit im Millisekunden-Bereich als auch die Verfügbarkeit (nur in Europa zu empfangen, Abschaltung bei Gewittern im Sendergebiet) dieser Aussendungen nicht aus.

Stattdessen bietet sich das vom amerikanischen Verteidigungsministerium *(Department of Defense, DoD)* in Auftrag gegebene und seit 1995 voll funktionsfähige *NavSTAR GPS (Navigation Satellite Timing And Ranging Global Positioning System)* an. Dieses anfänglich lediglich für militärische Zwecke entwickelte Satellitennavigationssystem ermöglicht eine genaue Positions– und Geschwindigkeitsbestimmung durch Empfang von kodierten Satellitensignalen. GPS ist auch zivilen Nutzern zugänglich, wenngleich auch nur mit eingeschränkter Genauigkeit. (Dies wird durch die sogenannte *Selective Availability, SA* erreicht.) Als Nebenprodukt der Navigationsanwendung ergibt sich die Möglichkeit des hochgenauen Zeitvergleichs (Genauigkeit einige ns) und der Zeitsynchronisation (Genauigkeit unterschiedlich, etwa 1 μs – 100 ns), und zwar mittels äußerst kostengünstiger GPS-Empfänger.

Der große Vorteil des GPS-Systems (ständige Verfügbarkeit, weltweit, bei jedem Wetter und zu jeder Tageszeit) darf aber nicht darüber hinwegtäuschen, daß für den Einsatz in Echtzeitsystemen — deren Versagen katastrophale Folgen nach sich ziehen könnten — Zuverlässigkeitsuntersuchungen des ganzen GPS-Systems notwendig sind.

1.2 Aufgabenstellung

Die Aufgabenstellung der Diplomarbeit umfaßt drei Teile:

- **Darstellung der theoretischen Grundlagen.** Zur Erleichterung des Einstiegs in den Themenbereich des NavSTAR GPS soll ein theoretischer Teil, der all das nötige Wissen überblicksmäßig darstellt, den Anfang der Diplomarbeit bilden.
- **Marktstudie.** Zur Untersuchung des potentiellen Fehlverhaltens des GPS ist eine Marktstudie durchzuführen, um geeignete Empfänger für eine experimentelle Evaluation auszuwählen, die dann mit Projektmitteln angekauft werden sollen. Die dabei gewonnenen Informationen und Einsichten sollen darüberhinaus zur Definition einer universellen Schnittstelle zu den GPS-Empfängern führen.
- **Experimentelle Evaluation.** Den letzten, wesentlichsten Teil der Diplomarbeit bildet die Planung, Durchführung und Auswertung einer experimentellen Evaluation des Fehlverhaltens der angekauften GPS-Empänger hinsichtlich der Anforderung des Projekts SynUTC.

Teil II

Theoretische Grundlagen

2 Zeit

2.1 Der physikalische Begriff der Zeit

Die Erforschung der objektiven Zeit ist eng mit der Entwicklung der Astronomie, der modernen theoretischen Physik und der Uhr als Zeitmesser verbunden. Wurde in der klassischen Physik (NEWTONsche Mechanik) noch davon ausgegangen, daß es eine *absolute Zeit* gäbe, so ist die Physik seit EINSTEINs *Relativitätstheorie* zur Erkenntnis gelangt, daß diese intuitive Annahme falsch ist. Denn das Fortschreiten der Zeit hängt wesentlich vom Bewegungszustand der Meßuhr und dem Gravitationsfeld, in dem sie sich befindet, ab. Obwohl diese Erkenntnis für unser Alltagsleben völlig belanglos ist[1], so gibt es dennoch einige Bereiche, in denen *relativistische Effekte* technisch relevant sind, insbesondere bei der Erzeugung der wichtigsten Zeitskalen (z.B. Internationale Atomzeit TAI) und bei Satellitennavigationssystemen, wie dem NavSTAR GPS.

Im folgenden soll daher ein Überblick über Begriffe und Ergebnisse der Relativitätstheorie dargestellt werden (siehe Fließbach [1995]):

2.1.1 Vorbereitende Begriffe

Bezugssysteme

Will man physikalische Vorgänge beschreiben, so benötigt man ein *Bezugssystem*. Ein Bezugssystem S läßt sich durch die Wahl eines Ursprungs $\mathbf{O}$ und dreier Koordinatenachsen $\mathbf{e}_i$ festlegen. Ein bestimmter Ort $\mathbf{x}$ wird dann durch die drei Koordinaten (x, y, z) beschrieben:

$$\mathbf{x} = \mathbf{O} + x\mathbf{e}_1 + y\mathbf{e}_2 + z\mathbf{e}_3. \tag{2.1}$$

In der Relativitätstheorie kann man Ort und Zeit nicht mehr unabhängig voneinander betrachten, deshalb fügt man zu den drei Raumkoordinaten noch die Zeitkoordinate hinzu. Dadurch werden nicht mehr Orte, sondern Ereignisse betrachtet. Ein Ereignis E wird dann in einem bestimmten Bezugssystem S durch die vier Koordinaten (x, y, z, t) fesgelegt: Am Ort (x, y, z) trat zum Zeitpunkt t das Ereignis E ein. Das physikalische Geschehen wird damit durch eine Menge von Ereignissen in einem 4-dimensionalen Raumzeitgebilde, dem *Minkowskiraum* beschrieben.

[1] Im alltäglichen Leben sind die erforderlichen Geschwindigkeiten und Gravitationspotentialdifferenzen viel zu gering um merkliche Effekte verursachen zu können.

Inertialsysteme

Inertialsysteme sind Bezugssysteme, die dadurch ausgezeichnet sind, daß in ihnen keine Trägheitskräfte wirken. Das heißt: In ihnen bleiben kräftefreie Massepunkte in Ruhe oder bewegen sich geradlinig und gleichförmig (unbeschleunigt). Aus den Newtonschen Bewegungsgleichungen folgt, daß jedes Bezugssystem genau dann ein Inertialsystem ist, wenn es sich geradlinig und gleichförmig relativ zu einem anderen Inertialsystem bewegt. Hat man daher ein Inertialsystem gefunden, so sind alle anderen Inertialsystem relativ zu diesem gleichförmig bewegt.

Im Experiment stellt man fest, daß genau jene Bezugssysteme Inertialsystem sind, die relativ zum Fixsternhimmel gleichförmig bewegt sind. Der Massenmittelpunkt unseres Sonnensystems stellt damit eine sehr gute Näherung für ein Inertialsystem dar (wobei man hier die Beschleunigung der Sonne bei der Bewegung um den Mittelpunkt unserer Galaxie vernachlässigen muß), sofern die Koordinatenachsen gegenüber dem Fixsternhimmel ruhen. Auch der Massenmittelpunkt der Erde kann noch als Ausgangspunkt für ein gutes Inertialsystem dienen. Verwendet man allerdings ein Bezugssystem, das fest mit der Erde verbunden ist (also mit dem Erdkörper gegen den Fixsternhimmel mitrotiert), so treten bereits beträchtliche Trägheitskräfe, wie Zentrifugalkräfte und Corioliskräfte (sichtbar in Tiefdruckwirbeln) auf, die jedoch gewöhnlich von der Gravitationskraft überdeckt werden.

Galileitransformation

Sind einmal geeignete Inertialsysteme gefunden worden, so stellt sich die Frage, wie sich die in einem Inertialsystem IS bestimmten Koordinaten in einem Inertialsystem IS$'$ darstellen. Zu diesem Zweck müssen die Koordinaten geeignet transformiert werden. In der Newtonschen Mechanik wurde die sogenannte *Galileitransformation* als die korrekte erachtet. Um diese Transformation zu beschreiben, wollen wir der Einfachheit halber von relativen Drehungen und Translationen der Inertialsysteme absehen und ihnen parallele Koordinatenachsen verpassen. Außerdem sollen die Ursprünge zum Zeitpunkt $t = t' = 0$ zusammenfallen. Schließlich fordern wir noch, daß die Geschwindigkeit, mit der sich die beiden Inertialsysteme bewegen, entlang der x-Achse verläuft. Diese Einschränkungen erhalten die wesentlichen Aspekte der Transformation, erleichtern uns aber das mathematische Leben (siehe Abbildung 2.1; die Koordinatenachsen z und z' wurden einfach weggelassen).

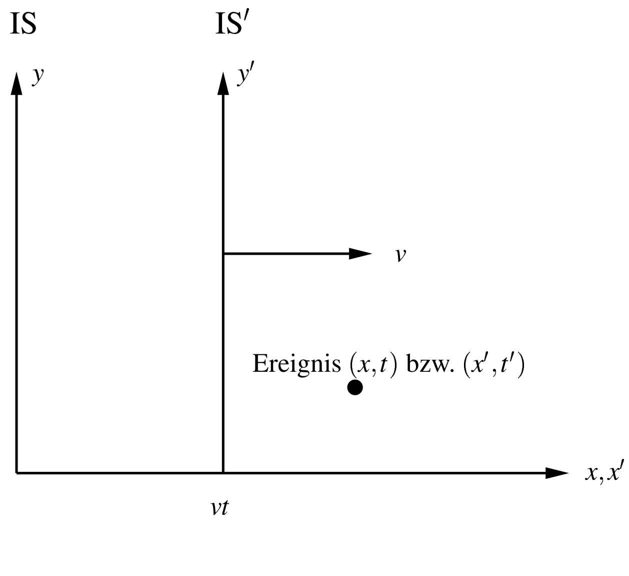

Abbildung 2.1: Transformation zwischen zwei Inertialsystemen IS und IS$'$. Die Koordinaten eines Ereignisses lautet in IS (x,t), im dazu mit v bewegten IS$'$ dagegen (x',t')

Die Koordinaten eines Ereignisses E in einem Inertialsystem IS (x,y,z,t) transformieren sich in das Inertialsystem IS$'$, das relativ zu IS mit der Geschwindigkeit v entlang der x-Achse bewegt ist, nach folgenden Gleichungen:

$$\begin{aligned} x' &= x - vt \\ y' &= y \\ z' &= z \\ t' &= t. \end{aligned}$$

Deutlich ist zu erkennen, daß die Zeitkoordinate t weder von den Ortskoordinaten, noch vom Bewegungszustand (Geschwindigkeit v) abhängt. Deshalb wurde auch der Begriff einer *absoluten Zeit* geprägt, die an allen Orten unter allen Bedingungen in gleicher Weise fortschreitet.

Erst EINSTEIN wagte es 1905, nach Vorarbeiten von LORENTZ, MACH und POINCARÉ, sich von dieser Vorstellung freizumachen, nachdem bereits 1887 MICHELSON und MORLEY ein Experiment durchführten, in dem Sie zu dem überraschenden Ergebnis gelangten, daß die Lichtgeschwindigkeit in verschiedenen Inertialsystem überall denselben Wert annimmt.

Dieses Experiment widersprach den Folgerungen aus den *Maxwellgleichungen* — die die Ausbreitung elektromagnetischer Wellen beschreiben — und der Galileitransformation, aus denen gemeinsam folgt, daß in verschieden bewegten Inertialsystemen die Lichtgeschwindigkeit verschieden sein müßte. EINSTEIN löste das Problem, indem er die Galileitransformation durch die Lorentztransformation ersetzte und dadurch den Begriff der Zeit relativierte.

Dieser erste Schritt (1905) führte zunächst zur *speziellen Relativitätstheorie*. Zur komplexeren *allgemeine Relativitätstheorie* mit entsprechend allgemeineren Transformationsgleichungen gelangte er durch Einbindung der Gravitationskräfte.

2.1.2 Spezielle Relativitätstheorie

Lorentztransformation

Die nunmehr korrekten relativistischen Transformationsgleichungen lauten, unter den gleichen Einschränkungen wie bei der Galileitransformation (siehe Abbildung 2.1),

$$\begin{aligned} x' &= \frac{x - vt}{\sqrt{1 - \frac{v^2}{c^2}}} \\ y' &= y \\ z' &= z \\ t' &= \frac{t - \frac{xv}{c^2}}{\sqrt{1 - \frac{v^2}{c^2}}}. \end{aligned}$$

Man beachte, daß die Lorentztransformation für $v \ll c$ zur Galileitransformation wird. Die Galileitransformation ist damit die erste Näherung für hinreichend kleine Geschwindigkeiten v.

Die neuen Transformationsgleichungen bewirken, daß das Fortschreiten der Zeit nun sowohl von den Raumkoordinaten als auch vom Bewegungszustand abhängt. Dies führt zu den der menschlichen Intuition zuwiderlaufenden Effekten der Relativitätstheorie:

Zeitdilatation

Bei der *Zeitdilatation* geht man von folgendem Szenario aus: Man betrachet ein Inertialsystem IS und stellt sich die Frage, wie schnell eine bewegte Uhr laufen wird. Dazu geht man in das Inertialsystem IS$'$ über, in dem die Uhr am Ort x' ruht. Nun kann man zwei Uhrenschläge als Ereignisse (x', t_1') und (x', t_2') identifizieren. Dies entspricht natürlich einer gemessenen Zeitspanne von $\Delta t' = t_2' - t_1'$. Transformiert man zurück ins System IS, so erhält man

$$t_1 = \frac{t_1' - \frac{x'v}{c^2}}{\sqrt{1 - \frac{v^2}{c^2}}}. \tag{2.2}$$

und

$$t_2 = \frac{t_2' - \frac{x'v}{c^2}}{\sqrt{1 - \frac{v^2}{c^2}}}. \tag{2.3}$$

In IS wird man also eine Zeitspanne von $\Delta t = t_2 - t_1$ messen:

$$\Delta t = \frac{\Delta t'}{\sqrt{1 - \frac{v^2}{c^2}}}. \tag{2.4}$$

Die bewegte Uhr in IS$'$ scheint also, wenn sie in IS gemessen wird, langsamer zu sein. Bemerkenswerterweise ist diese Aussage symmetrisch, denn ebenso wird ein Beobachter in IS$'$ feststellen, daß eine Uhr in IS um den entsprechenden Faktor verlangsamt ist. Diese Erkenntnis scheint anfangs verwirrend, ist aber Ergebnis unseres Widerwillens, einen absoluten Zeitbegriff zu verwerfen.

Längenkontraktion

Ähnlich wie bei der Zeitdilatation läßt sich eine analoge Beziehung für die Raumkoordinaten ableiten. Man betrachtet einen bewegten Längenmaßstab und würde gerne dessen Länge im Inertialsystem IS kennen. Neuerlich beginne man im Inertialsystem IS$'$, in dem der Maßstab ruhe. Der Anfang des Maßstabs befinde sich am Ort x_1', dessen Ende am Ort x_2'. Die Vermessung des Maßstabs in IS erfolgt durch gleichzeitiges Ablesen (zum Zeitpunkt t) der Koordinaten und Differenzbildung: $\Delta x = x_2 - x_1$. Die zum Ablesen gehörigen Ereignisse haben in IS$'$ die Koordinaten (x_1', t_1') und (x_2', t_2'). Die Zeitpunkte t_1', t_2' sind vorerst unbekannt und ergeben sich aus der Bedingung, daß das Ablesen gleichzeitigt erfolgt ($t_1 = t_2 = t$):

$$t = t_1 = \frac{t_1' - \frac{x_1'v}{c^2}}{\sqrt{1 - \frac{v^2}{c^2}}}. \tag{2.5}$$

und

$$t = t_2 = \frac{t_2' - \frac{x_2'v}{c^2}}{\sqrt{1 - \frac{v^2}{c^2}}}. \tag{2.6}$$

Dies führt auf die Bedingung

$$t_2' - t_1' = \frac{v}{c^2}(x_2' - x_1'). \tag{2.7}$$

Die Transformationsformeln für die Orte ergeben

$$x_1 = \frac{x_1' - vt_1'}{\sqrt{1 - \frac{v^2}{c^2}}}. \tag{2.8}$$

und

$$x_2 = \frac{x_2' - vt_2'}{\sqrt{1 - \frac{v^2}{c^2}}}. \tag{2.9}$$

Die Differenz $\Delta x = x_2 - x_1$ liefert unter Berücksichtigung obiger Bedingung für t_1' und t_2'

$$\Delta x = \frac{(x_2' - x_1')(1 - \frac{v^2}{c^2})}{\sqrt{1 - \frac{v^2}{c^2}}} = \Delta x' \sqrt{1 - \frac{v^2}{c^2}}. \tag{2.10}$$

Die in IS gemessene Länge ist also kleiner als im ruhenden System. Diese Verkürzung nennt man *Längenkontraktion*.

Dopplereffekt

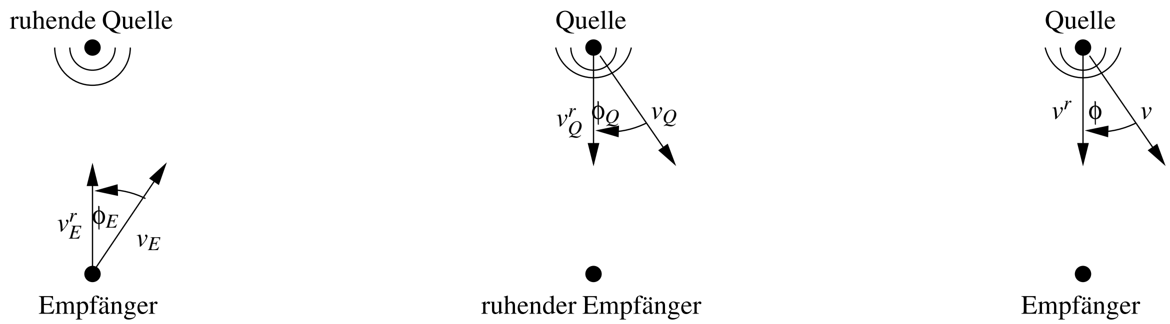

Abbildung 2.2: Dopplereffekt

Der *Dopplereffekt* ist ein allgemein bei Wellen beobachteter Effekt, der auftritt, wenn sich Quelle und Empfänger relativ zueinander oder relativ zu einem Ausbreitungsmedium bewegen. Für die folgenden Erklärungen denke man zunächst etwa an Schallwellen, bei denen es ein ausgezeichnetes Medium — die ruhende Luft — gibt.

Bewegt sich der Empfänger (Abbildung 2.2) relativ zur im Medium ruhenden Quelle mit der radialen[2] Geschwindigkeit $v_E^r = v_E \cos\phi_E$ (positiv, falls er sich auf den Sender zubewegt, sonst negativ), so stellt der

[2] Das ist die auf die Verbindungsachse zwischen Empfänger und Quelle projizierte Geschwindigkeit.

Empfänger gegenüber der Frequenz des Senders f_0 eine Frequenz f fest (c_m ist hierbei und im folgenden die Ausbreitungsgeschwindigkeit im Medium):

$$f = f_0(1 + \frac{v_E^r}{c_m}). \tag{2.11}$$

Wenn sich alerdings die Quelle mit der radialen Geschwindigkeit $v_S^r = v_S \cos\phi_S$ bewegt und der Empfänger ruht, beträgt die vom Empfänger beobachtete Frequenz f:

$$f = f_0 \frac{1}{1 - \frac{v_S^r}{c_m}}. \tag{2.12}$$

Hierbei ist v_S^r wieder positiv bei Annäherung. Kombiniert man beide Effekte so bekommt man

$$f = f_0 \frac{1 + \frac{v_E^r}{c_m}}{1 - \frac{v_S^r}{c_m}}. \tag{2.13}$$

Für Schallwellen sind diese Formeln korrekt. Im Falle elektromagnetischer Wellen läßt sich allerdings kein ruhendes Medium auszeichnen. Hier kommt es daher nur auf die relative Geschwindigkeit zwischen Empfänger und Quelle an. Ist diese sehr viel kleiner als die Lichtgeschwindigkeit c, so kann in erster Näherung auch in eine der Gleichungen (2.11) oder (2.12) eingesetzt werden.

Die allgemeine Dopplerformel für elektromagnetische Wellen sieht wie folgt aus (vergleiche Abbildung 2.2):

$$f = f_0 \frac{\sqrt{1 - \frac{v^2}{c^2}}}{1 - \frac{v}{c}\cos\phi} = f_0 \sqrt{1 - \frac{v^2}{c^2}} \frac{1}{1 - \frac{v_r}{c}}. \tag{2.14}$$

Hierbei ist ϕ der im Inertialsystem des Empfängers gemessene Winkel zwischen Geschwindigkeitvektor der Quelle und der Ausbreitungsrichtung der Welle. Mit $v_r = v\cos\phi$ wird wiederum die radiale Geschwindigkeit bezeichnet. Gegenüber der Gleichung (2.12) tritt zusätzlich ein relativistischer Korrekturfaktor $\sqrt{1 - \frac{v^2}{c^2}}$ auf.

Setzt man $\phi = 90°$ so erhält man die Gleichung für den *transversalen Dopplereffekt*, der auch *Dopplereffekt zweiter Ordnung* genannt wird:

$$f_{\text{transversal}} = f_0 \sqrt{1 - \frac{v^2}{c^2}}. \tag{2.15}$$

Hierbei bewegt sich die Quelle senkrecht zur Richtung aus der die Welle empfangen wird. In den nichtrelativistischen Formeln tritt hierbei überhaupt kein Dopplereffekt auf. Nur der relativistische Korrekturfaktor bleibt bestehen. Die hierbei beobachtete Frequenzänderung ist alleinige Folge aus dem Effekt der Zeitdilatation.

2.1.3 Allgemeine Relativitätstheorie

Eine Darstellung der allgemeinen Relativitätstheorie erfordert ein relativ aufwendiges und unhandliches mathematisches Instrumentarium. Deshalb sei hier nur ein wichtiges Ergebnis dieser Theorie dargestellt: Die Veränderung des Zeitablaufs durch die Gravitation.

Man betrachte zwei Orte A und B mit verschiedenen Graviationspotentialen Φ_A und Φ_B. Betrage die Gravitationspotentialdifferenz $\Delta\Phi = \Phi_B - \Phi_A$ (das ist jene Energie, die einer Einheitsmasse zugeführt werden muß, um sie gegen die Gravitation von A nach B zu transportieren), so gilt für die an den entsprechenden Orten verstrichenen Zeiten $\Delta t_A, \Delta t_B$ folgende Beziehung:

$$\frac{\Delta t_A - \Delta t_B}{\Delta t_A} = -\frac{\Delta\Phi}{c^2}. \tag{2.16}$$

Zur Veranschaulichung sei erwähnt, daß zwei ideale Uhren, von denen sich eine auf Meereshöhe und die andere in einer Höhenlage von 1000 m befindet, verschieden schnell zu laufen scheinen. Vergleicht man laufend die Zählstände, so wird man erkennen, daß die Uhr auf 1000 m um 1 Teil in 10^{13} schneller laufen wird als jene auf Meereshöhe (d.h. ihr Zählerstand ist nach 10^{13} Zählvorgängen um 1 größer als ihr Pendant auf 0 m), siehe Abbildung 2.3.

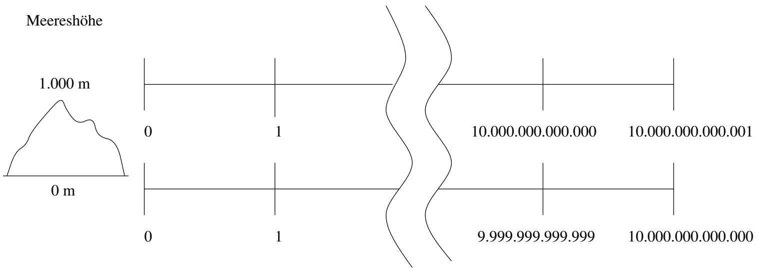

Abbildung 2.3: Gangunterschied von Uhren in unterschiedlichen Höhenlagen

2.2 Die Messung der Zeit

2.2.1 Allgemeines

Die Zeit ist jene physikalische Größe, die mit Abstand am genauesten gemessen werden kann. Moderne Atomuhren erreichen hierbei relative Genauigkeiten in der Größenordnung von 10^{-14}. Eine *Uhr* ist im allgemeinen ein Instrument, das den Ablauf der Zeit durch einen zeitabhängigen Vorgang in eine direkter zugängliche physikalische Größe umwandelt. Fast ausschließlich kommen hier periodische Schwingungsvorgänge (*Oszillatoren*) zum Einsatz, die durch ihre Periodendauer T_0 ein natürliches Zeitintervall vorgeben.

Allgemein läßt sich jedem solchen Vorgang — ob mit sinusförmigen Verlauf oder auch nicht — eine zeitabhängige Phase $\varphi(t) = t/T_0 = \nu_0 t$ zuordnen, die jeweils nach Ablauf einer Periode genau um 1 angewachsen ist. Der Faktor $\nu_0 = 1/T_0$ ist hierbei die *Frequenz* des Schwingungsvorgangs. Für den bei weitem wichtigsten Fall einer sinusförmigen Beziehung zwischen physikalischer Größe und Zeit, verwendet man den *momentanen Phasenwinkel*[3],

$$\Phi(t) = 2\pi\varphi(t) = 2\pi\nu_0 t. \tag{2.17}$$

der eine einfache Darstellung des Schwingungsvorganges erlaubt:

$$U(t) = U_0 \sin\Phi(t) = U_0 \sin 2\pi\nu_0 t. \tag{2.18}$$

[3]Zur Abkürzung wird im folgenden der Phasenwinkel, wenn auch nicht korrekt, manchmal kurz als Phase bezeichnet.

Die hier gewählte physikalische Größe ist die elektrische Spannung U. (U_0 ist der Amplitudenwert der Schwingung.) Um ein unbekanntes Zeitintervall Δt zu bestimmen, muß dazu die Phasendifferenz $\Delta\Phi = \Phi(t_2) - \Phi(t_1)$ zwischen Beginn (t_1) und Ende (t_2) des Intervalls gemessen werden. In Umkehrung der Gleichung (2.17) ergibt sich dann für das Zeitintervall

$$\Delta t = \frac{\Delta\Phi}{2\pi\nu_0}. \tag{2.19}$$

Die Bestimmung der Phasendifferenz unterliegt allerdings dem Problem, daß eine Sinusfunktion nicht eindeutig umkehrbar ist, weswegen folgendes Vorgehen notwendig ist (siehe Abbildung 2.4): Man messe die Phasenlage innerhalb der Periode zu Beginn (ϕ_1) und Ende (ϕ_2) des Zeitintervalls, und ermittle durch Abzählen die Anzahl n der dazwischenliegenden vollständig durchlaufenen Perioden (vom jeweils ansteigend durchlaufenen Nulldurchgang des Sinus zum nächsten). Die gesamte Phasendifferenz ist dann folgende ($T_0 = 1/\nu_0$):

$$\Delta\Phi = (2\pi - \phi_1) + 2\pi n + \phi_2. \tag{2.20}$$

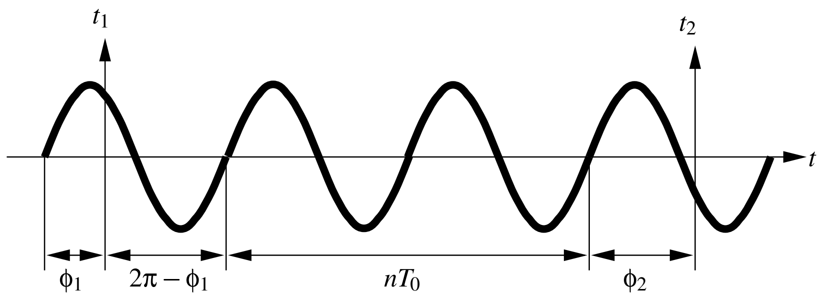

Abbildung 2.4: Messung der Phasenwinkeldifferenz zwischen den Zeitpunkten t_1 und t_2

Verzichtet man auf die Bestimmung der Phasenlage und zählt lediglich die Anzahl der innerhalb des Intervalls auftretenden ansteigenden Nulldurchgänge, so beträgt der maximale Fehler der dadurch begangen wird, $\pm T_0$. Für Uhren, die die Zeit auf solche Weise messen, begrenzt die Periodendauer T_0 daher die Auflösung. (Es handelt sich hier um eine Art Quantisierungsfehler.) Manchmal spricht man dann davon, daß die Uhr eine Granularität von $g = T_0$ hätte. Um diesem Manko zu begegnen, kann die Frequenz erhöht werden.

2.2.2 Charakterisierung realer Oszillatoren

In der Wirklichkeit wird obiges Idealmodell eines exakten Sinusverlaufs nie vorkommen. Vielmehr treten in technisch realisierten Oszillatoren Abweichungen auf, die vielfältige Ursachen haben können (z.B. Rauscheffekte, Einfluß von Umgebungsparametern, Alterungseffekte, …). Ein geeigneter allgemeiner Ansatz zur Beschreibung dieser Abweichungen ist folgender:

$$U(t) = (U_0 + \varepsilon(t))\sin(2\pi\nu_0 + \phi(t)). \tag{2.21}$$

Gewöhnlich kann der Term $\varepsilon(t)$, der die Amplitudenschwankungen beschreibt, vernachlässigt werden, insbesondere für längere Betrachtungzeiträume. Der wichtigere Term $\phi(t)$ drückt die Abweichung der Phase des Oszillators vom Idealmodell mit Nominalfrequenz ν_0 aus: $\phi(t) = \Phi_{\text{real}}(t) - \Phi_{\text{ideal}}(t) = \Phi(t) - 2\pi\nu_0 t$. Er wird die *momentane Phasenwinkelabweichung* genannt. Direkt mit der Phasenwinkelabweichung verknüpft ist die zeitliche Abweichung $x(t)$, die auch als *momentane Zeitabweichung* bezeichnet wird. Sie kennzeichnet die momentane zeitliche Ablage des Oszillators vom Idealmodell und ergibt sich unter anderem aus Gleichung (2.17):

$$x(t) = \frac{\phi(t)}{2\pi\nu_0}. \tag{2.22}$$

Das Frequenzverhalten des Oszillators läßt sich durch Definition der *momentanen Frequenz*

$$\nu(t) = \frac{1}{2\pi}\frac{\mathrm{d}\Phi(t)}{\mathrm{d}t}. \tag{2.23}$$

darstellen. Sie beschreibt die Geschwindigkeit, mit der die Phase voranschreitet. Eine aussagekräftigere Größe als die momentane Frequenz ist die *momentane relative Frequenzabweichung*

$$y(t) = \frac{\nu(t) - \nu_0}{\nu_0} = \frac{1}{2\pi\nu_0}\frac{\mathrm{d}\phi(t)}{\mathrm{d}t} = \frac{\mathrm{d}x(t)}{\mathrm{d}t}. \tag{2.24}$$

Mit ihr wird die Frequenzabweichung des Oszillators auf die Nominalfrequenz ν_0 bezogen. Bemerkenswert ist dabei der in Gleichung (2.24) ausgedrückte Zusammenhang zwischen $y(t)$ und $x(t)$.

Um das Verhalten eines Oszillators möglichst genau vorhersagen zu können, ist ein adäquates Modell nötig, das sein Abweichen vom Ideal möglichst exakt beschreibt. Bei einer solchen Beschreibung muß man allerdings darauf achten, daß jeder komplexere physikalische Vorgang — daher auch ein Oszillator — einerseits ein wohl determiniertes (systematisches) Verhalten und andererseits ein scheinbar dem Zufall gehorchendes (stochastisches) Verhalten zeigt. Die Ursache des zweiteren liegt in der Komplexität und in prinzipiellen Indeterminismen des Schwingungsprozesses. Zu ihrer Beschreibung muß man statistische Methoden einsetzen.

Die Trennung in beide Verhaltensweisen überträgt sich dabei auf alle beobachtbaren Größen, z.B. auch auf die relative Frequenzabweichung $y(t)$:

$$y(t) = y_{\text{systematisch}}(t) + y_{\text{stochastisch}}(t) \tag{2.25}$$

In den folgenden zwei Abschnitten sollen daher die beiden Terme der rechten Seite nacheinander behandelt werden.

2.2.3 Systematische Abweichungen — Frequenzgenauigkeit

Der systematische Anteil der relativen Frequenzabweichung ist meist direkt von verschiedenen Parametern wie Zeit t, Temperatur T, Druck P, … abhängig. Diese Abhängigkeit wird auch durch folgende funktionale Beziehung ausgedrückt.

$$y(t) = y(t, T, P, \ldots). \tag{2.26}$$

Es ist zu erkennen, daß trotz Berücksichtigung der Umgebungsparameter eine explizite Zeitabhängikeit vorhanden sein kann. Dieses Phänomen wird durch die Alterung eines Oszillators verursacht. Betrachtet man den Oszillator zunächst unter Standardbedingungen zum Zeitpunkt t_0, mit Temperatur T_0, Druck $P_0, \ldots$ und variiert die einzelnen Parameter, so kann $y(t)$— sofern die Variationen klein genug sind — als Summe unabhängiger Beiträge dargestellt werden:

$$y(t) = y(t_0, T_0, P_0, \ldots) + \Delta y_t(t - t_0) + \Delta y_T(T - T_0) + \Delta y_P(P - P_0) + \ldots. \tag{2.27}$$

Linearisiert man außerdem die einzelnen Funktionen und benutzt die Abkürzung $y_0 = y(t_0, T_0, P_0, \ldots)$, so kann man weiter vereinfachen:

$$y(t) = y_0 + D(t - t_0) + k_T(T - T_0) + k_P(P - P_0) + \ldots . \tag{2.28}$$

Die Abhängigkeit von einem der Parameter läßt sich somit direkt durch eine einzige Zahl ausdrücken. Die wichtigsten Parameter seien im folgenden näher besprochen (siehe auch Hellwig [1990]):

Drift — Alterung

Die *Drift* ist der Effekt der langsamen Änderung der Frequenz eines Oszillators durch Alterungsprozesse im Oszillator selbst. Sie kommt in Gleichung (2.28) durch den Koeffizienten D zum Ausdruck. Ist die dortige Näherung nicht zulässig, so läßt sich die zeitabhängige Drift so definieren:

$$D(t) = \frac{\partial y(t)}{\partial t}. \tag{2.29}$$

Ist die Drift konstant, so läuft die Frequenz linear mit der Zeit weg (Der Übersichtlichkeit wegen wurde $t_0 = 0$ gesetzt):

$$y(t) = y_0 + D * t. \tag{2.30}$$

Die zugehörige Zeitabweichung $x(t)$ zeigt dann quadratisches Verhalten:

$$x(t) = x_0 + y_0 t + \frac{1}{2} D t^2. \tag{2.31}$$

Die Ursache für die Drift liegt, wie bereits erwähnt, in Alterungsvorgängen innerhalb des Oszillators. Bei Quarzen beispielsweise kommt es zu einer langsamen Änderung der Kristallstruktur des Quarzmaterials. Die Alterung ist dabei am größten bei frisch produzierten Oszillatoren und nimmt im Laufe der Zeit ab. Deshalb werden Quarze künstlich gealtert, um eine möglichst geringe Drift zu erhalten.

Temperatur

Die Temperatur hat neben der Drift, oft den stärksten Einfluß auf die Frequenz des Oszillators. Man verwendet deshalb auch Quarzoszillatoren zur Temperaturmessung! Prinzipiell benutzt man zwei Methoden um den Einfluß zu minimieren: Verkleinerung der Temperaturabhängigkeit durch Kompensationsmechanismen oder Konstanthaltung der Temperatur.

Druck

Je nach Oszillatortyp wird man sehr unterschiedliche Sensitivität gegenüber Druckänderungen vorfinden. Diese Druckabhängigkeit ist aber meist (gegenüber der Temperaturabhängigkeit) von untergeordneter Bedeutung (Ausnahme: Einsatz von Oszillatoren in Flugzeugen, da der Druck stark mit der Höhe abnimmt).

Magnetfeld

Äußere Magnetfelder haben immer dann einen nicht vernachlässigbaren Einfluß, wenn der Oszillator selbst zum Betrieb ein Magnetfeld benötigt (z.B. Rubidumdampfatomuhr). Durch das Erdmagnetfeld kann es sogar zu einer Abhängigkeit von der Lage des Oszillators im Raum kommen.

Vibrationen und Beschleunigungen

Alle mechanisch schwingenden Systeme (hauptsächlich Quarze) sind natürlich empfindlich gegenüber Vibrationen und Beschleunigungen. Dies ist vornehmlich für den mobilen Einsatz eines Oszillators wichtig. Die natürliche Erdbeschleunigung kann hier ebenfalls zu einer Abhängigkeit der Frequenz von der Raumlage des Oszillators führen.

Spannungsversorgung und elektrische Belastung

In Systemen mit elektrischen Komponenten kann auch eine Änderung der Spannungsversorgung oder der elektrischen Belastung des Ausgangs des Oszillators eine Frequenzänderung nach sich ziehen. Sie sind aber im allgemeinen eher sehr gering.

Frequenzgenauigkeit

Die *Frequenzgenauigkeit* gibt allgemein an, wie genau ein Oszillator die gewünschte Frequenz darzustellen vermag. Für die Angabe einer solchen Frequenzgenauigkeit ist die Angabe des zulässigen Betriebbereiches, innerhalb dessen der Oszillator zu betreiben ist, um diese Genauigkeit zu erreichen, notwendig. Diese Genauigkeitsangabe beinhaltet dann natürlich alle Parametereinflüsse wie Temperatur, Druck, usw. Mit einbezogen ist hierbei auch der anfängliche Frequenzfehler, der durch den Produktionsprozess bedingt ist, da durch kleine Variationen in den räumlichen Ausmaßen, Materialzusammensetzung, usw. auch die Frequenz immer ein wenig unterschiedlich sein wird.

2.2.4 Stochastische Abweichungen — Frequenzstabilität

Wurden alle systematischen Abweichungen berücksichtigt, so bleiben nur mehr die stochastischen Schwankungen über. Eine statistische Analyse derselben kann dabei aus zwei verschiedenen Perspektiven erfolgen. Je nach Anwendung kann eine Betrachtung im *Zeitbereich* oder aber im *Frequenzbereich* zweckmäßig sein (siehe Allan [1987]):

Zeitbereich

Im Zeitbereich interessiert vor allem die über einen bestimmten Mittelungszeitraum τ akummulierte Zeitabweichung $\Delta x(t,\tau)$:

$$\Delta x(t,\tau) = x(t+\tau) - x(t). \tag{2.32}$$

Dividiert man durch die Mittelungszeit, so erhält man die über den Zeitraum τ gemittelte relative Frequenzabweichung $\overline{y}(t,\tau)$:

$$\overline{y}(t,\tau) = \frac{x(t+\tau) - x(t)}{\tau} = \frac{1}{\tau}\int_{t}^{t+\tau} y(t)\mathrm{d}t. \tag{2.33}$$

Ein geeignetes Maß für die Schwankung von $\overline{y}$ für festes τ beschreibt dann die *Frequenzstabilität* für diesen Mittelungszeitraum. Ein von der IEEE empfohlenes Maß ist die *Allan-Varianz*. Sie ist die Zwei-Proben Varianz von $\overline{y}$

$$\sigma_y^2(\tau) = \left\langle \frac{1}{2} \left(\overline{y}(t+\tau,\tau) - \overline{y}(t,\tau)\right)^2 \right\rangle . \tag{2.34}$$

Hierbei drücken die Klammern $\langle$ und $\rangle$ den über die gesamte Zeitachse erstreckten zeitlichen Mittelwert des von ihnen eingeschlossenen Ausdrucks aus:

$$\langle A(t) \rangle = \lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} A(t)\mathrm{d}t. \tag{2.35}$$

Mißt man alle τ Zeiteinheiten die mittlere Frequenzabweichung $y_i = \overline{y}(t+(i-1)\tau,\tau)$, so kann man mit N vorliegenden Meßwerten ($1 \leq i \leq N$) folgende Schätzung für die Allan-Varianz benutzen:

$$\sigma_y^2(\tau) = \frac{1}{N-1} \sum_{i=1}^{N-1} \frac{1}{2} \left(y_{i+1} - y_i\right)^2 . \tag{2.36}$$

Frequenzbereich

Als bevorzugtes Maß für die Frequenzstabilität wurde vom IEEE (siehe Stein [1985]) die einseitige spektrale Leistungsdichte $S_y(f)$ der Größe $y(t)$ empfohlen. Zu ihr gelangt man auf folgendem Wege: Man betrachte zunächst die Autokorrelationsfunktion $r_{yy}(\tau)$ von $y(t)$:

$$r_{yy}(t,\tau) = \mathrm{E}(y(t)y(t+\tau)). \tag{2.37}$$

Die Operation E drückt dabei den stochastischen Mittelwert über alle im betrachteten Ensemble auftretenden stochastischen Prozesse $y(t)$ aus. Beschränkt man sich auf den stationären Fall, (Hierbei sind alle stochastischen Eigenschaft zeitunabhängig.) so vereinfacht sich die Gleichung zu:

$$r_{yy}(\tau) = \mathrm{E}(y(t)y(t+\tau)) \quad t \text{beliebig}. \tag{2.38}$$

Führt man auf dieser Autokorrelationsfunktion die Fouriertransformation durch, so erhält man die zweiseitige spektrale Leistungsdichte $\overline{S}_y(f)$:

$$\overline{S}_y(f) = \int_{-\infty}^{\infty} r_{yy}(\tau) e^{-i2\pi f\tau} \mathrm{d}\tau. \tag{2.39}$$

Nachdem die Autokorrelationsfunktion eine gerade Funktion ist, ist die Fouriertransformierte reell und gerade. Daher läßt sich die einseitige spektrale Leistungsdichte so formulieren:

$$S_y(f) = \overline{S}_y(f) + \overline{S}_y(-f) = 2\overline{S}_y(f). \tag{2.40}$$

Weil $y(t)$ dimensionslos ist, ist die Einheit der Spektralleistungsdichte $1/\mathrm{Hz}$. Aus $S_y(f)$ kann der Beitrag eines Frequenzintervalls (f_1, f_2) zur „Signalleistung“ — dem mittleren Signalquadrat — $\langle y^2\rangle$ direkt bestimmt werden:

$$\langle y^2\rangle = \int_{f_1}^{f_2} S_y(f)\mathrm{d}f. \tag{2.41}$$

Da die spektrale Leistungsdichte $S_y(f)$ das Rauschverhalten eines Oszillators eindeutig festlegt, kann aus ihr die Allan-Varianz berechnet werden:

$$\sigma_y^2(\tau) = \frac{2}{(\pi\nu_0\tau)^2}\int_0^\infty S_y(f)\sin^4(\pi f\tau)\mathrm{d}f. \tag{2.42}$$

Die Allan-Varianz hängt, wie aus der Gleichung ersichtlich ,sehr stark vom Verhalten des Spektrums im obersten Frequenzbereich ab.

Andere verwendete Spektralleistungsdichten sind jene der Phasenabweichung $\phi(t)$ (zur Charakterisierung des Phasenrauschens, *Phase Noise*) und der Zeitabweichung $x(t)$. Die Beziehungen zu $S_y(f)$ sind einfach:

$$S_y(f) = \left(\frac{f}{\nu_0}\right)^2 S_\phi(f) = (2\pi f)^2 S_x(f). \tag{2.43}$$

Die Angabe einer Spektralleistungsdichte erfolgt meist als Dezibelangabe relativ zur Leistung der Trägerwelle mit der Frequenz ν_0.

Die entsprechende Einheit ist dann dBc. Das c steht hierbei für *carrier*.

Rauscharten

Sehr oft läßt sich die Spektraldichte $S_y(f)$ als Summe ganzzahliger Potenzen darstellen:

$$S_y(f) = \sum_{\alpha=-2}^{2} h_\alpha f^\alpha. \tag{2.44}$$

Jeder dieser Summanden überwiegt zumeist in einem gewissen Frequenzbereich und kann daher leicht am Anstieg in einem doppellogarithmischen Graph identifiziert werden (siehe Abbildung 2.5).

Die einzelnen Summanden haben eigene Namen und besitzen jede eine einfach ausdrückbare Allan-Varianz:

Bezeichnung	$S_y(f)$	$\sigma_y^2(\tau)$
Weißes Phasenrauschen	$h_2 f^2$	$3 f_h h_2/(2\pi)^2\tau^2$
Flicker-Phasenrauschen	$h_1 f$	$\frac{3(2+\ln(2\pi f_h\tau)-\ln 2)}{(2\pi)^2} h_1 \frac{1}{\tau^2}$
Weißes Frequenzrauschen	h_0	$h_0/2\tau$
Flicker-Frequenzrauschen	$h_{-1}f^{-1}$	$2\ln(2)h_{-1}$
Random-Walk Frequenzrauschen	$h_{-2}f^{-2}$	$\frac{1}{6}(2\pi)^2 h_{-2}\tau$

Man beachte, daß für einige Rauscharten eine obere Grenzfrequenz f_h eingeführt werden muß, ab der die Spektraldichte Null wird, damit endliche Allan-Varianzen existieren.

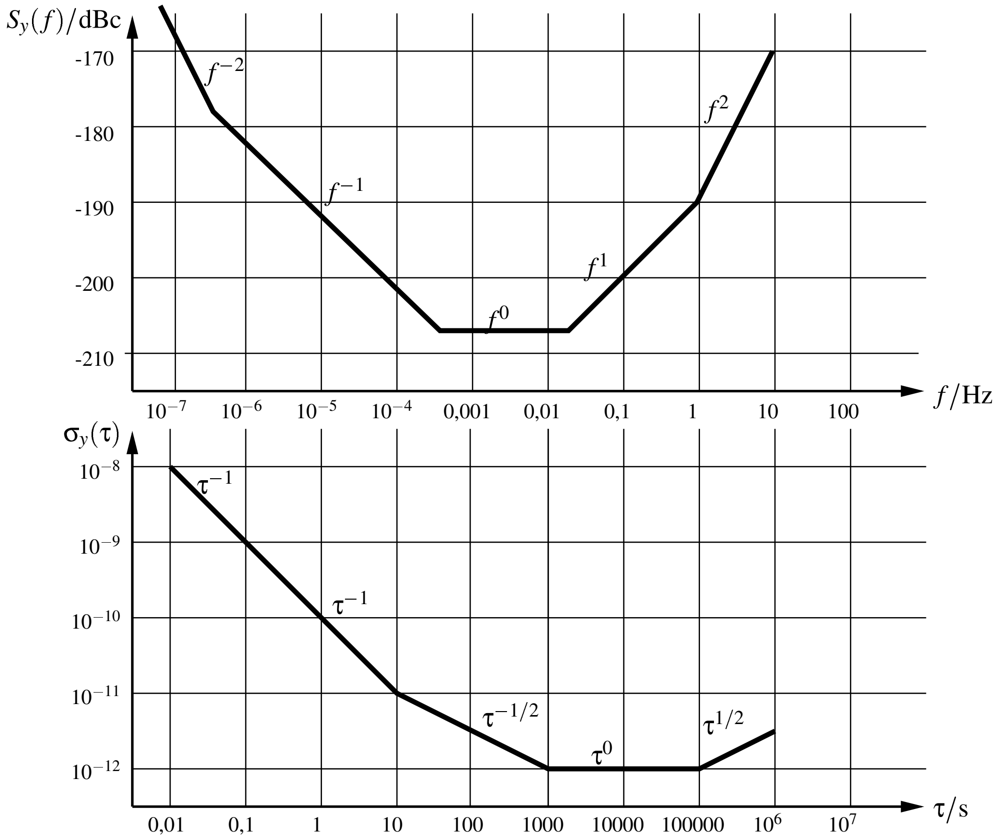

Abbildung 2.5: Wurzel aus der Allan-Varianz und Spektraldichte für einen Oszillator mit allen fünf Rauscharten

Akkumulierter Zeitfehler

Betrachtet man den Oszillator für eine bestimmte Mittelungszeit τ und wird das Rauschverhalten um τ durch eine Rauschart dominiert, so läßt sich die Standardabweichung des akkumulierten Zeitfehlers $\Delta x(t,\tau)$ angeben (Diese Standardabweichung ist als stochastische Eigenschaft im vorliegenden stationären Fall zeitunabhängig.):

$$\sigma_{\Delta x(\tau)} = \sigma_{\Delta x(t,\tau)} = \sqrt{\mathrm{E}(\Delta x(t,\tau)^2)} \quad t \text{beliebig.} \tag{2.45}$$

angeben. (Es wurde hierbei die Tatsache benutzt, daß der Erwartungswert für den akkumulierten Zeitfehler für alle Rauscharten gleich Null ist.) Die entsprechenden Formeln sind in folgender Tabelle enthalten (τ_0 stellt die Abtastperiode für das Datenmaterial dar, aus dem die Allan-Varianz berechnet wurde; sie stellt eine Art Bandbegrenzung dar):

Rauschart	$\sigma_{\Delta x(\tau)}$
Weißes Phasenrauschen	$\tau \cdot \sigma_y(\tau)/\sqrt{3}$
Flicker-Phasenrauschen	$\sim \tau \cdot \sigma_y(\tau)\sqrt{\ln\tau/2\ln\tau_0}$
Weißes Frequenzrauschen	$\tau\sigma_y(\tau)$
Flicker-Frequenzrauschen	$\tau\sigma_y(\tau)/\sqrt{\ln 2}$
Random-Walk Frequenzrauschen	$\tau\sigma_y(\tau)$

2.3 Quarz- und Atomuhren

2.3.1 Quarzoszillatoren

Das Kernstück eines Quarzoszillators ist der Quarzkristall, der durch den umgekehrten piezoelektrischen Effekt zu Schwingungen angeregt werden kann. Wird er im positiven Rückkopplungszweig einer elektrischen Oszillatorschaltung verwendet, so beginnt der Quarz mit seiner Resonanzfrequenz zu schwingen.

Es gibt Quarzkristalle für Frequenzen von einigen 10 kHz bis über 2 GHz. Ihr Vielfalt äußert sich auch in wichtigen Leistungsdaten wie Sensitivität gegenüber verschiedenen Umgebungsparametern (Temperatur, Druck, Luftfeuchtigkeit, Magnetfeldstärke, ...), Kurzzeitstabilität, Driftrate, Leistungsaufnahme, Baugröße und nicht zuletzt dem Preis.

Für verschiedene Typen von Quarzoszillatoren haben sich Abkürzungen eingebürgert, die in folgender Liste wiedergegeben sind:

Abkürzung	Bezeichnung	Erklärung
XO	clock oscillator	gewöhnlicher Quarzoszillator
VCXO	voltage controlled XO	spannungsgesteuerter Quarzoszillator
TCXO	temperature compensated XO	temperaturkompensierter Quarzoszillator
OCXO	oven controlled XO	temperaturstabilisierter Quarzoszillator

Da auch die besten technischen Uhren immer einen Quarzoszillator (VCXO) besitzen, deren Frequenz auf die Zielfrequenz hingeregelt wird, ist deren Leistungsfähigkeit für hochgenaue Uhren von großer Bedeutung. Insbesondere die Kurzzeitstabilität hängt bei diesen Uhren vom benutzen Quarzoszillator ab.

2.3.2 Atomuhren

Die Sekunde ist derzeit die einzige Einheit des SI-Systems, die über Eigenschaften eines einzelnen Atoms, nämlich des Cäsium-133, definiert ist. Sei dem Jahre 1967 gilt die folgende auf der 13. Generalkonferenz für Maß und Gewicht beschlossene Sekundendefinition:

> *Die Sekunde ist die Dauer von 9.192.631.700 Perioden der Strahlung, die dem Übergang zwischen den Hyperfeinstrukturniveaus des Grundzustandes des ^{133}Cs-Atoms entspricht.*

Diese Definition basiert auf dem Wissen aus der Quantenmechanik, daß Atome in verschiedenen Energiezuständen existieren können, die durch den Energiewert E gekennzeichnet werden. Durch die Wechselwirkung mit elektromagnetischer Strahlung kann ein Atom von einem Energiezustand E_1 in einen höheren E_2 übergehen. Dabei kann ein *Photon* (Wechselwirkungsteilchen der elektromagnetischen Strahlung) mit der Frequenz $f_0 = (E_2 - E_1)/h$ absorbiert (aufgenommen) werden[4]. Entsprechend kann in einem höheren Energiezustand durch spontanen Übergang in den niedrigeren Zustand ein Photon entsprechender Energie emittiert (*spontane Emission*) werden.

Bei den Atomuhren wird weiters die Tastsache verwendet, daß die Wahrscheinlichkeit, daß ein Photon bestimmter Frequenz absorbiert wird, umso größer ist, je näher sie bei der durch den Energiedifferenzbetrag ΔE des Übergangs bestimmten Frequenz $f_0 = \Delta E/h$ liegt. Eine Wahrscheinlichkeitsverteilung nach der Frequenz aufgetragen ergibt für gewöhnlich eine sehr scharf definierte Resonanzfigur.

Von den vielen exisiterenden Übergängen, die in einem Atom existieren, werden bei den Atomuhren meist jene zwischen *Hyperfeinstrukturniveaus* benutzt, die sich aus der Wechselwirkung des magnetischen Moments des Atomkerns und des äußersten Hüllenelektrons ergeben. Aufgrund der geringen Wechselwirkungsenergie führt dies zu den relativ leicht handhabbaren Frequenzen im Mikrowellenbereich. In Abbildung 2.6 sind die Energieübergänge der drei wichtigsten Uhrentypen dargestellt.

[4] h ist das sogenannte *Plankscher Wirkungsquantum*

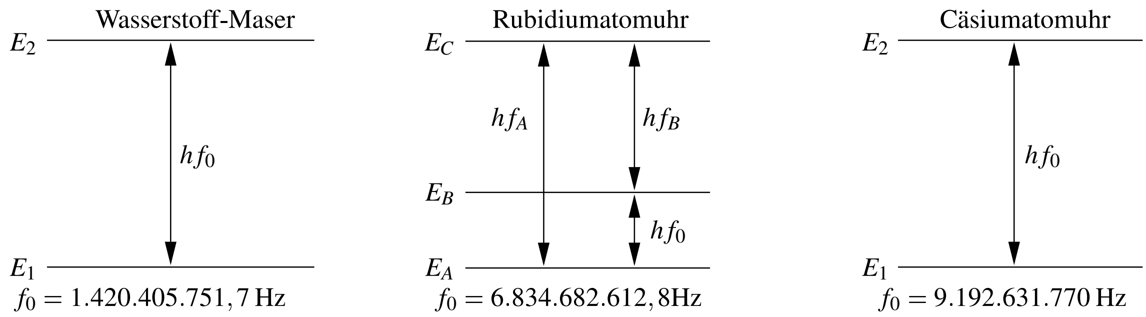

Abbildung 2.6: Die in Atomuhren benutzten Energieniveaus der Atome H, Rb und Cs

Cäsiumatomuhr

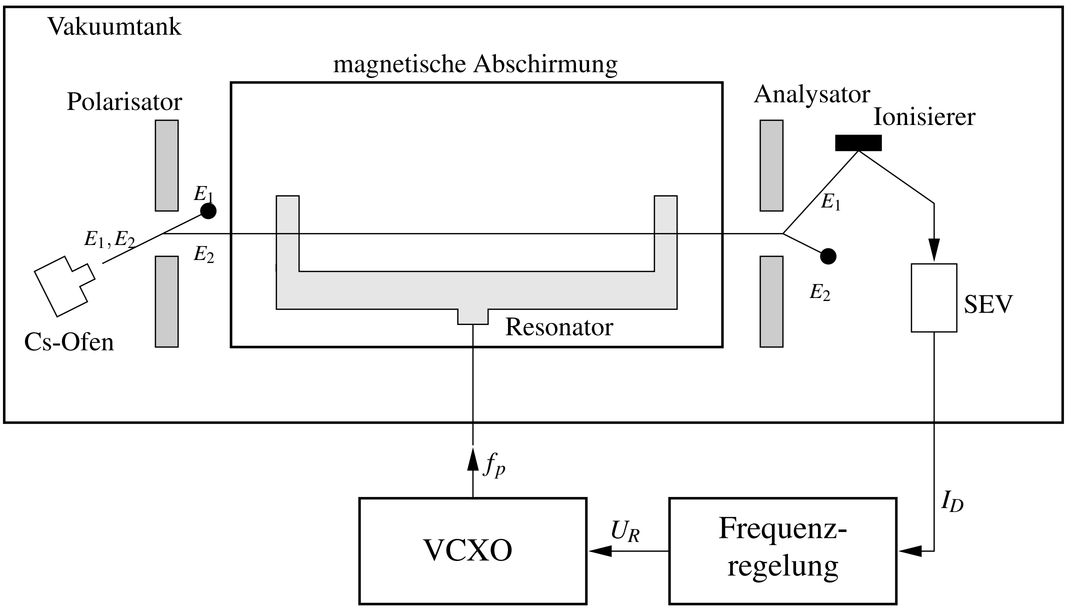

Abbildung 2.7: Aufbau einer Cäsiumatomuhr

Der Aufbau einer Cäsiumatomuhr kann der Abbildung 2.7 entnommen werden (siehe auch Hilbert [1993]). Sie basiert auf der sogenannten *Atomstrahlresonanzmethode*: Im Cäsiumofen wird Cäsium erhitzt und ein Gasstrahl atomaren Cäsiums durch einen Polarisator geführt. Dieser filtert alle Atome im niedrigen Energiezustand E_1 heraus und leitet alle anderen Atome im Energiezustand E_2 in den Resonator weiter. Ein Resonator wird von einem steuerbaren Quarz (VCXO) gespeist und strahlt dadurch Mikrowellenstrahlung der Frequenz f_p auf die Atome ein. Im Resonanzfall $f_p = f_0$ gehen die Atome in den Zustand E_1 über. In einem Analysator werden schließlich nur alle Atome im Zustand E_1 zum Ionisierer weitergeleitet, der die Cs-Atome zu Cs^+-Ionen umwandelt. Diese werden auf einen Sekundärelektronenvervielfacher gelenkt, der einen Strom I_D abgibt, der proportional zur Anzahl der auftreffenden Cäsiumatome ist. Dieser wird schließlich an eine Frequenzregelung geführt, die die Steuerspannung U_R für den VCXO entsprechend anpaßt.

Die nominelle Resonanzfrequenz des verwendeten Übergangs (Übergang zwischen den Hyperfeinstrukturniveaus des Grundzustands) des unbeeinflußten Cäsiumatome wird mit der SI-Sekundendefinition auf 9.192.631.770 Hz festgelegt. Durch verschiedene Einflüsse kommt es aber zu einer Verschiebung der Resonanzfrequenz. Diese müssen im nachhinein rechnerisch korrigiert oder zumindest abgeschätzt werden. Immer wird dabei eine gewisse Unsicherheit zurückbleiben, die die erreichbare Frequenzgenauigkeit beschränkt. Die ungefähre Unsicherheit der besten Cäsiumatomuhren liegt derzeit ungefähr bei $1 \cdot 10^{-14}$. Obwohl die erste Cäsiumatomuhr bereits 1955 konstruiert wurde, ist dieser Uhrentyp bis heute in seiner Langezeitstabilität und Frequenzunsicherheit ungeschlagen.

Interessante Details über Cäsiumatomuhren lassen sich in dem Artikel Audoin [1992] finden.

Rubidiumdampfatomuhr

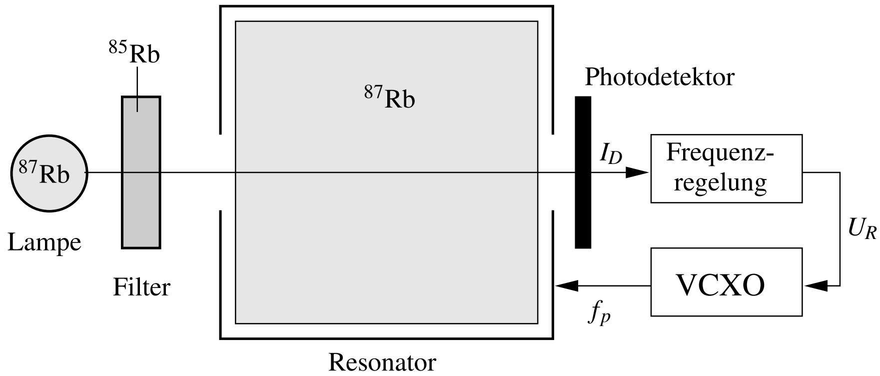

Abbildung 2.8: Aufbau einer Rubidiumdampfatomuhr

In Abbildung 2.6 ist das Energieniveauschema, das für die Rubidiumdampfatomuhr relevant ist, abgebildet. Es treten hier zwei Übergänge ($E_A \rightarrow E_C$ und $E_B \rightarrow E_C$) auf, die Frequenzen f_A, f_B im optischen Frequenzbereich entsprechen. Die Frequenz f_0, die ausgenutzt werden soll, ist durch den Übergang von $E_A \rightarrow E_B$ gegeben.

Das Funktionsprinzip sei anhand des schematischen Aufbaus (Abbildung 2.8) beschrieben: Eine Gasentladungslampe gefüllt mit Rubidiumdampf (^{87}Rb) erzeugt ein Emissionspektrum, das die Frequenzen aller möglichen Übergänge, darunter auch f_A und f_B im sichtbaren Spektrum enthalten. Beim Durchgang durch eine Filterzelle, die ^{85}Rb enthält, wird das Licht mit der Frequenz f_B herausgefiltert. Das restliche Licht (Frequenz f_A) gelangt in den Resonator. In diesem Resonator befinden sich ebenfalls ^{87}Rb-Atome, die durch das einfallende Licht (Absorption) in den Energiezustand E_C gelangen. Durch spontane Emission fallen die Atome mit gleicher Wahrscheinlichkeit in den Zustand E_A und E_B zurück. Da die Atome auf dem Energieniveau E_A durch das Licht sofort wieder auf E_C angehoben werden, sind viel mehr Atome im Zustand E_B als im Zustand E_A (*Selektion*). Setzt man die Atome nun einem Mikrowellenfeld mit einer Frequenz, die dem Übergang $E_A \rightarrow E_B$ entspricht aus (Resonanzfall), so tritt ein Austausch zwischen den beiden Niveaus auf, der dazu führt, daß wieder Atome im Zustand E_A vorhanden sind. Durch das Licht werden diese wiederum in den Zustand E_C gehoben, wodurch die Intensität des Lichtes abgeschwächt wird. Diese Abschwächung ist dabei umso größer, je effektiver der Austausch zwischen den beiden Niveaus E_A und E_B ist. Im Resonanzfall (die eingestrahlte Frequenz entspricht dem Energieübergang zwischen E_A und E_B) tritt ein Minimum der Lichtintensität auf. Dies wird mit einem Photodetektor wahrgenommen, der mit seinem Ausgangsstrom I_D eine Frequenzregelung speist. Deren Regelungsspannung U_R steuert wiederum den VCXO an, der die Mikrowellenfrequenz f_p erzeugt.

Ähnlich wie bei der Cäsiumatomuhr treten hier einige frequenzverschiebende Effekte auf. Als weiterer Nachteil gesellt sich bei der Rubidiumatomuhr die starke Alterung hinzu. Aus diesem Grund kann die Rubidiumatomuhr nicht als primäres Normal angesehen werden. Sie muß kalibriert werden. Ihr besonderer Vorteil ist der relativ geringe Preis, das kleine Bauvolumen und die weitestgehende Wartungsfreiheit.

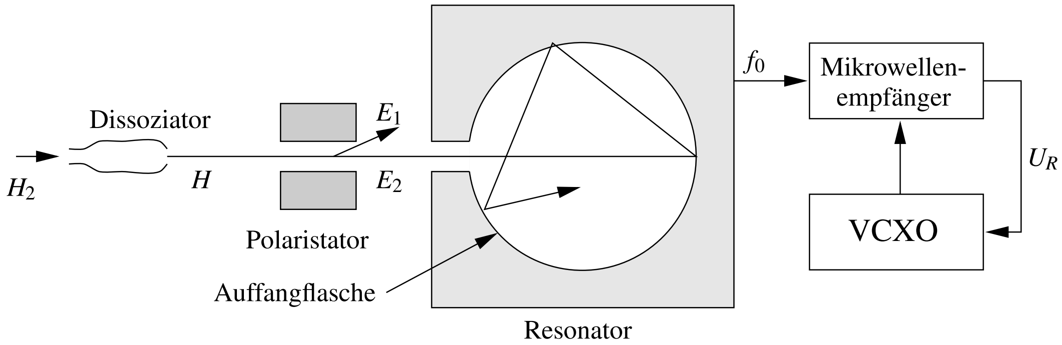

Abbildung 2.9: Aufbau eines Wasserstoff-MASERs

Wasserstoff-Maser

Gemäß dem Energieniveauschema in Abbildung 2.6 ist die atomare Resonanzfrequenz beim Wasserstoff-Maser (H-Maser) $f_0 = 1.420.405.751{,}7\ldots$ Hz. Ein Maser funktioniert ähnlich wie ein Laser, nur sind die erzeugten Frequenzen nicht im sichtbaren Bereich des elektromagnetischen Spektrums zu finden, sondern im Mikrowellenbereich.

Beim H-Maser wird ein Dissoziator mit Wasserstoffgas (H_2) gespeist. Dort werden die Wasserstoffmoleküle in Atome (H) gespalten. Diese treten in einen Polarisator, der Atome im hohen Energiezustand E_2 durchläßt und solche im Zustand E_1 zurückhält. Weiters gelangen sie durch eine kleine Öffnung in den Resonator, in dem sie relativ lange verweilen und öfters mit der mit Teflon beschichteten Innenwand des Resonators kollidieren und dabei nicht abstrahlen. Durch *stimulierte Emission* fallen die H-Atome in den Energiezustand E_1 und emittieren in Phase der einfallenden Strahlung Mikrowellenphotone der gewünschten Frequenz f_0. Ein Teil dieser Strahlung wird ausgekoppelt und über einen Mikrowellenempfänger und einem VCXO herausgefiltert.

Dieses Verfahren ist im Gegensatz zur Cäsiumatomuhr und Rubidiumatomuhr ein aktives. Es wird keine Resonanzfrequenz eingeregelt, sondern lediglich die von den Atomen emittierte Strahlung gemessen. Der große Vorteil von H-Masern ist deren hervorragende Frequenzstabilität für Zeiten in der Gegend von $\tau =$ 1000 s bis 20000 s: $\sigma_y(\tau) \approx 10^{-15}$.

H-Maser weisen eine relativ große Alterung (Frequenzdrift) auf. Diese rührt vermutlich von der Alterung der Teflonbeschichtung im Resonator her. Der große Nachteil von H-Masern ist dessen Preis, Baugröße und Gewicht.

2.4 Zeitskalen

Je größer der Kommunikations– und Transportbedarf zwischen weit auseinandergelegenen Orten ist, desto wichtiger wird es, eine globale Zeitbasis zu besitzen, die eine Synchronisation der Handlungsabläufe ermöglicht. Dies gilt auch für globale Meßverfahren in der Astronomie. Eine besonders große Bedeutung hat eine globale Zeitskala aber in Satellitennavigationssystemen. Synchronisationsfehler von nur 1 μs verursachen dort Positionsfehler von etwa 300 m.

2.4.1 Bestandteile einer Zeitskala

Die Grundlage einer Zeitskala ist eine Uhr im weitesten Sinne. Dies kann — wie im Beispiel der Internationalen Atomzeit TAI — auch die korrigierte gewichtete Mittelung mehrerer realer Uhren sein. Zur hinreichenden

Definition einer Zeitskala muß folgendes festgelegt werden:

- **Ein Bezugssystem.** Die Festlegung eines solchen Bezugssystems trägt dem lokalen Charakter der Zeit Rechnung (→ Allgemeine Relativitätstheorie, Abschnitt 2.1.3).
- **Einen Startzeitpunkt.** Dieser Zeitpunkt legt den Nullpunkt der Zeitskala fest.
- **Ein Zeitmaß.** Das Zeitmaß ist die Periodendauer eines periodischen Vorgangs (Uhr im weitesten Sinne).

Bei allen drei Begriffen ist zwischen deren Definition und der konkreten Realisierung zu unterscheiden, da die konkrete Realisierung die Definition (meist) nur mit einer gewissen Genauigkeit annähern kann. Nur falls die Zeitskala durch ein reales Objekt selbst definiert wird, ist dieser Hinweis hinfällig. Diese Art der Definition ist allerdings nicht sehr wünschenswert, da darunter die universelle Reproduzierbarkeit der Zeitskala bzw. des Zeitmaßes leidet.

2.4.2 Astronomische Zeitskalen

Das bürgerliche Leben ist abhängig von den wichtigsten astronomischen Uhren in unserem Sonnensystem. Der Tag (verursacht durch die Erdrotation gegen den Fixsternhimmel und die tägliche Fortbewegung der Erde auf ihrer Bahn um die Sonne) und das Jahr (volle Umkreisung der Sonne) legen die grundlegenden Rhythmen für Leben auf der Erde fest. Vor allem der tägliche Rhythmus schwankt nicht unwesentlich (siehe Hilbert [1988] und Meeus [1994]).

Weltzeit UT (universal time)

Die Zeitintervalle zweier zeitlich aufeinanderfolgender Höchststände der Sonne, wie sie an einem gewissen Punkt der Erde beobachtet werden, stellen die sogenannten *wahren Sonnentage* dar, deren Länge im Laufe des Jahres um bis zu ± 30 s von der über das ganze Jahr gemittelten Tageslänge abweichen. Dies hat zwei Gründe: Die Erde bewegt sich auf einer ellipsenförmigen Bahn und ist in der Nähe des Aphel (sonnennächster Punkt der Bahn) schneller als im Perihel (sonnenfernster Punkt). Zweitens bewegt sich die Sonne auf der Himmelskugel nicht entlang des Himmeläquators (sondern in der sogenannten Ekliptik). Durch Einführung einer fiktiven *mittleren Sonne* kann dem begegnet werden. Dabei geht man allerdings in zwei Schritten vor. Zunächst denkt man sich eine erste fiktive Sonne, die sich völlig gleichmäßig in der selben Ebene wie die wahre Sonne bewegt und im Aphel (und dadurch auch im Perihel) mit ihr übereinstimmt. Im zweiten Schritt führt man die eigentliche mittlere Sonne ein, die sich gleichmäßig entlang des Himmelsäquators bewegt und im Frühlingspunkt (Hier überschreitet die Sonne den Himmelsäquator aus der südlichen Himmelshemisphäre kommend; dies tritt zum astronomischen Frühlingsbeginn ein) mit der ersten fiktiven Sonne übereinstimmt. Die Differenz zwischen der mittleren und der wahren Sonne wird durch die sich sehr langsam ändernde *Zeitgleichung* ausgedrückt. Die maximale Abweichung der mittleren Sonne von der wahren Sonne beträgt etwa 16 Minuten.

Diese mittlere Sonne hängt zunächst noch von der astronomischen Länge des Beobachtungsortes ab. Deshalb nennt man die aus ihr konstruierte Zeitskala *Ortszeit*. Zu deren Definition legt man fest, daß dem Höchststand der mittleren Sonne die Ortszeit 12 Uhr entspricht. Wählt man *jene* Ortszeit aus, wie sie am Standard-Meridian — dieser liegt nahe dem früheren Greenwich-Meridian — gemessen wird, so erhält man die sogenannte Weltzeit UT[5]. Aus ihr ergeben sich durch Addition oder Subraktion ganzer Stunden (bis maximal 12) die sogenannten Zonenzeiten. Diese unterscheiden sich üblicherweise von der lokalen Ortszeit nicht mehr als um 1 Stunde. Die Bestimmung der Weltzeit obliegt dem *International Earth Rotation Service, IERS*, dessen zentrales Büro sich im Pariser Observatorium (*Observatoire Paris, OP*) befindet.

Durch den mittleren Sonnentag kann auch die Sekunde, wie es sehr lange bis zum Jahr 1956 der Fall war, definiert werden:

[5] Die Weltzeit UT wird heute manchmal falsch als *Greenwich Mean Time, GMT* bezeichnet; Bei GMT entspricht dem Sonnenhöchstand die Uhrzeit 0 Uhr. Diese beiden Zeitskalen differieren daher um 12 Stunden

Die Sekunde ist der 86.400 Teil eines mittleren Sonnentages.

Korrigierte Weltzeit UT1

Genaue Messungen zeigen, daß der Rotationspol innerhalb des Erdkörpers sich ständig verlagert. (Der Hauptanteil ist eine periodische Bewegung von etwa 6 m Radius). Dieser Einfluß führt zu Schwankungen in der Frequenz der UT-Zeitskala (in unseren Breiten Änderungen von bis zu $\pm 10^{-8}$). Deshalb reduziert man lokal gemessene Werte für die mittlere Sonnenzeit auf einen per Konvention festgelegten Rotationspol *CIO* (*Conventional Inertial Origin*). Diese korrigierte Weltzeit bezeichnet man als UT1. Entsprechend erhalten die noch unkorrigierten ortsabhängigen Weltzeiten den Namen UT0.

Korrigierte Weltzeit UT2

Eine von dieser Rotationspolwanderung unabhängige Schwankung wird durch die Änderung des Trägheitsmomentes der Erde hervorgerufen, die zu jährlichen periodischen Änderungen der Rotationsfrequenz der Erde führt. Wird dieser jahreszeitliche Einfluß zusätzlich berücksichtigt, so erhält man die zweite korrigierte Weltzeit UT2. Sie ist die gleichmäßigste Zeitskala, die aus der Erdrotation abgeleitet werden kann. UT2 hat heute keine Bedeutung mehr, da wesentlich gleichmäßigere Zeitskalen zur Verfügung stehen. Die Differenz zwischen UT1 und UT2 kann bis zu ±30 ms betragen.

Ephemeridenzeit

Da die Rotationsperiode der Erde, und damit auch die Weltzeit langfristig abnimmt und dabei sehr unregelmäßigen Schwankungen unterliegt, die nicht theoretisch modelliert werden können, sah man sich veranlaßt, die Einheit der Zeit über die Zeitdauer des tropischen Jahres (d.h. von Frühlingspunkt zu Frühlingspunkt) zu definieren (1956):

Die Sekunde ist der 31.556.925,9747te Teil des tropischen Jahres für den 0. Januar 1900 um 12 Uhr. [6]

Diese Definition hängt also vom Umlauf der Erde um die Sonne ab. Das tropische Jahr verändert sich ständig (deshalb auch die Bezugnahme auf einen speziellen Zeitpunkt), allerdings ist diese Abnahme hinreichend genau bekannt. Die hierdurch definierte Sekunde wird auch *Ephemeridensekunde* genannt. Die entsprechende Zeitskala heißt *Ephemeridenzeit*, kurz ET.

2.4.3 Atomzeitskalen

Bessere Zeitskalen als die durch die Uhren der Himmelsmechanik definierten sind die auf *Atomuhren* basierenden. Diese Atomzeitskalen fußen auf der Definition der *SI-Sekunde* von 1967, die möglichst gleich lang wie die ET-Sekunde sein sollte. Sie sei nochmals abgedruckt:

Die Sekunde ist die Dauer von 9.192.631.700 Perioden der Strahlung, die dem Übergang zwischen den Hyperfeinstrukturniveaus des Grundzustandes des ^{133}Cs entspricht.

Die Einheit der Zeit wird damit durch eine spezifische Eigenschaft einer Atomart festgelegt, von der angenommen wird, daß sie sich zeitlich und räumlich nicht ändert. Die Genauigkeit der durch diese Definition festgelegten Sekunde ist lediglich durch die meßtechnischen Möglichkeit begrenzt, die Periodendauer dieser Atomstrahlung zu reproduzieren.

[6] Der 0. Januar 1900 entspricht dem 31. Dezember 1899

TAI, Temps Atomique International

Die Internationale Atomzeit TAI ist auf der SI-Sekunde aufgebaut. Sie kann wie folgt definiert werden: Ihr Nullpunkt fällt mit dem 1. Januar 1958, 0 Uhr Weltzeit zusammen. Das Skalenmaß ist die in Meereshöhe (auf dem rotierenden Geoid) realisierte SI-Sekunde. Die Bezugnahme auf die in *Meereshöhe realisierte Sekunde* trägt den relativistischen Forderungen nach Angabe eines Bezugssystem Rechnung.

Obige Definition ist natürlich der Idealfall und kann nur so genau erfüllt werden, wie es die zur Verfügung stehenden Uhren erlauben. Zur Bildung der TAI-Zeitskala wird zunächst ein gewichtetes Mittel der Uhrenablesungen einer großen Anzahl von Atomuhren, die sich in Instituten auf der ganzen Welt befinden, erzeugt. Die Gewichte richten sich dabei nach der Qualität der jeweiligen Uhr. Diese Atomzeitskala wird als EAL (Echelle Atomique Libre) bezeichnet. Da die meisten Uhren kommerzielle Uhren sind, die signifikante Frequenzfehler besitzen, stimmt das Skalenmaß von EAL vorerst nicht mit der SI-Sekunde überein, dennoch erhält man hiermit eine Zeitskala mit hoher Kurzzeitstabilität.

Indem die Dauer der EAL-Sekunde langfristig auf die SI-Sekunde hingesteuert wird, erhält man schließlich die Internationale Atomzeit TAI. Die SI-Sekunde versucht man dabei durch die wenigen primären Uhren darzustellen. Die wichtigsten primären Atomuhren sind dabei die Cs-1 und Cs-2 der PTB (Physikalisch Technischen Bundesanstalt) in Deutschland. Die relative Unsicherheit bei der Realisierung der SI-Sekunde liegt bei diesen Uhren bei etwa 10^{-14}.

Die Ablagen der Uhren der einzelnen Institute werden schließlich mit einer Verzögerung von 1 bis 2 Monaten in dem Dokument *Cirular T* vom BIPM veröffentlicht (siehe Anhang Abschnitt C.3).

TD, Temps Dynamique, Dynamische Zeit

Mit der Einführung der TAI-Zeitskala wurde die ET-Zeitskala abgelöst. Um die für astronomische Zwecke wichtige ET-Skala weiterzuführen, wurde die Dynamische Zeit TD eingeführt, die eine relativistisch korrekte Weiterführung der ET-Zeitskala darstellt. Die Beziehung zwischen TAI und der Dynamische Zeit kann so ausgedrückt werden:

$$\mathrm{TDT} = \mathrm{ET} = \mathrm{TAI} + 32,184\ \mathrm{s} \tag{2.46}$$

In Wahrheit existieren wie die Gleichung bereits andeutet zwei dynamische Zeiten: *TDT, Temps Dynamique Terrestrique*, die für eine erdbezogene Uhr gilt und die *TDB, Temps Dynamique Baricentrique*, welche auf den Sonnensystemsschwerpunkt bezogen ist. Die gegenseitige Abweichung besitzt rein periodisches Verhalten (verursacht durch den Erdumlauf um die Sonne) und beträgt maximal 1,6 ms.

UTC, Universal Time Coordinated, koordinierte Weltzeit

Da die Weltzeit UT weit unregelmäßiger ist als die internationale Atomzeit TAI, entfernen sich die beiden Zeitskalen immer mehr voneinander. Um trotzdem eine gleichmäßige Zeitskala zu besitzen, die gleichzeitig mit guter Genauigkeit mit der Weltzeit UT1 synchron läuft, wurde die koordinierte Weltzeit UTC (Universal Time Coordinated) eingeführt. UTC ist dabei bis auf eine ganze Anzahl von Schaltsekunden mit TAI synchron. Durch Einfügen (oder falls notwendig durch Ausfallenlassen) von Schaltsekunden wird UTC synchron zu UT1 gehalten, und zwar so, daß der Differenzbetrag $\mathrm{DUT1} = \mathrm{UT1} - \mathrm{UTC}$ dem Absolutbetrag nach nicht größer als 0,9 s wird. Der bevorzugte Zeitpunkt zur Einfügung — bisher mußte noch keine Sekunde ausgelassen werden — einer Schaltsekunde ist jeweils nach der letzten Sekunde des Tages 31. Dezember oder 30. Juni eines Jahres (Diese Zeitpunkte sind Angaben nach der Zeitskala UTC). Die anderen möglichen Zeitpunkte sind am Ende des Tages 31. März bzw. 30. September.

Die Zeitskala UTC wurde am 1. Jänner 1972 eingeführt und hatte von Beginn an eine Schaltsekundendifferenz von $\mathrm{TAI} - \mathrm{UTC} = 10$ s (Da TAI und UT1 zu diesem Zeitpunkt bereits um einen entsprechenden Betrag auseinanderlagen). In den *Circular T* des BIPM wird die jeweils aktuelle Schaltsekundendifferenz n angegeben:

$$n = \mathrm{TAI} - \mathrm{UTC} \tag{2.47}$$

Seit der Schaltsekunde zum Jahreswechsel 1995/1996 beträgt der Wert von $n = 30$ s (Im Abschnitt C.4 sind alle bisher eingefügten Schaltsekunden aufgelistet).

Da UTC aus TAI abgeleitet wird, ist sie genauso von den bei TAI verwendeten langfristigen Mittelungsverfahren abhängig. Da TAI und UTC durch das vom BIPM herausgegebene Dokument *Circular T* definiert werden, spricht man auch davon, daß UTC und TAI im Gegensatz zu realen Uhren „Papieruhren" seien.

Da die Zeitskalen *post faktum* berechnet werden, sind sie dadurch auch nicht in Echtzeit verfügbar. Stattdessen muß für die Echtzeitanwendung eine reale Uhr eines der Institute verwendet werden, um eine genäherte Form von UTC zu erhalten.

GPST, GPS Time, GPS Systemzeit

Ein Satellitennavigationssystem, wie das GPS, benötigt eine sehr genaue Zeitbasis, da hier die Fehler in der Zeitmessung sich direkt in Positionsfehlern bemerkbar machen (in 1 μs durchlaufen elektromagnetische Signale 300 m). Die globale Zeitskala des GPS wird als *GPS Time (GPST)* oder als *GPS-Systemzeit* bezeichnet. Sie ist durch mehrere genaue Uhren (in den Satelliten und den Bodenstationen) definiert. Man nennt dieses Uhrenensemble auch *GPS Composite Clock (GPS CC)*. GPST basiert auf der sogenannten *Masterclock (MC)* des *United States Naval Obervatory (USNO)*, die die Zeitskala UTC(USNO, MC) definiert. Da sie in Echtzeit verfügbar ist, unterscheidet sie sich geringfügig von der langfristig ermittelten Institutszeit UTC(USNO) (sie ist wie UTC eine „Papieruhr"). GPS wird *UTC(USNO)* nachgesteuert. Die auftretenden Schaltsekunden macht GPST allerdings nicht mit . Es wird dabei garantiert, daß die Abweichung des GPST von UTC(USNO) betragsmäßig niemals größer als 1 μs wird. Tatsächlich betrug die Abweichung in den letzten Jahren selten mehr als 100 ns. Die Beziehung zwischen UTC(USNO, MC) und GPST läßt sich wie folgt darstellen:

$$\mathrm{GPST} - \mathrm{UTC(USNO, MC)} = n + A_0 + A_1(t - t_{\mathrm{ref}}) \tag{2.48}$$

Die Werte n, A_0, A_1 und t_{ref} werden im Navigationscode der GPS-Satellitensignale übermittelt. Der Parameter n beschreibt die durch die Schaltsekunden verursachte ganzzahlige Sekundendifferenz. Seit der Schaltsekunde Ende des Jahres 1995 betrug dieser Wert $n = 11$ s. A_0 und A_1 sind die Konstanten eines Polynoms erster Ordnung zur Beschreibung der kleinen Abweichungen zwischen UTC(USNO, MC) und GPS und t_{ref} ist der Anwendungszeitpunkt des Polynoms.

3 Das NavSTAR-GPS

3.1 Allgemeines

Das **Nav**igations **S**ystem with **T**iming **a**nd **R**anging **G**lobal **P**ositioning **S**ystem, kurz *NavStar GPS* oder *GPS*, ist ein satellitengestütztes Radionavigationssystem, das den mit entsprechenden GPS-Empfangsgeräten ausgerüsteten Nutzern die Bestimmung der Position (in drei Dimensionen), der Geschwindigkeit (in drei Dimensionen) und der Zeit in einem globalen erdbezogenen Bezugssystem in Echtzeit erlaubt. Der große Vorteil von GPS ist die außerordentliche Verfügbarkeit und Globalität: Es ist zu jeder Tageszeit, bei jedem Wetter und weltweit verwendbar, und zwar auf der Grundlage des einheitlichen und globalen Koordinatensystems *WGS-84 (World Geodetic System 1984)*. Diese Eigenschaften und die Erhältlichkeit preiswerter GPS-Empfangsgeräte machen es attraktiv führ sehr viele Anwendungsbereiche, die sich in drei Hautgruppen einteilen lassen:

- Navigationsanwendungen
- Geodätische Anwendungen
- Zeitanwendungen

Zu berücksichtigen ist, daß GPS für die Echtzeitnavigation im militärischen Bereich konzipiert und entwickelt wurde und sowohl die geodätischen als auch die Zeitanwendungen als Nebenprodukt abfielen. Deshalb besitzt das GPS zwei verschiedene Dienste: Allen Nutzern frei zugänglich ist der *Standard Positioning Service (SPS)*, der gegenüber dem zweiten Dienst, dem *Precise Positioning Service (PPS)* weit geringere Genauigkeit besitzt. Dieser zweite Dienst kann gewöhnlicherweise nur von Nutzern, die im Besitz des entsprechenden Codes sind — hauptsächlich das amerikanische Militär — verwendet werden. Die Mittel, die ergriffen werden, um zivilen Nutzern die Genauigkeit des PPS zu verwehren, heißen *Selective Availability (SA)* und *Anti-Spoofing (A-S)*, siehe Abschnitt 3.6.3 und 3.6.4.

Gute und recht ausführliche Einführungen in das GPS-System lassen sich in Hofmann-Wellenhof et al. [1994], Seeber [1989], Bauer [1992] und Wells et al. [1986] finden. Darüberhinaus wird in Dana [1996] auf die Eignung des GPS für die Zeitsynchronisation in Echtzeitsystemen eingegangen. Eine wichtige Quelle, insbesondere für Entwickler von Empfängern ist NAV [1992], die Spezifikation des GPS für zivile Nutzer.

3.2 Betreiber

Das GPS-Programm wurde 1973 in den USA gestartet. Entwickler und Eigner dieses Systems ist das amerikanische Verteidigungsministerium *(Departement of Defense, DoD)*. Obwohl das System anfänglich lediglich für militärische Zwecke genutzt werden sollte, wurde das DoD vom amerikanischen Kongreß unter Führung des Präsidenten Reagan angewiesen, auch dessen zivile Nutzung zu ermöglichen. Mittlerweile ist der Markt für zivile Nutzer des GPS zu einem wichtigen Wirtschaftsfaktor geworden.

3.3 Aufbau

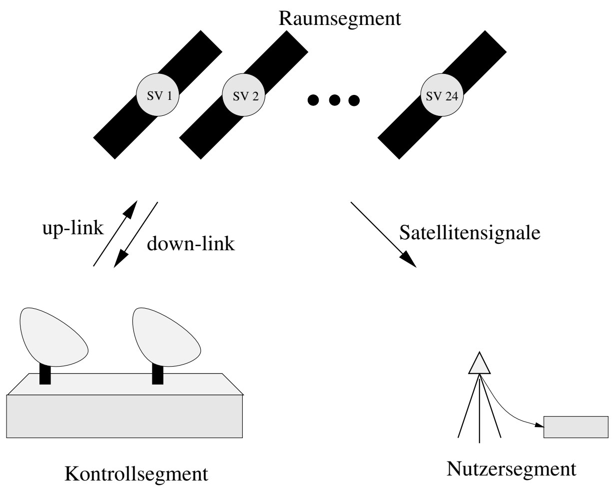

Abbildung 3.1: Die drei Bestandteile des GPS-Systems

GPS wird üblicherweise in drei Segmente eingeteilt: Das *Raumsegment* umfaßt alle Satelliten des GPS-Systems. Mit dem *Kontrollsegment* werden die auf der Erdoberfläche befindlichen Kontrollstationen zusammengefaßt, die für die Funktionsfähigkeit des GPS notwendig sind. Das *Nutzersegment* schließlich enthält alle GPS-Empfangsgeräte, die die von den Satelliten ausgesandten Signale empfangen, siehe Abbildung 3.1.

3.3.1 Raumsegment

Das Raumsegement besteht aus den zur Zeit (Daten vom 1. März 1996) 25 Satelliten in sechs Orbitebenen. Diese Orbitebenen sind gegeneinander um jeweils 60° verdreht und haben alle einen Inklinationswinkel (Winkel, unter dem die Satelliten den Äquator queren) von 55°. Die Umlaufszeit eines jeden Satelliten beträgt 12 siderische Stunden (Stunden nach der Sternzeit), das sind entsprechend 11 h 58 min 2 sec.

Es gibt mehrere Satellitentypen, die sich in ihrer Leistungsfähigkeit unterscheiden:

Block I-Satelliten

Die 11 Block-I Satelliten mit SVN (Satellite Vehicle Number) 1–11 wurden in den Jahren 1978 bis 1985 gestartet. Diese Satelliten befinden sich allerdings in anderen Orbitebenen wie die später verwendeten Block II-Satelliten, da sie einen Inklinationswinkel von 63° besitzen. Sie sind jeweils mit einer Cäsium– und zwei Rubidiumatomuhren ausgestattet. Konzipiert wurden diese Satelliten für eine Lebensdauer von 5 Jahren, ihre tatsächliche Lebenserwartung war aber weit höher. Dieser Satellitentyp wurde zur Erprobung des GPS-Konzepts entwickelt (von der Firma Rockwell).

Ein einziger der Block I-Satelliten (SVN 10) ist noch in Betrieb.

Block II,IIA,IIR,IIF-Satelliten

Die 9 Block II Satelliten mit SVN 13–21 wurden im Zeitraum vom Februar 1989 bis Oktober 1990 gestartet. Sie sind dafür ausgelegt, bis zu 14 Tage lang ohne Kontakt zum Kontrollsegment betriebsfähig zu sein.

Die 19 Block IIA (A für Advanced) Satelliten (SVN 22-40), von denen allerdings bisher nur 15 gestartet wurden (seit November 1990), sind für einen Zeitraum von 180 Tagen in der Lage ohne Kontakt zum Kontrollsegment auszukommen. Die dabei allerdings unvermeidlich auftretende Verringerung der Genauigkeit wird im Navigationscode des Satellitensignals vermerkt. Seit dem Start des letzten Block IIA-Satelliten war die vollständige Konfiguration von 24 Satelliten erreicht. Der Start weiterer Satelliten erfolgt nach Bedarf.

Beide Satellitentypen Block II und IIA wurden von Rockwell International gefertigt und haben eine spezifizierte Lebensdauer von 7,5 Jahren. Sie sind jeweils mit zwei Cäsium– und zwei Rubidiumatomuhren ausgestattet und sind außerdem zur Ausführung der *Selective Availability* und *Anti-Spoofing*–Operationen fähig (siehe Abschnitt 3.6.3 und 3.6.4).

Die nächste Generation von Satelliten, die Block-IIR (R für Replenishment) Satelliten (gefertigt von Gerneral Electrics) mit den SVN 41–66 sollen in der Lage sein, 14 Tage ohne Kontakt zum Kontrollsegment unter voller Betriebsfähigkeit auszukommen. Im *autonomen Navigationsmodus* soll dieser Zeitraum auf 180 Tage ausgedehnt werden können (unter voller Genaugikeit). Dies soll durch Kommunikation zwischen den Block-IIR-Satelliten erreicht werden. Die spezifizierte Lebensdauer eines Block-IIR-Satelliten ist 7,5 Jahre. Sie werden vermutlich mit Wasserstoff-Masern ausgerüstet.

In Planung befinden sich derzeit die Block-IIF (F für Future oder Follow On) Satelliten, die im nächsten Jahrtausend gestartet werden sollen.

3.3.2 Kontrollsegment

Das *Kontrollsegment (Control Segment, CS)* besteht aus mehreren, über die ganze Welt verstreuten Stationen. Es ist zuständig für die Bestimmung der genauen Orbitalpositionen der Satelliten, die Synchronisation der Satellitenuhren und die Übertragung der gewonnenen Daten an die Satelliten. Allerdings ist das Kontrollsegment auch für die Wartung des Systems verwantwortlich; darunter fällt auch die regelmäßige Wartung der Satelliten (Repositionierung im Orbit, Wartung der Atomuhren).

Master Control Station

Die *Master Control Station* ist das Herzstück des Kontrollsegments. Es ist im Consolidated Space Operations Center (CSOC) in Falcon Air Force Base (Falcon AFB), Colorado Springs in Colorado untergebracht. Es speichert die Daten der *Monitor Stations* und berechnet daraus die genauen Orbit– und Uhrendaten der Satelliten. Diese werden dann an die *Ground Control Stations* übermittelt, die diese Daten über eine Funkverbindungen zu den Satelliten übertragen.

Monitor Station

Es gibt derzeit fünf *Monitor Stations*, von denen vier sich auf kleinen Inseln in der Nähe des Äquators befinden (Hawaii, Ascension im Südatlantik, Diego Garcia im Indischen Ozean und Kwajalein im Nordpazifik) und die *Master Control Station* in Falcon AFB, die gleichzeitig auch als *Monitor Station* fungiert. Ihre Aufgabe ist die Sammlung von Meßwerten der Satelliten. Dabei werden P-Code Empfänger eingesetzt, um die Bahndaten und auch die Uhrendaten der Satelliten zu ermitteln. Diese Daten werden mit Hilfe von Meßdaten über die Atmosphäre und die Ionosphäre korrigiert und anschließend an die *Master Control Station* übermittelt.

Ground Control Stations

Die drei *Ground Control Stations* übernehmen die Kommunikation mit den Satelliten. Sie befinden sich bei den Monitorstationen in Ascension, Diego Garcia und Kjawalein. Mit Hilfe großer Bodenantennen werden die Informationen, die die Benutzer schließlich in ihren Empfängern erhalten, hinaufgeladen (upload). Um die Sicherheit des ganzen Systems zu erhöhen (Beispielsweise bei Ausfall einer *Ground Control Station*), wird versucht, die regelmäßige Übertragung neuer Bahn– und Uhrendaten in immer längeren Intervallen stattfinden zu lassen (Derzeit einmal pro Tag und Satellit).

3.3.3 Nutzersegment

In das *Nutzersegment* fallen alle Empfänger, die die Satellitensignale empfangen. Hierbei lassen sich die Anwendungsbereiche, wie anfangs bereits erwähnt, in die drei Hauptbereiche einteilen: Navigationsanwendungen, geodätische Anwendungen und Zeitanwendungen. Den Nutzern stehen dabei zwei Arten von Diensten zur Verfügung: Der *Precise Positioning Service (PPS)* und der *Standard Positioning Service (SPS)*:

Precise Positioning Service, PPS

PPS können nur jene benutzen, die im Besitze eines geheimen Codes sind. Um Zugang zu diesem Code erhalten zu können, müssen gewisse Bedingungen erfüllt sein:

- Es muß im nationalen Interesse der Vereinigten Staaten sein
- Gewisse Sicherheitsanforderungen können erfüllt werden
- Es gibt keine vernünftige Alternative zur Nutzung des PPS

Diese Bedingungen sind für zivile Nutzer, im besonderen für Nicht-Amerikaner schwer zu erfüllen. Deshalb wird man sich gewöhnlich auf die Nutzung des SPS-Dienstes mit seiner minderen Genauigkeit beschränken müssen.

Im *Federal Radionavigation Plan (FRP)* wird die Genauigkeit des PPS wie folgt beschrieben:

> The Precise Positioning Service is defined as the most accurate direct positioning, velocity and timing information continously available, worldwide, from the basic GPS. This service ist limited to users specifically authorized by the U.S. P(Y)-code capable military user equipement and provides predictable positioning accuracy of at least 22 metres (95 percent) horizontally and 27.7 metres (95 percent) vertically and time transfer accuracy to UTC within 200 nanoseconds (95 percent)

Standard Positioning Service, SPS

Der SPS ist frei zugänglich, besitzt aber eine wesentlich verringerte Genauigkeit. Das folgende Zitat aus dem FRP beschreibt die Genauigkeit des SPS:

> The Standard Positioning Service is defined as the standard specified level of positioning an timing accuracy that is available, without restrictions, to any user on a continous worldwide basis. The accuracy of this service will be established by the DoD and DoT[1] based on U.S. security interests. SPS provides a predictable positioning accuracy of 100 metres (95 percent) horizontally and 156 metres (95 percent) vertically and time transfer accuracy to UTC within 340 nanoseconds (95 percent).

[1] Departement of Transportation

Die durch die verringerte Genauigkeit vorhandene militärische Sicherheit ist durch den Bau von DGPS-Referenzstationen längst untergraben worden, siehe Abschnitt 3.10.1.

3.4 Signalstruktur der Satellitenaussendungen

Jeder Satellit sendet auf den beiden L–Band-Trägerfrequenzen $L_1 = 1{,}57542$ GHz und $L_2 = 1{,}2276$ GHz. Diese werden von der *Fundamentalfrequenz* $f_0 = 10{,}23$ MHz der Satelliten abgeleitet. Auf diesen Trägerfrequenzen sind mehrere Binärcodes in der sogenannten Biphasenmodulationstechnik *(Bi-phase shift keying, BPSK)* aufmoduliert. Die verwendete BPSK-Modulation läßt sich auch als Amplitudenmodulation verstehen, bei der die Amplitude entweder -1 oder 1 ist. Zuvor müssen allerdings die Binärcodes, die mit den Ziffern 0 und 1 dargestellt werden entsprechend umgewandelt werden ($0 \to 1$, $1 \to -1$).

Die auftretenden Binärcodes und ihre Raten sind in Tabelle 3.1 wiedergegeben.

Signal	Formelzeichen	Bitrate (Bit/s)
Navigationscode	$D(t)$	50
C/A-Code	$G_i(t)$	1.023.000
P(Y)-Code	$P_i(t)$ oder $Y_i(t)$	10.023.000

Tabelle 3.1: Die Binärcodes des GPS

Der Navigationscode $D(t)$ ist ein Bitstrom, der wichtige Daten über das Satellitensystem enthält. Der C/A-Code und der P(Y)-Code enthalten hingegen keine Daten im eigentlichen Sinn; Sie sind sogenannte periodische Pseudo-Random-Noise–Codes (PRN-Codes), die Ähnlichkeiten mit zufälligen Bitfolgen aufweisen.

Der Zweck dieser Codes ergibt sich aus folgendem: Angenommen, ein Satellit und ein Empfänger hätten zwei Uhren, die exakt synchronisiert sind. Strahlt der Satellit nun einen PRN-Code $G(t)$ aus, so benötigt das Signal eine gewisse Laufzeit, bis es bei der Antenne des Empfängers ankommt. Generiert der Empfänger nun synchron zum Satelliten genau den gleichen Code $G(t)$, so wird dieser zu dem vom Satelliten empfangenen um die Laufzeit zeitlich verschoben sein (siehe Abbildung 3.2). Um diese zeitliche Verschiebung zu ermitteln, kann man nun eine besondere Technik anwenden: Die *Code-Korrelation*. Hierbei wird die Eigenschaft der PRN-Codes verwendet, daß die Autokorrelationsfunktion eines solchen Codes $G(t)$ (T ist eine hinreichend lange Mittelungszeit)

$$A(\tau) = \int_{-T}^{T} G(t)G(t+\tau)\mathrm{d}t. \tag{3.1}$$

ein scharfes Maximum für $\tau = 0$ besitzt und für alle anderen Werte von τ nahezu verschwindet. Zur Ermittlung der Laufzeit muß der Empfänger das empfangene Signal mit dem generierten Signal kreuzkorrelieren (das heißt multiplizieren und hinreichend lange mitteln). Verschiebt er nun das generierte Signal solange, bis der Maximums-Peak der Korrelation erreicht ist, so decken sich empfangenes und generiertes Signal genau. Die erforderliche Verschiebung stimmt dann mit der Laufzeit überein.

Mit dieser Technik läßt sich die Verschiebung auf besser als 1 % der Bitlänge des Codes bestimmen. Darüberhinaus ist diese Technik überaus störsicher gegenüber Rauschsignalen, da diese bei der Korrelation herausgemittelt werden. Die Codes stellen damit nichts anderes als eine Art „kontinuierliche“ Zeitmarke dar, dessen Lage technisch sehr genau bestimmt werden kann (Beim C/A-Code üblicherweise auf etwa 3 ns genau).

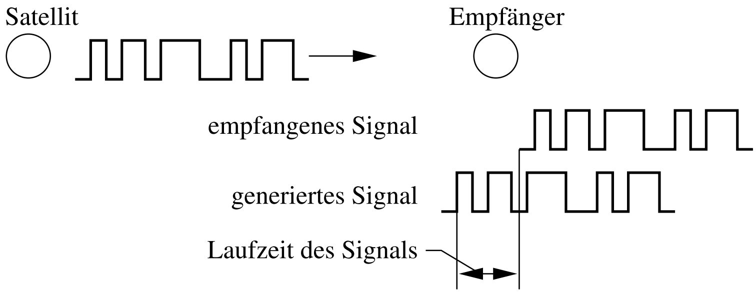

Abbildung 3.2: Laufzeit vom Satelliten zum Empfänger

3.4.1 Der zivile C/A-Code

Der zivile C/A-Code (*Coarse Aquisition*) $G_i(t)$ besteht in Wirklichkeit aus 36 verschiedenen Codes ($1 \leq i \leq 36$), die alle aus jeweils 1023 Bits bestehen. Von diesen stehen für den normalen Betrieb nur 32 zur Verfügung. Da die Bitrate 1,023 MHz beträgt, wiederholt sich ein solcher Code alle 1 ms. Ein Chip (das ist ein Bit des Codes) dauert somit etwa 1 μs, das entspricht 300 m. Wie der C/A-Code genau generiert wird, ist in NAV [1992] zu finden. Der C/A-Code wird im Normalbetrieb gemeinsam mit dem Navigationscode nur auf die Trägerfrequenz L_1 aufmoduliert und ist Grundlage des SPS.

3.4.2 Der militärische P(Y)-Code

Der militärische Code setzt sich aus zwei Gruppen von Codes zusammen: Der bekannte P-Code und der verschlüsselte Y-Code. Ist das sogenannte *Anti-Spoofing (A-S)* aktiviert, so wird der geheime Y-Code verwendet, ansonsten der öffentliche P-Code. Dies hat den Sinn, daß — sofern A-S aktiv ist — niemand der den Code nicht kennt, in der Lage ist, das Satellitensignal zu imitieren und damit millitärische Einheit fehlzuleiten. Ähnlich wie beim C/A-Code existieren in Wirklichkeit jeweils 37 verschiedene Einzelcodes $P_i(t)$ und $Y_i(t)$, die durch den Index i (die PRN-Nummer) unterschieden werden. Im Normalbetrieb werden nur die ersten 32 Codes den einzelnen Satelliten zugeordnet. Die Periode jeder der Codes ist exakt eine Woche (604.800 s). Die Bitrate des P(Y)-Codes beträgt 10,23 MHz (Das 10-fache des C/A-Codes). Die Dauer eines Chips ist daher 100 ns; das entspricht 30 m.

Im Gegensatz zum C/A-Code wird der P(Y)-Code (gemeinsam mit den Navigationscode) auf beide Frequenzen L_1 und L_2 aufmoduliert. Diese Übertragung auf zwei Frequenzen hat den Sinn, die durch die Ionosphäre eingebrachte Signalverzögerung messen zu können. Dies ist möglich, da sich die Signalverzögerung mit der Frequenz ändert.

Der P(Y)-Code ist gemeinsam mit dem C/A-Code Grundlage des PPS. Der PPS ist genau dann von jedermann verwendbar wenn, A-S und SA[2] deaktiviert sind. Leider ist Anti-Spoofing seit dem 31. Jänner 1994 ununterbrochen in Betrieb.

3.4.3 Navigationscode

Der Navigationscode enthält die wichtige *Navigation Message*. In ihr sind wichtige Parameter, wie Bahn– und Uhrendaten der Satelliten enthalten. Diese Message wird regelmäßig in die Satelliten hochgeladen.

[2] Bei aktivem SA werden die Bahn- und Uhrendaten derart verfälscht, daß nur jene, die im Besitze des geheimen Codes sind die ursprünglichen unverfälschten Daten rekonstruieren können.

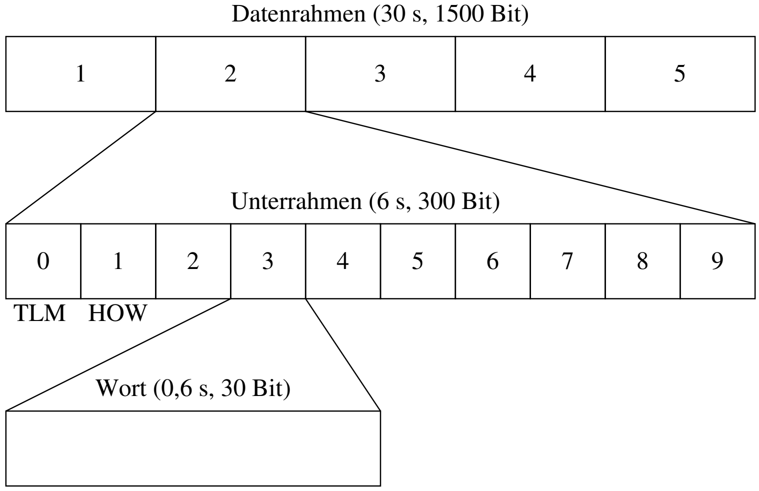

Abbildung 3.3: Struktur des Navigationscodes

Jeder Satellit sendet seine eigene Navigation Message mit einer Bitrate von 50 Bit/s. In Zeitraum von 30 s wird der 1500 Bit umfassende Datenrahmen periodisch gesendet (siehe Abbildung 3.3). Er besteht selbst aus 5 Unterrahmen, die in jeweils 6 s übermittelt werden. Die Unterrahmen schließlich bestehen aus 10 Worten zu je 30 Bits. Sie enthalten übrigens Paritätsbits zur Fehlererkennung und –korrektur.

Im Unterrahmen 1 sind hauptsächlich die Uhrendaten des betreffenden Satelliten untergebracht, die Unterrahmen 2 und 3 enthalten die Bahndaten des Satelliten. Die restlichen beiden Unterrahmen 4 und 5 durchlaufen zyklisch jeweils eine Folge von 25 verschiedenen Unterrahmen, die Korrekturparameter für die UTC-Zeit und ionosphärische Ausbreitungsverzögerung, Satellitenstatusdaten und auch den sogenannten *Almanach* enthalten, der die ungefähren Bahn– und Uhrendaten aller Satelliten beschreibt.

Die detaillierte Struktur der Navigation Message kann in NAV [1992] nachgeschlagen werden.

3.4.4 Signalstruktur

Die Trägerfrequenzen werden gemäß folgender Gleichung moduliert (A_p, A_c und B_p sind konstante Amplitudenwerte für die Modulationen; Die Binärcodes $P_i(t)$, $G_i(t)$ und $D(t)$ liegen hierbei bereits als Folge -1 und 1 vor):

$$\begin{aligned} \mathrm{L}_1(t) &= A_p P_i(t) D(t) \cos(2\pi f_1 t) + A_c G_i(t) D(t) \sin(2\pi f_1 t). \\ \mathrm{L}_2(t) &= B_p P_i(t) D(t) \cos(2\pi f_2 t). \end{aligned}$$

Wie aus den Gleichungen ersichtlich, sind die Binärcodes für P-Code und C/A-Code um 90°phasenverschoben, um die gegenseitige Beeinflussung zusätzlich zu verringern. Jeder Satellit muß einen anderen Code abstrahlen, daher ist sowohl der P-Code als auch der C/A-Code eigentlich ein Satz von Codes, die durch den Index i — die PRN-Nummer — unterschieden werden ($1 \le i \le 32$). Die Bitraten der einzelnen Codes werden aus der Fundamentalfrequenz $f_0 = 10{,}23$ Mhz generiert. Entsprechend werden die Trägerfrequenzen durch Frequenzvervielfachung gewonnen ($\mathrm{L}_1 = 154 \cdot 10{,}23$ MHz und $\mathrm{L}_2 = 120 \cdot 10{,}23$ MHz). Daher sind Trägerfrequenzen und Codes zum Zeitpunkt der Aussendung zueinander kohärent.

3.5 Funktionsweise des GPS

Das Funktionsprinzip von GPS basiert auf dem Einwegmeßverfahren. Das heißt, daß die Funksignale nur in einer Richtung, vom Satelliten zum GPS-Empfänger ausgesandt werden. Im Gegensatz zum Zweiwegmeßverfahren muß der Nutzer daher keine Funksignale zum Satelliten ausstrahlen, sodaß dessen Standort nicht durch Funkpeilung eruiert werden kann (wichtig in der militärischen Nutzung). Ein zweiter Vorteil, der sich daraus ergibt, ist die kleine Antenne und die praktisch unbegrenzte Teilnehmerzahl.

3.5.1 Funktionsprinzip

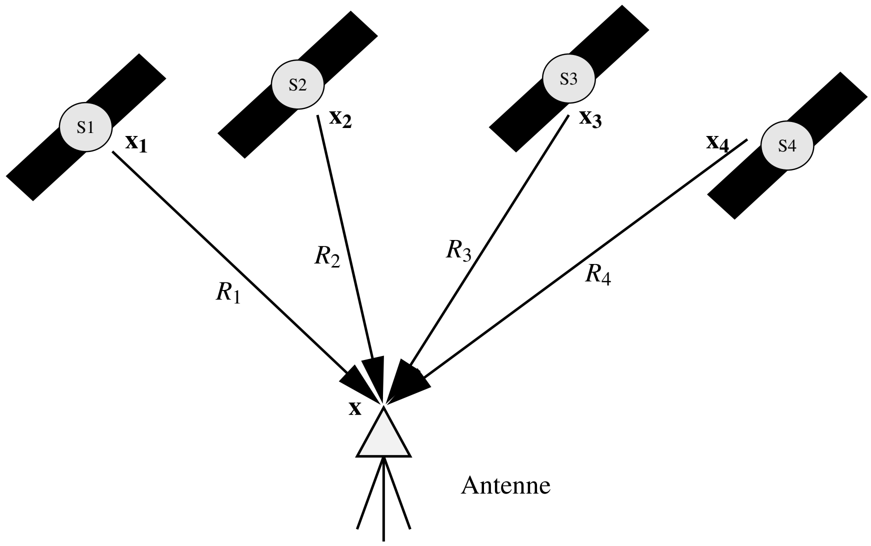

Abbildung 3.4: Navigationslösung mit 4 Satelliten

Zunächst soll das Funktionsprinzip anhand eines vereinfachten Modells dargestellt werden. Die zunächst getroffenen vereinfachenden Annahmen sind folgende: Die Uhren der Satelliten seien exakt miteinander synchronisiert. Die zugrundegelegte Zeitskala sei die GPS-Systemzeit. Die Positionen $\mathbf{x_i}$ der Satelliten seien darüberhinaus genau bekannt. Weiterhin soll angenommen, daß sich die Signale geradlinig und ungestört im Vakuum ausbreiten und relativistische Effekte vernachlässigt werden können. Die anschließenden Beschreibungen in Abschnitt 3.6 werden das vorgestellte Modell präzisieren.

Die Satellitensignale enthalten eine Art von Zeitmarkierungen, aus denen zurückgeschlossen werden kann, wann ein bestimmter Abschnitt — relativ zur GPS-Systemzeit — abgesandt wurde. Eine solche Zeitmarkierung wandert nun mit Lichtgeschwindigkeit c vom Satelliten zum Empfänger und benötigt für diese Strecke R *(range)* eine Laufzeit von $\tau = \mathrm{R}/c$. Wäre der Empfänger mit einer Uhr ausgestattet, die exakt mit den Satellitenuhren synchronisiert wäre, so könnte die Laufzeit τ einfach gemessen werden. Stattdessen wird die Empfängeruhr aber einen Uhrenfehler Δt_u gegenüber den Satellitenuhren aufweisen, weswegen statt der wahren Laufzeit nur die mit dem Empfängeruhrfehler *(clock bias)* verfälschte Laufzeit τ' gemessen werden kann:

$$\tau' = \tau + \Delta t_u. \tag{3.2}$$

Multipliziert man die Gleichung mit der (Vakuum-) Lichtgeschwindigkeit $c = 299.792.458{,}000\,\mathrm{m/s}$ so erhält man

$$c\tau' = c\tau + c\Delta t_u \tag{3.3}$$

Die Größe $c\tau'$ bezeichnet man auch als *Pseudorange* PR, da sie die durch den Uhrenfehler verfälschte Wegstrecke R darstellt. Die Gleichung lautet daher auch:

$$\mathrm{PR} = \mathrm{R} + c\Delta t_u. \tag{3.4}$$

Schreibt man diese Gleichung für vier Satelliten an, so erhält man folgendes Gleichungssystem:

$$\begin{aligned} \mathrm{R}_1 + c\Delta t_u &= \mathrm{PR}_1 \\ \mathrm{R}_2 + c\Delta t_u &= \mathrm{PR}_2 \\ \mathrm{R}_3 + c\Delta t_u &= \mathrm{PR}_3 \\ \mathrm{R}_4 + c\Delta t_u &= \mathrm{PR}_4. \end{aligned}$$

Berücksichtigt man die Formel $\mathrm{R}_i = |\mathbf{x} - \mathbf{x_i}|$, wobei $\mathbf{x}$ die unbekannte Position der Antenne und $\mathbf{x_i}$ die Position des i-ten Satelliten ist, so schreibt sich das Gleichungssystem folgendermaßen:

$$\begin{aligned} |\mathbf{x} - \mathbf{x_1}| + c\Delta t_u &= \mathrm{PR}_1 \\ |\mathbf{x} - \mathbf{x_2}| + c\Delta t_u &= \mathrm{PR}_2 \\ |\mathbf{x} - \mathbf{x_3}| + c\Delta t_u &= \mathrm{PR}_3 \\ |\mathbf{x} - \mathbf{x_4}| + c\Delta t_u &= \mathrm{PR}_4. \end{aligned}$$

Da die Pseudoranges PR_i sich als Meßwerte ergeben und Positionen der Satelliten $\mathbf{x_i}$ voraussetzungsgemäß bekannt sind, bleiben als Unbekannte nur die drei Koordinaten des Antennenortsvektors $\mathbf{x} = (x, y, z)$ und der Uhrenfehler Δt_u über. Diese können direkt aus diesem <u>nichtlinearen</u> Gleichungssystem mit 4 Gleichungen gelöst werden. Üblicherweise löst man das System allerdings mit iterativen Verfahren. Die Darstellung eines solchen führt auf einige wichtige Begriffe:

Faßt man die vier Unbekannten als einen 4-dimensionalen Vektor $\mathbf{X} = (x_u, y_u, z_u, \Delta t_u)$ zusammen und tut das selbe für die vier Pseudoranges $\mathbf{PR} = (\mathrm{PR}_1, \mathrm{PR}_2, \mathrm{PR}_3, \mathrm{PR}_4)$, so läßt sich das obige nichtlineare Gleichungssysteme kompakt als eine Gleichung formulieren:

$$\mathbf{PR} = \mathbf{PR}(\mathbf{X}). \tag{3.5}$$

Kennt man einen Näherungswert $\mathbf{X_0}$ für die Unbekannten, so läßt sich die Gleichung lokal linearisieren:

$$\mathbf{PR}(\mathbf{X}) \approx \mathbf{PR}(\mathbf{X_0}) + \frac{\partial \mathbf{PR}}{\partial \mathbf{X}}(\mathbf{X} - \mathbf{X_0}). \tag{3.6}$$

Führt man den reduzierten Pseudorangevektor $\mathbf{l} = \mathbf{PR}(\mathbf{X}) - \mathbf{PR}(\mathbf{X_0})$ und den reduzierten Distanzvektor $\mathbf{x} = \mathbf{X} - \mathbf{X_0}$ ein, so lautet obige Gleichung ($\mathbf{A} = \frac{\partial \mathbf{PR}}{\partial \mathbf{X}}$):

$$\mathbf{l} = \mathbf{A}\mathbf{x}. \tag{3.7}$$

Die Lösung der Gleichung lautet:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{l}. \tag{3.8}$$

Dieser Wert $\mathbf{x}$ läßt sich als Korrekturwert benutzen, um zu einer besseren Lösung $\mathbf{X_0'} = \mathbf{X_0} + \mathbf{x}$ zu gelangen. Das Verfahren läßt sich beliebig oft wiederholen, so daß man der exakten Lösung der Gleichung immer näher kommt. Gleichung (3.8) zeigt auch, wie sensibel der Ergebnisvektor $\mathbf{X_0}$ von Fehlern in den Pseudoranges abhängt. Seien die Meßfehler in den Pseudoranges durch den Vektor $\varepsilon_\mathbf{R}$ dargestellt, so ergibt sich ein entsprechender Fehler beim Vektor für die Unbekannten $\varepsilon_\mathbf{X}$:

$$\varepsilon_\mathbf{X} = \mathbf{A}^{-1}\varepsilon_\mathbf{R}. \tag{3.9}$$

Gewöhnlich wird man die Fehler in den Pseudoranges durch Varianzen oder Standardabweichungen beschreiben. Es stellt sich dann die interessante Frage, wie sich die korrespondierenden Varianzen für die Fehler der Koordinaten und des Uhrenfehlers darstellen. Die Varianzen der Pseudoranges stellt man durch die Kovarianzmatrix dar (es wird dabei angenommen, daß der Erwartungswert der Meßfehler Null sei):

$$\mathbf{Cov}(\varepsilon_\mathbf{R}) = \mathrm{E}(\varepsilon_\mathbf{R}\varepsilon_\mathbf{R}{}^\mathrm{T}). \tag{3.10}$$

Mit E wird hier der Erwartungswert gekennzeichnet. Die Kovarianzmatrix enthält in den Diagonalelementen die Varianzen der entsprechenden Koordinaten. Die Nebenelemente kennzeichnen die Kovarianz zwischen verschiedenen Pseudoranges. Sie sind Null, sobald die Pseudoranges statisch unabhängig voneinander variieren. Entsprechend läßt sich auch eine Kovarianzmatrix für die Unbekannten definieren:

$$\mathbf{Cov}(\varepsilon_\mathbf{X}) = \mathrm{E}(\varepsilon_\mathbf{X}\varepsilon_\mathbf{X}{}^\mathrm{T}). \tag{3.11}$$

Gleichung (3.9) ermöglicht uns, eine Beziehung zwischen den beiden Kovarianzmatrizen herzustellen:

$$\mathbf{Cov}(\varepsilon_\mathbf{X}) = \mathbf{A}^{-1}\mathbf{Cov}(\varepsilon_\mathbf{R})\mathbf{A}^{-\mathbf{T}}. \tag{3.12}$$

Nimmt man nun an, daß die Varianzen der einzelnen Pseudoranges statistisch voneinander unabhängig und alle gleich σ_r^2 sind, so ist die Kovarianzmatrix der Pseudoranges eine Diagonalmatrix mit Diagonalelementen σ_r^2. Die Kovarianzmatrix der Ergebniswerte lautet dann:

$$\mathbf{Cov}(\varepsilon_\mathbf{X}) = \sigma_r^2(\mathbf{A}^\mathbf{T}\mathbf{A})^{-1}. \tag{3.13}$$

Ausführlich geschrieben:

$$\mathbf{Cov}(\varepsilon_\mathbf{X}) = \sigma_r^2 \begin{pmatrix} q_{xx} & q_{xy} & q_{xz} & q_{xt} \\ q_{yx} & q_{yy} & q_{yz} & q_{yt} \\ q_{zx} & q_{zy} & q_{zz} & q_{zt} \\ q_{tx} & q_{ty} & q_{tz} & q_{tt}. \end{pmatrix} \tag{3.14}$$

Die Werte q_{xx}, q_{yy}, q_{zz}, q_{tt} stellen die Varianzen in den einzelnen Koordinatenrichtungen dar. Die Varianz des Positionsfehlers ergibt sich damit zu

$$\sigma_P^2 = \sigma_r^2(q_{xx} + q_{yy} + q_{zz}). \tag{3.15}$$

Ist das zugrundeliegende Koordinatensystem überdies so aufgebaut, daß die x-Achse nach Norden, die y-Achse nach Osten und die z-Achse zum Zenit des ermittelten Punktes auf der Erdoberfläche hin gerichtet ist, was durch geeignete Koordinatentransformation erreicht werden kann, so können die Varianzen in horizontaler Richtung σ_H^2 (xy-Ebene) in vertikaler Richtung σ_V^2(z-Achse) und natürlich in der Zeit σ_T^2 (Zeitkoordinate) einfach berechnet werden:

$$\begin{aligned} \sigma_H^2 &= \sigma_r^2(q_{xx} + q_{yy}) \\ \sigma_V^2 &= \sigma_r^2 q_{zz} \\ \sigma_T^2 &= \sigma_r^2 q_{tt}. \end{aligned}$$

In Navigationssystemen benutzt man oft die sogenannte *Dilution of Precision (DOP)*. Sie ist durch den Zusammenhang zwischen der Standardabweichung einer Größe und denen der Pseudoranges definiert:

$$\sigma = \mathrm{DOP} \cdot \sigma_r. \tag{3.16}$$

Entsprechend lassen sich DOP-Werte für die horizontale, vertikale und 3D-Position und die Zeit bestimmen:

$$\begin{aligned}
\sigma_H &= \mathrm{HDOP} \cdot \sigma_r \\
\sigma_V &= \mathrm{VDOP} \cdot \sigma_r \\
\sigma_P &= \mathrm{PDOP} \cdot \sigma_r \\
\sigma_T &= \mathrm{TDOP} \cdot \sigma_r.
\end{aligned}$$

Man definiert allerdings auch einen kombinierten Wert für Zeit und Position *General Dilution of Precision (GDOP)* wie folgt:

$$\mathrm{GDOP} = \sqrt{\mathrm{PDOP}^2 + \mathrm{TDOP}^2}. \tag{3.17}$$

Die einzelnen DOPs beschreiben, wie sich die momentane Satellitenkonstellation $\mathbf{x_i}$ auf die Genauigkeit der betroffenen Größe (Horizontalposition, Zeit, usw.) auswirkt. Sie lassen sich leicht ausrechnen:

$$\begin{aligned}
\mathrm{HDOP} &= \sqrt{q_{xx} + q_{yy}} \\
\mathrm{VDOP} &= \sqrt{q_{zz}} \\
\mathrm{PDOP} &= \sqrt{q_{xx} + q_{yy} + q_{zz}} \\
\mathrm{TDOP} &= \sqrt{q_{tt}} \\
\mathrm{GDOP} &= \sqrt{q_{xx} + q_{yy} + q_{zz} + q_{tt}}.
\end{aligned}$$

3.5.2 Bestimmung der Geschwindigkeit der Antenne — genaue Frequenz

Die Satellitensignale erfahren durch die Bewegung der Satelliten gegenüber der Antenne eine Dopplerverschiebung. Wesentlich ist hierbei die Geschwindigkeit in Richtung der Verbindungsachse Antenne–Satellit (radiale Geschwindigkeit). Da die radiale Geschwindigkeit der Satelliten zu jedem Zeitpunkt bekannt ist (aus den Bahndaten), kann aus der Dopplerverschiebung auf die radiale Geschwindigkeit der Antenne geschlossen werden. Besäße man überdies eine hochgenaue Referenzfrequenz, könnte man mit 3 Satelliten die 3 Geschwindigkeitskoordinaten der Antenne berechnen. Da die momentane Frequenz der Empfängeruhr allerdings nicht bekannt ist, verwendet man ähnlich wie für die Position einen vierten Satelliten und erhält schließlich zusätzlich zu den 3 unbekannten Geschwindigkeitskoordinaten den unbekannten Frequenzfehler der Empfängeruhr (4 Gleichungen, 4 Unbekannte).

3.5.3 Lösungen mit weniger als vier Satelliten

Die normale Lösung der Positions– und Zeitbestimmung erfolgt mit vier Satelliten (3D-Lösung). Der Grund liegt natürlich darin, daß zur Ermittlung aller vier Unbekannten (Position und Zeit) in Gleichung (3.5) vier Gleichungen erforderlich sind. Kennt man allerdings eine oder einige der Bestimmungsgrößen oder auch eine oder mehrere zusätzliche Gleichungen, so kann man mit entsprechen weniger Satelliten das Auslangen finden.

2D-Lösung

Ist die Höhe der Antenne mit guter Genauigkeit bekannt, so kann die horizontale Position als auch die Zeit mit 3 Satelliten bestimmt werden. Ein typischer Anwendungsfall wäre die Navigation auf dem Meer. Dem Empfänger muß lediglich die Höhe in GPS-Ellipsoidkoordinaten oder aber in Höhe über Meeresniveau (Mean Sea level, MSL) angegeben werden.

Time Transfer Lösung

Ist die GPS-Ellipsoid Position hinreichend genau bekannt, so kann durch Vermessung eines einzigen Satelliten die GPS-Zeit ermittelt werden. Vermißt man außerdem gleichzeitig mehrere Satelliten in diesem Modus, so kann durch Mittelung die Genauigkeit zustätzlich gesteigert werden.

Im statischen Fall (Geschwindigkeit gegen Erdoberfläche null) kann genauso — sofern der Empfänger das erlaubt — aus dem Vermessen eines einzelnen Satelliten eine genaue Frequenz abgeleitet werden.

3.6 Wichtige Einflußfaktoren im Satelliten

Der vorliegende und die folgenden Abschnitte präzisieren das in Abschnitt 3.5 beschriebene Modell. Hierbei folgt die Beschreibung dem Weg des Signals vom Satelliten über das Ausbreitungsmedium zum Empfänger.

3.6.1 Uhrendaten

Jeder Satellit besitzt eigene Atomuhren, die die Satellitenaussendungen steuern. Eine der Atomuhren jedes Satelliten definiert eine eigene Zeitskala, die als *SV-Zeit (Space Vehicle-Zeit)* des Satelliten bezeichnet wird. Sie ist nicht vollkommen synchron zur GPS Systemzeit, sondern driftet von ihr weg. Aus diesem Grund ist es erforderlich, die Abweichung der Uhren jedes Satelliten zu kennen. Diese Abweichung $\Delta t_{SV} = t_{SV} - t_{GPS}$ wird mit einem Polynom zweiten Grades modelliert:

$$\Delta t_{SV} = a_{f0} + a_{f1}(t_{GPS} - t_{oc}) + a_{f2}(t_{GPS} - t_{oc})^2 + \Delta t_r \tag{3.18}$$

Der Zeitpunkt t_{oc} ist der Anwendungszeitpunkt des Polynoms. Die Parameter a_{f0},a_{f1},a_{f2} und t_{oc} werden im Navigationscode übertragen. Der Term Δt_r hingegen ist eine relativistische Korrektur, die ein Empfänger korrekterweise anbringen muß. Sie wird im Abschnitt 3.6.5 besprochen.

3.6.2 Bahndaten — Ephemeriden

Wie aus dem Gleichungssytem (3.5) ersichtlich, sind zu dessen Lösung die genauen Positionen $\mathbf{x_i}$ der Satelliten erforderlich. Zu diesem Zweck sind im Navigationscode die Orbitalparameter (Ephemeriden) des entsprechenden Satelliten enthalten. Diese Ephemeriden sind so formuliert, daß die Satellitenpositionen für jeden Zeitpunkt im Gültigkeitszeitraum der Ephemeriden (1 Stunde) in dem geozentrischen Koordinatensystem *World Geodetic System 1984 (WGS-84)* berechnet werden können. Dieses Koordinatensystem ist fix mit dem Erdkörper verbunden und wird dementsprechend als *Earth Centered Earth Fixed (ECEF)*-Koordinatensystem bezeichnet.

Dem WGS-84 liegt ein Referenzellipsoid zugrunde, dessen Zentrum im Massenmittelpunkt der Erde liegt. Seine große Halbachse (Äquatorradius) beträgt $a = 6378137,00000$ m, die kleine Halbachse (Polradius) ist $b = 6356752,31425$ m. Koordinaten, wie sie im WGS-84 gegebenen sind, stimmen selten mit den Koordinaten, wie sie auf Landkarten gedruckt sind, überein. Diese basieren auf anderen Ellipsoiden. Speziell bei Höhenangaben kommt häufig die Angabe einer *Höhe über Meeresniveau* (*Mean Sea Level, MSL*) vor. Empfänger besitzen oft einprogrammierte Tabellen, die die Umrechnung von MSL auf WGS-84–Koordinatenhöhen erlauben (und vice versa).

Um die Genauigkeit der mit Hilfe der im Navigationscode enthaltenen Uhrendaten und Bahndaten zu bewerten, steht im Navigationscode auch der Parameter *User Range Accuracy (URA)* zur Verfügung. Er beschreibt den gemeinsamen Fehler der Pseudorange, der durch die Uhren– und Bahndaten verursacht wird, in Form eines 1 σ-Wertes. Für einen typischen Block II Satelliten mit aktiviertem SA — siehe Abschnitt 3.6.3 — beträgt dieser Wert 32 m.

3.6.3 Selective Availability, SA

Mit *Selective Availability (SA)* wird die Maßnahme bezeichnet, die die Genauigkeit des SPS gezielt verringert. Dies wird erreicht, indem einerseits die Uhr des Satelliten mit künstlichem Rauschen überlagert wird, und andererseits — dies ist der größere Beitrag zum Fehler — die Bahndaten verfälscht werden. Der durch SA verursachte Fehler ist die größte Fehlerursache im SPS. Für kurze Zeiten (kleiner 30 s) wirkt SA wie weißes Rauschen. Es besitzt allerdings auch über längere Zeiträume sich verändernde Beiträge (mehrere Stunden). Das Verhalten von SA wurde zwar spezifiziert, diese Spezifikation ist allerdings nicht öffentlich zugänglich.

SA ist seit 1. Juli 1991 ständig aktiv. Es gibt aber in letzter Zeit (März 1996) massive Bestrebungen hauptsächlich von Seiten der amerikanischen Industrie, SA abzuschaffen. Bisher ist allerdings noch keine endgültige Entscheidung gefallen.

3.6.4 Anti Spoofing, A-S

Das *Anti-Spoofing (AS)* ist jene Maßnahme, die den öffentlichen P-Code durch den verschlüsselten und geheimen Y-Code ersetzt. Anti-Spoofing ist seit dem 31. Jänner 1994 ständig aktiv.

3.6.5 Relativistische Korrekturen

Aufgrund der hohen Genauigkeit von GPS sind relativistische Korrekturen erforderlich. Wegen der relativ zur Lichtgeschwindigkeit nicht mehr vernachlässigbaren Geschwindigkeit des Satelliten und des zur Erdoberfläche beträchtlich unterschiedlichen Gravitationspotentials laufen zeitliche Vorgänge in Satellit und Erdoberfläche unterschiedlich schnell ab (Dies geschieht unabhängig vom Dopplereffekt). Daher wird die Frequenz der Atomuhr des Satelliten nicht mit dem Nominalwert von $f_0 = 10{,}230.000.000.000$ MHz betrieben, sondern mit dem modifzierten Wert von $f_0' = 10{,}229.999.995.43$ MHz. Damit erreicht man, daß der korrekte Nominalwert auf der Erdoberfläche empfangen wird.

Neben diesem konstanten Beitrag treten allerdings noch periodische Abweichungen auf, die sich etwa im Term Δt_r der Gleichung (3.18) äußern. Da die Bahn des Satelliten nicht exakt kreisförmig ist, schwanken die Geschwindigkeit und das Graviationspotential des Satelliten, was sich wiederum in entsprechenden Zeitabweichungen äußert.

Neben diesen Effekten tritt noch ein recht kleiner relativistischer Effekt auf, der durch die Raumzeitkrümmung auf dem Weg des Signals verursacht wird. Sein Betrag liegt in der Größenordnung vom Zentimetern und ist daher nur für Geodäten interessant.

3.7 Wichtige Einflußfaktoren im Ausbreitungsmedium

3.7.1 Ionosphärische Einflüsse

Die *Ionosphäre* ist der Bereich von 70 km bis 1000 km über der Erdoberfläche. In ihm wird die Ausbreitung von Wellen stark von freien Ladungsträgern (Ionen und Elektronen) beeinflußt. Deren Erzeugung wird durch die Einstrahlung der Sonne verursacht, weswegen die Ionosphäre zu Tagzeiten wesentlich dicker ist als während der Nacht.

Im Navigationscode werden Parameter α_i, β_i $(1 \leq i \leq 4)$ zur Verfügung gestellt, mit denen die Ionosphärischen Verzögerungen abgeschätzt werden können. Messungen haben ergeben, daß mit diesem Modell etwa die Hälfte des Fehlers eliminiert werden kann. Nach Anwendung des Modells muß man mit einem Fehler von 60–90 ns für Tagzeiten und 10–20 ns für Nachtzeiten rechnen.

Nutzer des PPS können die Tatsache ausnutzen, daß sich die Ionosphäre auf verschiedene Frequenzen auch unterschiedlich auswirkt. Deshalb wird der P(Y)-Code auf den beiden Frequenzen L_1 und L_2 ausgesandt. Durch Messung der Laufzeitdifferenz zwischen den beiden Codes kann somit auf die Auswirkung der Ionosphäre zurückgeschlossen und der Fehler stark reduziert werden.

3.7.2 Troposphärische Einflüsse

Die *Troposphäre* (dies ist der Bereich der Atmosphäre bis etwas 12 km Höhe in dem sich das Wettergeschehen abspielt) enthält im Gegensatz zur Ionosphäre keine freien Elektronen oder Ionen. Dafür ist dort die Luft sehr dicht. Es kommt daher zu einer kontinuierliche Brechung (Refraktion) der Radiowellen. Die sich ergebende Verzögerung wird einerseits durch den verlängerten Signalweg und die in Luft langsamere Lichtgesschwindigkeit verursacht. Da die Verzögerung von Druck, Temperatur und Feuchtigkeit der betroffenen Luftsäule abhängt, varriert sie örtlich und zeitlich relativ stark. Außerdem hängt sie vom Einfallswinkel in die Troposhäre ab. Signale, die von Satelliten nahe dem Horizont kommen, müssen nämlich einen viel größeren Weg in der Troposhäre durchlaufen, als solche von Satelliten im Zenit.

Das GPS selbst bietet keine Parameter für ein Modell zur Korrektur troposphärischer Fehler an. Der typische Fehler verursacht durch die Troposhäre beträgt etwa 30 ns. Durch Verwendung einfacher Standardmodelle für die Troposhäre kann der zurückbleibende Fehler auf etwa 10 ns reduziert werden.

3.8 Wichtige Einflußfaktoren beim Empfänger

3.8.1 Mehrwegausbreitung

Die Umgebung einer Antenne kann einen wesentlichen Fehlanteil verursachen. Werden Satellitensignale an planen Flächen reflektiert und mischen sich an der Antenne mit den direkt empfangenen, so führt dies zu Interferenz, wodurch es zu Fehlern in den Pseudoranges kommt, die in sehr ungünstigen Fällen bis zu 100 m betragen können. Gewöhnlich wird dieser Fehler bei sorgfältiger Aufstellung der Antenne kleiner als 10 m sein. Umgebende Objekte sind nicht nur in der Lage, das Phasenzentrum der Antenne um eine konstante Strecke zu verschieben, sondern bewirken auch, daß das Phasenzentrum mit der Einfallsrichtung der Satellitensignale variiert.

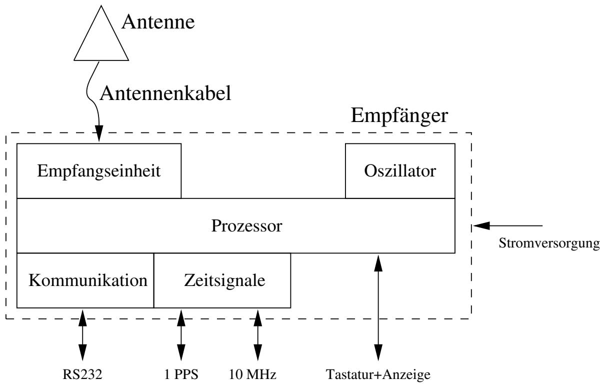

Abbildung 3.5: Aufbau eines GPS-Empfängers

3.8.2 Meßrauschen

Die Codekorrelation gelingt natürlich nicht perfekt. Es kommt zu Schwankungen, die üblicherweise zu etwa 3 ns angesetzt werden. Darüberhinaus können natürlich Laufzeitunterschiede zwischen Kanälen, die unterschiedliche Satelliten messen, auftreten. Durch Kalibirierung lassen sich diese elimieren. Zum Meßrauschen trägt auch die Instabilität der Empfängeruhr bei.

3.8.3 Satellitenkonstellation

Die Satellitenkonstellation hat einen wesentlichen Anteil an der Genauigkeit. Zunächst ist es für die einfache Navigationsanwendung erforderlich, zumindest 4 Satelliten in Sicht zu haben. Damit meint man gewöhnlicherweise, daß zumindest 4 Satelliten einen Höhenwinkel über etwa 10° besitzen müssen. Das Raumsegment ist so konstruiert worden, daß dies für jeden Ort der Erde garantiert werden kann. Es können zweitweise sogar bis zu 10 Satelliten sichtbar sein.

Neben der Anzahl spielt auch die Anordnung der Satelliten eine wesentliche Rolle. Deren Einfluß wird durch die *Dilution of Precision* ausgedrückt, siehe Gleichung (3.16).

3.9 Empfängeraufbau

In Abbildung 3.5 ist der schematische Aufbau eines GPS-Empfängers dargestellt. Die Satellitensignale werden über eine geeignete Antenne empfangen und über ein Antennenkabel an den Empfänger weitergeleitet. In der Empfangseinheit werden die Signale vermessen. Die erhaltenen Meßwerte gelangen zum Prozessor, der die relativ umfangreichen Berechnungen durchführt. Über eine Kommunikationsschnittstelle (meist RS232) können die Ergebnisse (Position, Zeit, Geschwindigkeit) der Berechnungen schließlich abgefragt werden. Viele Empfänger generieren aus den Meßdaten auch Zeit– und Frequenzsignale (z.B 1 PPS und 10 MHz, siehe Abschnitt 4.4). Manche Empfänger besitzen auch eine Tastatur und Anzeige zur interaktiven Kommunikation.

Im folgenden seien die wichtigsten Komponenten beschrieben:

3.9.1 Antenne und Antennenkabel

Die Antenne besteht aus der eigentlichen Antenne und eventuell einem Vorverarbeitungsteil. Da die von den Satelliten emittierten Signale in einem verhältnismäßig hohen Frequenzbereich liegen (Trägerfrequenz L_1 =

1,57542 GHz), muß obacht gegeben werden, dieses Signal ohne große Dämpfung an die Empfängereinheit zu übertragen. Entweder werden teure dämpfungsarme Kabel verwendet oder das Signal wird vorverarbeitet:

Vorverarbeitung der Satellitensignale in der Antenne

Kann die Entfernung zwischen Antenne und Empfängereinheit sehr kurz gehalten werden (Größenordnung 10 m), so kann auf eine Vorverarbeitung des Signals verzichtet und beide Einheiten direkt miteinander verbunden werden *(passive Antenne).*

Sollen größe Distanz überbrückt werden, so ist es notwendig, eine *aktive Antenne* zu verwenden. Diese enthalten entweder einen Vorverstärker, der das Signal soweit verstärkt, daß auch in größeren Entfernungen akzeptable Signalstärken vorhanden sind, oder sie sind mit einem Frequenzkonverter ausgerüstet, der das Signal von der hohen L_1-Frequenz auf eine Frequenz f transformiert, wobei f meist im Bereich von 10 – 80 MHz liegt. Da gängige Kabel in diesem Frequenzbereichen wesentlich geringere Dämpfungen aufweisen, sind auch größere Kabellängen möglich.

Es gibt auch eine Variante, bei der Empfänger und Antenne in ein Gehäuse eingebaut sind. Die benötigten Informationen werden dabei über Lichtwellenleiter an die möglicherweise sehr weit entfernt liegende Ausgabeeinheit übermittelt, die diese Informationen dann über die üblichen Schnittstellen anbietet.

Antennenmontage

Der Standort der Antenne sollte so gewählt sein, daß für Erhebungswinkel über 10° freie Sicht auf den Himmel besteht. Dies soll ermöglichen, daß auch wirklich alle verfügbaren Satelliten jederzeit empfangen werden können. Außerdem wird so vermieden, daß auch reflektierte Signale empfangen werden (siehe Abschnitt 3.8.1). Aus diesem Grund ist eine Montage auf dem Dach eines Gebäudes anzuraten.

3.9.2 Empfangseinheit

Die Empfangseinheit muß für jeden zu vermessenden Satelliten einen sogenannten *Korrelationskanal* vorsehen. Die meisten Empfänger besitzen heute meist zwischen 6 und 12 solcher Kanäle. Der schematisch Aufbau eines solchen Kanals ist in Abbildung 3.6 dargestellt (Diese Abbildung lehnt sich an die Darstellung in Wells et al. [1986] an).

Zunächst werden die Satellitensignale von der Antenne in der hohen L-Band–Frequenz über einen Mixer und einen Bandpaß auf eine technisch leichter zu handhabende *Intermediate Frequency (IF)* heruntertransformiert. Das sich ergebende Signal wird einer Regelschleife, der *Delay Locked Loop (DLL)* zugeführt. Diese kreuzkorreliert das Signal mit einem intern generierten C/A– oder P(Y)–Code (mit einer der 32 PRN-Nummern). Durch langsames Verschieben wird das Korrelationsmaximum gesucht. Dieses wird erreicht, sobald das generierte Signal mit dem Satellitensignal (mit richtiger PRN-Nummer) genau synchron läuft. Diese Schleife nennt man auch *Code Tracking Loop*, da sie sich mit dem PRN-Code synchronisiert.

Durch Mixen und Bandpaßfilterung mit dem intern generierten Codesignal kann die PRN-Kodierung aus dem Satellitensignal entfernt werden. Dieses solcherart gefilterte Signal wird nun dem Navigationscode-Demodulator zugeführt, der den Navigationscode herausfiltert. Das schließlich auch vom Navigationscode befreite Signal wird schließlich in eine *Phase Locked Loop (PLL)* gebracht, die den zentralen VCXO auf die Phase der Intermediate Frequency einregelt. Da die Regelschleife sich zumindest indirekt mit der Trägerphase des Satellitensignals synchronisiert, besitzt sie den Namen *Carrier Tracking Loop*.

Als Meßwert ergibt sich der Verschiebungswert aus der Codekorrelation, der durch Vergleich mit einem stabilen internen Oszillator — die Empfängeruhr — die Pseudoranges ergibt. Aus der Navigationscodedemodulation erhält man den Navigationscode und über die PLL gelangt man zur Phase der Intermediate Frequency (*Carrier Beat Phase*). Man spricht allerdings oft kurz von der Trägerphase (*Carrier Phase*).

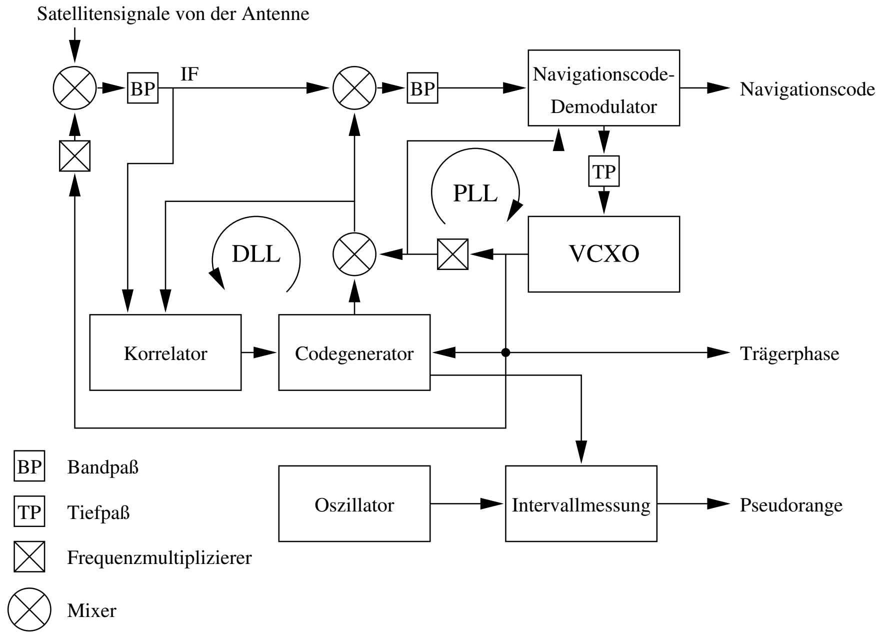

Abbildung 3.6: Aufbau eines Korrelationskanals zum Messen eines Satelliten

3.9.3 Prozessor

Der Prozessor übernimmt die von der Empfangseinheit generierten Meßdaten und –signale und berechnet daraus alle gewünschten Werte. Bevor allerdings die Empfangseinheit überhaupt Messungen durchführen kann, müssen zuerst geeignete Satelliten ausgewählt werden.

Satellitenauswahl

Geeignet sind dafür natürlich nur Satelliten die sich über dem Horizont befinden bzw. einen gewissen Höhenwinkel überschreiten (Zur Verringerung troposhärischer und ionosphärischer Fehler). Darüberhinaus ist natürlich die Satellitenkonstellation von besonderer Bedeutung. Je nachdem, welche Größe benötigt wird, versucht der Empfänger dabei, den GDOP–, TDOP–, PDOP–, HDOP– oder VDOP–Wert zu maximieren.

Um DOP-Werte berechnen und die Frage entscheiden zu können, ob sich ein Satellit über dem Horizont befindet, ist die Kenntnis zumindest der ungefähren Bahndaten der Satelliten, die ungefähre Antennenposition und die ungefähre Zeit erforderlich. Die ungefähren Bahndaten sind im *Almanach* des Navigationscodes verfügbar. Steht kein Almanach zur Verfügung oder kennt der Empfänger die ungefähre Zeit und Position nicht, so muß der Empfänger notgedrungen versuchen, die Satelliten sukzessive zu tracken. Findet er einen Satelliten, so kann er auch den Almanach laden (mindestens 12,5 Minuten Ladezeit) und verfügt über eine ungefähre Zeit (über den Satelliten). Außerdem können die genauen Ephemeriden (Bahndaten) des betreffenden Satelliten geladen werden (mindestens 30 Sekunden Ladezeit).

Berechnungen

Verfügt der Prozessor über die Navigationscodedaten und Pseudoranges genügend vieler Satelliten (Im Navigationsfall mindestens vier), so kann er die notwendigen Berechnungen durchführen:

Mit den verfügbaren Uhrendaten kann die Pseudorange jedes Satelliten von der Satellitenzeit SV-Zeit auf die systemglobale GPS-Zeit umgerechnet werden. Die Bahndaten ermöglichen entsprechend die Bestimmung der genauen Position zum Zeitpunkt der Signalaussendung. Berücksichtigt man zusätzlich relativistische, ionosphärische und troposhärische Korrekturen, so erhält man die korrekten Pseudoranges zur Berechnung der Antennenposition. Dies kann beispielsweise mit iterativen Verfahren erfolgen, siehe Abschnitt 3.5.

3.10 Spezielle Techniken

3.10.1 Differential GPS

Die Technik des Differential GPS benutzt eine Referenzstation (oder derer mehrere), deren Position sehr genau bekannt ist. Vemißt die Station alle sichtbaren Satelliten mit GPS-Empfängern, so kann sie die durch SA, Ionosphäre und Troposhäre eingebrachten Fehler durch Differenzbildung aus der bekannten Position errechnen. Werden diese Korrekturdaten — etwa über Funk — anderen GPS-Empfängern zugänglich gemacht, so können diese ihre selbst ermittelte Position entsprechend verbessern. Sofern diese GPS-Empfänger nicht zu weit (etwa 100 km) von der Referenzstation entfernt liegen, werden die Fehler für beide stark korreliert sein und die erhaltene Genauigkeit entsprechend groß sein (Submetergenauigkeit ist damit erreichbar). Das Problem in Europa ist derzeit die geringe Verfügbarkeit von DGPS-Referenzstationen. Allerdings ist ein System in Planung, das über das RDS-System des Hörfunks DGPS-Daten aussenden soll. Dieses System wird in den kommenden Jahren verwirklicht werden.

Die Zeitgenauigkeit kann mit der Differenzbildung grundsätzlich leider nicht erhöht werden, da hierzu der Referenzstation neben der genauen Position auch die genaue Zeit bekannt sein müßte.

3.10.2 Common View

Vermessen zwei weit auseinandergelegene Stationen mit exakt bekannten Positionen denselben Satelliten auf dieselbe Weise, so fallen die meisten Fehler weg. Durch Vergleich der Satellitenuhr mit den lokalen Uhren der Stationen erhält man gleichzeitig einen Vergleich zwischen diesen beiden lokalen Uhren. Die Genauigkeit diese Vergleichs beträgt 8 ns für Stationen mit einer Entfernung von 1000 km.

Teil III

Marktstudie

4 Marktstudie

Die Marktstudie hat den Zweck, einen Überblick über die am Markt frei erhältlichen GPS-Empfänger zu geben, die für das Projekt SynUTC eingesetzt werden können. Dabei wurden zwei Arten von Empfängern betrachtet: *Preiswerte* Geräte, deren Leistungsfähigkeit und Zuverlässigkeit experimentell bestimmt werden sollte, und *Hochleistungsgeräte*, die als Referenzgeräte zur Vermessung der preiswerten Geräte in Frage kamen. Diese Betrachtungen waren Ausgangspunkt für die Anschaffung mehrerer (4 oder 5) preiswerter Geräte und eines Referenzgerätes.

Die Grundlage für die Markstudie bildete die Übersicht über die erhältlichen GPS-Empfänger in der Zeitschrift *GPS-World Ausgabe Januar 1994* und *GPS-World Ausgabe Januar 1995*. Aufgrund dieser Übersichten wurden die in Frage kommenden Firmen angeschrieben und um nähere Produktinformation gebeten. Folgende Firmen wurden angeschrieben: Allen Osborne Associates, Astech Inc., Austron Co., Bancomm, Canadian Marconi Company, Datum Inc., Furuno Electric Co. Ltd, Garmin, Hopf Electronic, Lange Electronic, Leica AG, Magellan Systems, Motorola, Navstar Systems/NavSymm, NovAtel Communications, Odetics, r+a rost, Rockwell, Sercel, Stellar, 3S Navigation, Trak Systems, Trimble Navigation und TrueTime Inc.

4.1 Hochleistungsgeräte

In Tabelle 4.1 sind die Geräte, die nach einer ersten Auswahl als Hochleistungsgeräte geeignet erschienen, aufgelistet. In dieser Tabelle gibt der Wert in der Spalte *Preis* den Preis des Gerätes in ÖS inklusive Mehrwertsteuer (20 %) an. *UTC(SA)* gibt die Genauigkeit der UTC-synchronen Zeitsignale (1PPS, 10 MHz) zu UTC(USNO) an, und zwar unter SA-Bedingungen. Entsprechend kennzeichnet *UTC* die Genauigkeit ohne SA-Effekte. Die Angaben der Genauigkeit zu UTC bedeuten, falls kein Zusatz vorhanden ist absolute Schranken, sonst wird durch einen Zusatz *rms (Abkürzung für root mean square)* verdeutlicht das es sich um eine Standardabweichung handelt. In der Spalte *Frequenz* ist die Frequenzstabilität eines gegebenenfalls vorhandenen Frequenzausgangs angegeben. Die Abkürzung *n.b. (nicht bekannt)* zeigt an, daß entsprechende Werte nicht in der Spezifikation enthalten waren.

4.2 Preiswerte Geräte

Tabelle 4.2 listet all jene preiswerten Empfänger, die für die Vermessung geeignet erschienen. Die Spalte *Preis* gibt den Preis des Modells inklusive 20 % Mehrwertsteuer an. In der Spalte *UTC(SA)* sind die Genauigkeiten der 1 PPS-Pulse gegenüber UTC(USNO) angegeben. Ein Zusatz deutet an, daß es sich um eine Standardabweichung handelt. Die Spalte *Frequenz* gibt an, welche Frequenzausgänge vorhanden sind. In Spalte *Status*

Model	**Preis/ÖS**	**UTC(SA)/ns**	**UTC/ns**	Frequenz	Baujahr
Stellar 100 GPS Clock OCXO	61.000	300	100	10^{-11}	1993
Trak 8821-1	78.000	1.000	*n.b.*	*n.b.*	1993
Austron 2201A	86.000	*n.b.*	100	10^{-10}	1989
Odetics SatSync II	102.000	*n.b.*	100	*n.b.*	*n.b.*
Trak 8820A	129.000	300 rms	100	*n.b.*	1992
Ball Efratom Rubidium 2000	151.000	300	*n.b.*	$5 \cdot 10^{-12}$	*n.b.*
Austron 2200A	162.000	*n.b.*	100	10^{-11}	1989
Trak 8812	167.000	300 rms	100	10^{-12}	1993
Odetics SatSync II Rubidium	186.000	*n.b.*	100	*n.b.*	*n.b.*
Ball EfratomGPS-RR	302.000	100	*n.b.*	10^{-12}	*n.b.*
Trak 8860	924.000	100	100	10^{-12}	1993

Tabelle 4.1: Hochleistungsgeräte

wird beschrieben, über welche Schnittstelle Statusinformation zugänglich sind (über RS232, Alarmleitungen oder LEDs). Die Abkürzung *n.b. (nicht bekannt)* zeigt an, daß entsprechende Werte nicht in der Spezifikation enthalten waren.

Model	**Preis/ÖS**	**UTC(SA)/ns**	**Frequenz**	**Status**	**Baujahr**
Motorola Oncore Kit	16.000	50 rms/130 rms	*keine*	RS232	1992
Trimble SVeeSix-104	19.000	1000	*keine*	RS232	1992
Trimble SVeeSix-CM2	19.000	1000	*keine*	RS232	1993
NavSymm XR-5 Kernel	21.000	100 rms	*keine*	RS232	1993
Leica GPS Engine	30.600	300	*keine*	*n.b.*	1991
Sercel GPS Board	37.000	200 + SA	*keine*	RS232	*n.b.*
TrueTime XL	41.000	*n.b.*	optional	RS232&Alarm	1993
Stellar GPS Model 100	41.500	300	10 MHz	RS232	1993
NavSymm NTFR-S	42.000	100 rms	.5,1,5,10 MHz	RS232&LEDs	1994
Ashtech Sensor Aircraft	46.200	*n.b.*	*keine*	RS232	*n.b.*
Ashtech Sensor Marine	53.000	*n.b.*	*keine*	RS232	*n.b.*
TrueTime 705	53.000	*n.b.*	optional	RS232	1994
Odetics GPStarplus	68.000	*n.b.*	*keine*	RS232&Alarm	1995
NovAtel Performance Series	72.600	300 rms	*keine*	RS232	1994

Tabelle 4.2: Preiswerte Geräte

4.3 Ausgewählte Geräte

Die hohen Preise für die Hochleistungsgeräte haben dazu geführt, daß auf ein Referenzgerät verzichtet wurde. Stattdessen wurde das Rubidiumfrequenznormal FRS-C von Ball Efratom beschafft, das in Kombination mit dem besten verfügbaren preiswerten Empfänger (Stellar) die notwendige Referenzzeitskala erzeugen soll.

Es wurden schließlich folgende Geräte angeschafft:

- Modell 100 A GPS Clock mit OXCO-Option und TTL-Option von Stellar Absolute Time Corporation
- Navsymm Timing and Frequency Receiver NTFR-S

- Motorola VP Oncore Evaluation Kit
- Trimble SVeeSix-CM2 Starter Kit mit Rooftop-Antenna
- Magellan Brain Timing Receiver
- Rubidiumfrequenznormal FRS-C mit TTL-Option von Ball Efratom

4.4 Schnittstelle zu den GPS-Empfängern

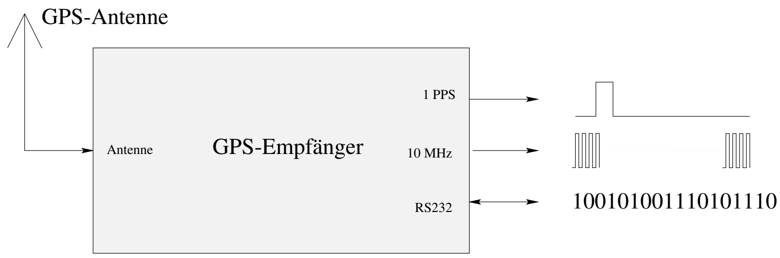

Abbildung 4.1: Schnittstellen eines GPS-Empfängers

Die Marktstudie war die Grundlage für die Definition einer einheitlichen Schnittstelle zu den GPS-Empfängern. Diese wird im folgenden beschrieben:

Von der GPS-Antenne gelangen die Satellitensignale über das Antennenkabel zum eigentlichen Empfänger und werden dort verarbeitet. Dadurch wird der Empfänger in die Lage versetzt die gewünschten Zeitsignale und Zeitnachrichten zu generieren. Die auftretenden Schnittstellen sind in Abbildung 4.1 skizziert.

4.4.1 1 PPS – 1 Puls Pro Sekunde

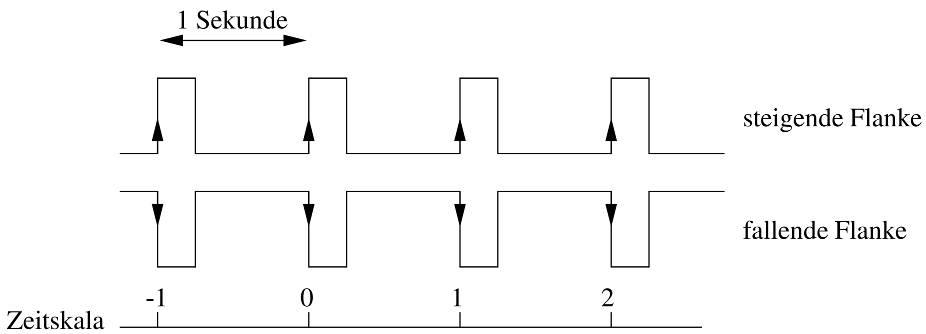

Abbildung 4.2: 1 PPS — 1 Puls Pro Sekunde

Das hauptsächlich verwendete Zeitsignal bei GPS-Empfängern ist das sogenannte 1 PPS-Signal (*1 Pulse Per Second*). Es liefert entweder mit seiner steigenden oder fallenden Flanke den Beginn jeder Sekunde der gewünschten Zeitskala (siehe Abbildung 4.2). Die auftretenden Pegel sind meist TTL-Pegel, es kommen aber auch andere Pegel vor (z.B. 0 V und 10 V-Pegel). Auch die Pulsdauer variiert stark von Empfänger zu Empfänger (1 μs bis 0,25 s). Für die Zeitskala steht bei den meisten Empfängern die Wahl zwischen GPST (GPS-Systemzeit) und UTC (Koordinierte Weltzeit) frei.

4.4.2 Frequenzausgänge, 10 MHz

Bei Frequenzausgängen müssen zwei Arten unterschieden werden: Jene die synchron zu den 1 PPS-Impulsen laufen und lediglich eine Frequenzvervielfachungen derselben darstellen und jene, bei denen der Empfänger versucht, jederzeit die maximal erreichbare Frequenzgenauigkeit zu bieten und daher auf lange Frist dem 1 PPS-Signal „davonläuft". Die am häufigsten benutzte Frequenz ist 10 MHz. Allerdings besitzen nicht alle GPS-Empfänger einen Frequenzausgang.

4.4.3 Zeitnachrichten

Gemeinsam mit den Signalen werden verschiedene Nachrichten über eine RS232 Schnittstelle geliefert. Die Form dieser Nachrichten unterscheiden sich je nach Empfänger. In diesen Nachrichten sind folgende Informationen enthalten

1. Sekundenanteil. Damit ist jener ganzzahlige Sekundenwert gemeint, der durch einen bestimmten 1 PPS-Puls dargestellt wird (Er ist klarerweise von der gewählten Zeitskala abhängig: UTC oder GPST). Ein oft gewählter Ansatz, diesen Sekundenanteil darzustellen, ist die Angabe der vergangenen Sekunden seit der ersten GPS-Woche (deren erster Tag Sonntag, der 6. Jänner 1980 war). Die entsprechende Nachricht wird meist nach Auftreten eines 1 PPS-Pulses übermittelt. Je nach Empfänger gilt der dabei übermittelte Sekundenwert für den vorausgegangenen oder den kommenden Puls.

2. Gültigkeit der Signale. Diese Information teilt mit, ob das 1 PPS Signal synchron mit der gewählten Zeitskala läuft und/oder das 10 MHz-Signal mit der spezifizierten Genauigkeit anliegt.

3. Genauigkeit der Signale. Manche Empfänger übermitteln auch Informationen über die momentane Genauigkeit der Signale.

4. Sonstige Informationen. Hiermit sind sehr spezifische Informationen gemeint, die z.B. angeben, welche oder wieviele Satelliten gerade vermessen werden.

4.4.4 Alarm– bzw. Statussignale

Direkte Alarmsignalleitungen, die eine Fehlfunktion oder die Ungültigkeit eines Pulses anzeigen, sind gewöhnlich nur bei relativ teuren Empfängern anzutreffen. Für Empfänger, wie sie bei der Verfügbarkeitsuntersuchung eingesetzt werden, werden solche Informationen über die serielle RS232–Schnittstelle übertragen, häufig gemeinsam mit der Nachricht, die den Sekundenanteil eines Pulses enthält. Daher bleiben bei der vorliegenden Untersuchung Signale dieser Art unberücksichtigt.

Teil IV

Experimentelle Evaluation

5 Untersuchung des Fehlverhaltens

5.1 Untersuchungsziel

Das Untersuchungsziel dieser Diplomarbeit ist, festzustellen, wie zuverlässig die Zeitsignale der GPS-Empfänger (1 PPS, 10 MHz) die jeweilige Zeitskala (GPS-Systemzeit bzw. UTC) darstellen. Insbesondere interessiert folgendes:

- Wie stark schwanken die 1 PPS-Signale um die Referenzzeit ? Wie sieht die Verteilungsfunktion der *Ablage* der 1 PPS-Signale von der Referenzzeit aus ?
- Wie stark schwanken die 1 PPS-Signale über eine gewisse Mittelungszeit τ ? Wie sieht die zugehörige Verteilungsfunktion aus ?
- Wie sieht das Frequenzverhalten der 10 MHz-Frequenzausgänge aus ?
- Wie zuverlässig sind die von den GPS-Empfänger gelieferten Nachrichten (z.B. Sekundenanteil) ?
- Treten Fehlpulse auf oder werden Pulse ausgelassen ?
- Wann verhalten sich Empfänger fehlerhaft ?

5.2 Untersuchungsplanung

Die Verfügbarkeitsunteruchung soll in zwei Phasen abgewickelt werden werden: Nach Probeläufen des zu entwickelnden Meßaufbaus soll eine Langzeituntersuchung von circa 2 Monaten (etwa 5 Millionen Sekunden) am Dach des neuen Elektrotechnischen Insitutes der Technischen Universität Wien (Gußhausstr. 25-27, 1040 Wien) durchgeführt werden. Die gewählte Meßdauer ergibt sich aus dem Kompromiß, einerseits ausreichend Datenmaterial zur Verfügung zu haben und andererseits den Aufwand betreffend Speicherung und Auswertung der Daten nicht zu hoch zu treiben.

Ein wesentliches Problem der Messung ist die schwere Zugänglichkeit der exakten GPS bzw. UTC-Zeitskalen. Die beste Methode, diese Zeitskalen zu erhalten, ist wiederum die Benutzung eines GPS-Empfängers. Daher soll der beste der gekauften Empfänger (Stellar GPS Model 100 A) benutzt werden, um die Referenzskala darzustellen. Zur Steigerung der Genauigkeit wurde ein Rubidiumfrequenznormal beschafft, das einen größeren

Teil der auftretenden statistischen Schwankungen (hauptsächlich verursacht durch Selective Availability) auszumitteln vermag. Es existieren zwar bessere Methoden; z.B. Common View mit Genauigkeit von einigen ns, allerdings sind dazu Uhren höherer Qualität erforderlich. Nebenbei wäre der Aufwand für einen kontinuierlichen Betrieb sehr viel höher.

5.3 Meßaufbau der Untersuchung

Für den Meßaufbau stehen folgende Geräte zur Verfügung (für Geräte, die nicht extra angekauft wurden, ist die Herkunft in Klammern angegeben):

- GPS-Empfänger
  1. Modell 100 A GPS Clock mit OCXO-Option und TTL-Option von Stellar Absolute Time Corporation
  2. Navsymm Timing and Frequency Receiver NTFR-S
  3. Motorola VP Oncore Evaluation Kit
  4. Trimble SVeeSix-CM2 Starter Kit mit Rooftop-Antenna
  5. Magellan Brain Timing Receiver
  6. Rockwell Mickrotracker (Leihgabe der Fernuniversität Hagen)
- Rubibum-Atomuhr FRS-C von Ball Efratom
- Kühlbox Marke GioStyle
- Frequenzteilerschaltung (von Fritz Mörth im Rahmen des Projekts angefertigt)
- zwei Logikanalysatoren HP16500B von Hewlett Packard (einer davon im Besitz des Instituts für Computertechnik, ICT an der TU Wien)
- Meß-PC zur Steuerung des Meßaufbaus (Besitz des ICT)
- Kommunikations-PC zur Übermittlung der Daten in das lokale TU-Netz (Besitz des ICT)

In den folgenden Kapiteln ist die Hardware des Meßaufbaus detailliert beschrieben.

6 Aufbau des Meßsystems

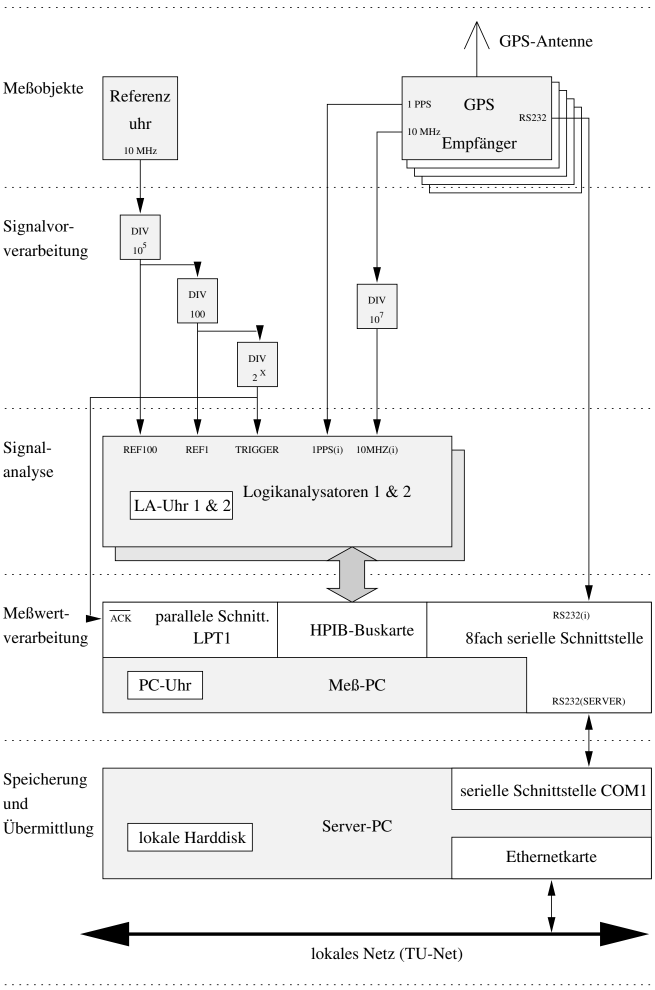

Abbildung 6.1: Schematischer Aufbau des Meßsystems

Aus Abbildung 6.1 ist der schematische Aufbau des Meßsystems zu entnehmen. Er ist vertikal nach der Funktionalität der einzelnen Teile gegliedert. Der grobe Ablauf ist wie folgt:

Zuoberst stehen die *Meßobjekte*, das sind die Referenzuhr und die GPS-Empfänger. Daran schließt sich die *Signalvorverarbeitung* an, die durch Frequenzteilung und Signalanpassung die geeigneten Meßsignale erzeugt. Dadurch liegen alle Signale in der gleichen Form vor: Sie sind 1 PPS-Signale mit TTL-Pegeln.

Diese Signale werden mit Hilfe von Logikanalysatoren gemessen (mit einer Granularität von 8 ns). Die daraus gewonnenen Daten (die Zeitstempel der Signalflanken) werden an den Meß-PC übergeben, der das gesamte System steuert. Im Meß-PC werden auch die Nachrichten der GPS-Empfänger über die RS232-Schnittstellen empfangen und mit den Pulsdaten in Beziehung gesetzt (es wird u.a. die Verzögerung einer Nachricht zum zugehörigen Puls ermittelt). Außerdem werden alle Zeitstempel auf die Referenzuhr umgerechnet. Diese komplexen Berechnungen werden hier unter dem Titel *Meßwertverarbeitung* zusammengefaßt. Die sich ergebenden Rohdaten werden über eine serielle Schnittstelle an den Server-PC übermittelt, der die Daten auf Festplatte speichert und die manuelle Übertragung dieser Daten in das lokale Netz (TU-Net) erlaubt.

Diese Rohdaten, die nun in Dateiform vorliegen, können geeignet ausgewertet werden.

Die einzelnen Teile des Systems sind im folgenden detailliert beschrieben:

6.1 Meßobjekt Referenzuhr

Sowohl die GPS-Empfänger als auch die Referenzuhr werden als die zu untersuchenden Meßobjekte betrachtet. Die Referenzuhr wird allerdings darüber hinaus zur zeitlichen Steuerung der Logikanalysatoren verwendet und nimmt daher aktiv an der Messung teil. Dies äußert sich auch darin, daß alle Zeitstempel in den endgültigen Daten nur Bezug auf die Referenzuhr nehmen. Sie nimmt daher eine Sonderstellung ein.

Die Anforderungen an die Referenzuhr sind neben Zuverlässigkeit auch eine möglichst hohe Frequenzstabilität. Die Frequenzgenauigkeit hingegen spielt keine Rolle, da die Frequenzabweichung der Referenzuhr mit Hilfe der 1 PPS Signale der GPS-Empfänger durch Langzeitmittelung bestimmt werden kann. Solange die genaue Ablage der Referenzuhr von der betrachteten Zeitskala nicht bekannt ist, können daher auch keine systematischen Abweichungen der Empfänger bestimmt werden. Meßbar ist allerdings das stochastische Verhalten. Die dabei erzielbare Meßgenauigkeit ist von der Güte der Referenzuhr (insbesondere der Frequenzstabilität) abhängig.

Bei den vorliegenden Messungen wird ein Rubidiumoszillator Model FRS-C der Firma Ball Efratom als Referenzuhr eingesetzt. Die in der Spezifikation angegebenen Frequenzstabilitäten sind in Tabelle 6.1 (siehe Handbuch Bal [1989]) zusammengefaßt.

Mittelungszeit	Frequenzstabilität
1 s (Allan Var.)	$1 \cdot 10^{-10}$
10 s (Allan Var.)	$3,16 \cdot 10^{-11}$
100 s (Allan Var.)	$1 \cdot 10^{-11}$
1 Tag[a]	$4 \cdot 10^{-11}$

[a] Die Stabilitätsangabe für den Mittelungszeitraum von einem Tag gilt nur wenn, der Oszillator bereits mindestens 7 Tage kontinuierlich betrieben wurde und die Umgebungstemperatur innerhalb des Intervalls 25 ± 3 °C bleibt.

Tabelle 6.1: Frequenzstabilität des Rubidiumoszillators

Zu Vermeidung der durch Temperaturschwankungen hervorgerufenen Frequenzänderungen wurde die Uhr in einer Kühlbox mit Peltier-Element untergebracht. Eine Temperatursteuerung hält die Luftemperatur im Inneren der Kühlbox auf 25 ± 1°C. (siehe Abbildung 6.2) Der Umwälzlüfter sorgt für konstante Temperaturverhältnisse im Inneren. Zum Betrieb der Kühlbox und des Lüfters ist eine Versorgungsspannung von 12 V bei 4,2 A erforderlich.

Die Rb-Uhr selbst wird bei einer Spannung von 24 V betrieben und zieht einen Strom von 2 A zu Beginn (zum Aufheizen) und 0,5 A im Dauerbetrieb. Der Frequenzausgang liefert 10 MHz mit TTL Pegeln an 50 Ω und wird im Meßaufbau über ein Koaxialkabel mit einem BNC(m)-Stecker nach außen geführt. Um ein korrektes Signal zu erhalten, ist das Ende der Übertragungsleitung entsprechend mit 50 Ω abzuschließen. Hier geschieht dies in der Frequenzteilerschaltung, in die das 10 MHz-Signal eingespeist wird.

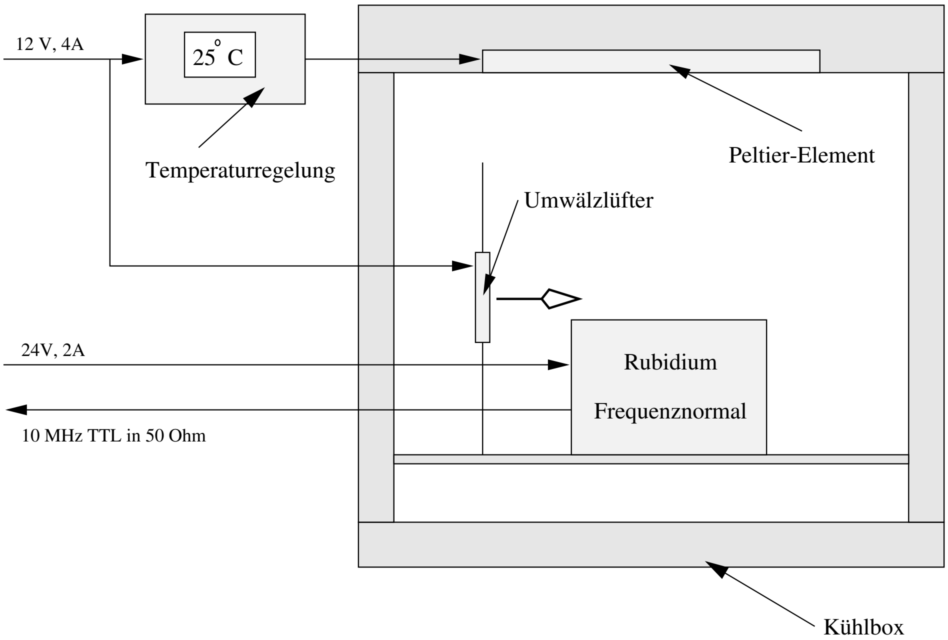

Abbildung 6.2: Die Rubidiumuhr

6.2 Meßobjekt GPS-Empfänger

Für die Messung werden 6 sehr unterschiedliche GPS-Empfänger verwendet (der Rockwell Microtracker ist eine Leihgabe, alle anderen wurden eigens angeschafft). Alle diese Empfänger kommunizieren über eine RS232-Schnittstelle, allerdings mit sehr verschiedenen Protokollen. Die Empfänger sind zum Großteil in einem 19″ Rack untergebracht und werden dort zentral von einem Schaltnetzteil versorgt:

6.2.1 19″ Rack

Außer dem Empfänger von Stellar, der ein eigenes Gehäuse und Netzteil besitzt, wurden alle Empfänger als Rack-Einschübe in einem 19″ Rack untergebracht und werden von einer zentralen Stromversorgung (Schaltnetzteil, 5 V (3 A) 12 V (2 A) 24 V (2 A)) versorgt. Die Rackeinschübe erhalten die Versorgung über die 64 poligen VG-Buchsenleisten (32 Pinreihen zu je 2 Pins) an der Rückseite jeder Karte. Die Belegung dieser Buchsenleiste ist einheitlich folgende:

Pinreihe	Belegung
1	Masse (0 V)
17	12 V
32	5 V

Die Spannungen des Schaltnetzteiles werden auch an der Front des 19″ Racks über Bananenbuchsen herausgeführt. Auf einer Aluplatte sind dafür 3 Buchsenreihen bestehend aus jeweils einer schwarzen und roten Buchse vertikal angeordnet. Die schwarze Buchse liefert jeweils die Masse, die rote die Spannung. Von oben nach unten lauten die Spannungen 5 V, 12 V und 24 V. Im Meßaufbau bleiben diese 5 V unbenutzt, sie dienen lediglich als schnell verfügbare Spannungsversorgung. Die 12 V werden zur Versorgung der Antennen–Empfangseinheit des NavSymm-Empfängers benutzt. Die 24 V versorgen die Rubidiumatomuhr.

Um die Schnittstellen einheitlich zu halten, wurden vier GPS-Empfängern eigene Trägerplatinen (Europaformat) spendiert. Die anderen GPS-Empfänger lagen bereits als Rackeinschub vor (*Timing-Unit* des NavSymm) oder hatten ein eigenes Gehäuse (Empfänger von Stellar)

Die einheitliche Frontplatte ist in Abbildung 6.3 dargestellt. Über den BNC(f)-Stecker wird der 1 PPS-Puls mit TTL-Pegel geliefert. Die Pulsweite ist dabei vom jeweiligen Empfänger abhängig. Alle 1 PPS-Pulse werden über einen 74LS125 getrieben. Die SUB-D25(f)-Buchse liefert die Kommunikationsleitungen der jeweiligen RS232-Schnittstellen. Dabei werden nur drei Leitungen benutzt: TxD, RxD und Masse. Die Belegung der Buchse (Ausnahme Empfänger von Trimble: Hier sind die Leitungen RxD und TxD vertauscht) ist wie folgt:

Pin	Belegung
2	TxD aus der Sicht des Empfängers
3	RxD aus der Sicht des Empfängers
7	Masse

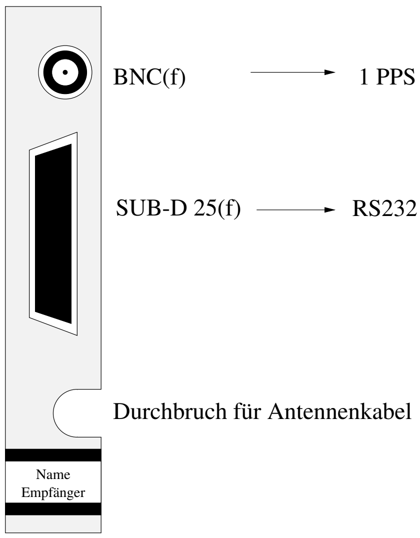

Abbildung 6.3: Die Frontplatte der 4 Empfänger

Im folgenden werden die GPS-Empfänger einzeln beschrieben:

6.2.2 Model 100A GPS Clock von Stellar Absolute Time Corporation

Allgemeines

Dieses Modell ist ein speziell für die Bereitstellung von Zeitsignalen entwickelter GPS-Empfänger und besitzt neben einem 1 PPS-Ausgang auch einen 10 MHz-Frequenzausgang. Dieser Empfänger wurde mit der Option

010, High Performance Oscillator, und der Option 091, TTL Frequency Output für den 10 MHz-Ausgang eingekauft.

Dieser Empfänger kann nur stationär betrieben werden, da er zunächst sofort nach Inbetriebnahme (oder per Befehl) in den *Static Positioning Mode* übergeht und dabei durch Mittelung über einen längeren Zeitraum (ca. 6,5 Stunden) versucht, die Position der Antenne möglichst genau zu bestimmen. Sobald diese Position vorliegt, geht der Empfänger in den *Time Transfer Mode* über, in dem er erst in der Lage ist — sofern Satelliten getrackt werden — die korrekten Zeitsignale auszugeben. Diese Vorgänge können anhand der LEDs an der Frontplatte verfolgt werden.

Die ermittelte Position lautet in GPS-Ellipsoidkoordinaten: Nördliche Breite: 48° 11′46.″400 N, Östliche Länge: 16° 22′9.″500 E, Höhe über Ellipsoid: 239.00 m Dieser Standort befindet sich am Dach des neuen Elektrotechnischen Instituts der Technischen Universität Wien.

Schnittstellen

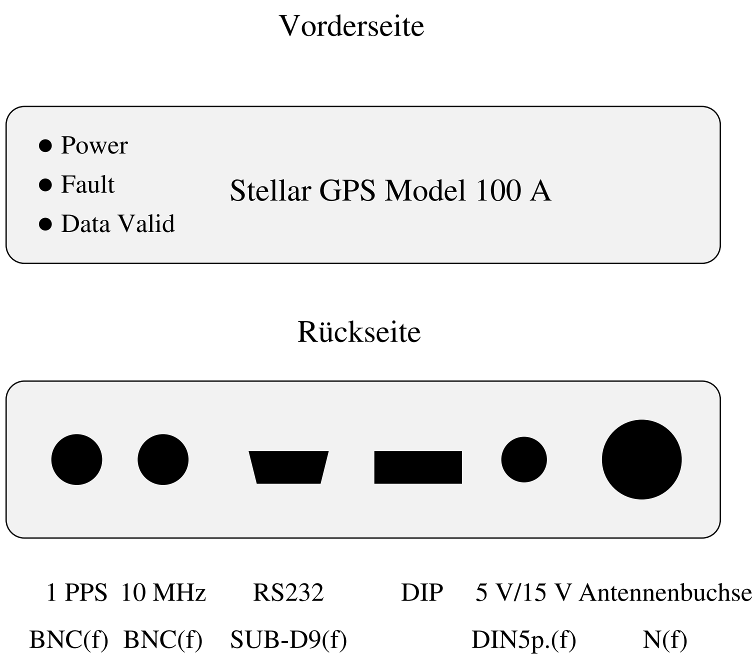

Abbildung 6.4: Der GPS-Empfänger Model 100A GPS Clock von Stellar Absolute Time Corporation

Dieser GPS-Empfänger besitzt ein eigenes Gehäuse (Maße 185 mm × 240 mm × 39 mm) und ein mitgeliefertes Netzteil (5 V und 15 V). In Abbildung 6.4 ist die Ansicht der Front– und Rückseite dargestellt. An der Frontseite sind drei LEDs angebracht, die den Zustand des Geräts anzeigen (siehe nachfolgende Tabelle):

Farbe	Bezeichnung	Bedeutung
gelb	POWER	Die Stromversorgung ist angeschlossen
rot	FAULT	Es wird kein Satellit getrackt
grün	DATA VALID	Die Ausgangssignale (1 PPS und 10 MHz) sind gültig: Allerdings erlischt die DATA VALID LED spezifikationsgemäß erst 5 Minuten nach Verlust des letzten Satelliten. Dies ist möglich, da dieser Empfänger einen guten Oszillator besitzt.

An der Rückseite befinden sich die verschiedenen Buchsen für Signalausgänge, RS232-Schnittstelle, Antenneneingänge und Stromversorgung. Außerdem gibt es DIP–Schalter für diverse Einstellungen. In der folgenden Tabelle sind die verschiedenen Buchsen kurz skizziert:

Bezeichnung	Buchsentyp	elektrische Spezifikation
1 PPS	BNC(f)	steigende Flanke von 0 auf 10 V kennzeichnet den Beginn einer Sekunde der gewählten Zeitskala (mit 50 Ω belastbar)
10 MHz	BNC(f)	TTL-Pegel
RS232-C	SUB-D9(f)	9polige RS232-C Schnittstelle für DTE
Antennenbuchse	TNC(f)	12 V Versorgung für aktive Antenne und Antennensignale
Stromversorgung	DIN-5pol.(f)	5 V und 15 V

Die an der Rückseite befindlichen DIP-Schalter ermöglichen die Einstellung der Standardwerte, die nach Verlust der Spannungsversorgung eingestellt werden. Drei Einstellungen können vorgenommen werden: Die Zeitskala, die Baudrate der RS232-Kommunikation und die Länge des Antennenkabels (zur Kompensation der Verzögerung durch das Antennenkabel). In folgender Tabelle sind diese Einstellungen detailliert aufgelistet (fett gedruckt sind die werkseitigen Einstellungen):

Einstellung	Wert	DIP-Schalter	
Zeitskala	**GPS**	DIP1=ON	
	UTC	DIP1=OFF	
RS232-Baud-Rate	1200	DIP2=ON	DIP3=ON
	2400	DIP2=OFF	DIP3=ON
	9600	DIP2=ON	DIP3=OFF
	19200	DIP2=OFF	DIP3=OFF
Kabellänge	**25 m**	DIP4=OFF	DIP5=ON
	100 m	DIP4=ON	DIP5=OFF
	300 m	DIP4=OFF	DIP5=OFF

Antenne und Antennenkabel

Die Antenne besitzt eine kreisrunde Form mit Durchmesser 8,9 cm und hat eine Höhe von 1,7 cm. Über eine TNC(f)-Buchse an der Unterseite werden die Versorgungsspannung (12 V) zugeführt und gleichzeitig die Satellitensignale übertragen.

Das Antennenkabel der Type RG-213 besitzt einen Länge von 25 m und ist beidseitig mit einem TNC(m)-Stecker versehen.

Kommunikation über RS232

Die Default-Einstellung für die Kommunikation sieht eine Baudrate von 9600 Baud mit 8 Datenbits, keinem Paritätsbit und einem Stopbit vor. Außerdem werden defaultmäßig die Modemstatusleitung (CTS, RTS) nicht verwendet. Die 9 polige SUB-D9 Buchse ist derart beschaltet, wie es für ein DTE (*Data Terminal Equipment*) üblich ist.

Das verwendete Protokoll benutzt nur ASCII-Zeichen und hält die Nachrichten in einer für Menschen lesbaren Form. Die Nachrichten zum Empfänger (Befehle) haben alle folgende Form:

`$`*Befehlsname Parameterliste*`<cr>`[1]

Dabei ist die Parameterliste durch den Befehlsnamen definiert und kann je nach Befehl auch vollkommen entfallen. Bei mehr als zwei Parametern sind diese durch Kommas getrennt.

Nachrichten vom Empfänger erscheinen periodisch (die Periodendauer läßt sich einstellen, beträgt aber defaultmäßig den kleinstmöglichen Wert von 1 Hz) knapp nach dem Auftreten des 1 PPS-Pulses, wobei die verschiedenen Nachrichten einzeln ein– und ausgeschaltet werden können. Die Reihenfolge der Nachrichten ist allerdings nicht spezifiziert. Die Form dieser Nachrichten ist folgende:

`$`*Nachrichtenname*`,`*Werteliste*`*`*Prüfsumme*`<cr><lf>`

Die Werte der Werteliste sind wieder durch Kommas getrennt. Die *Prüfsumme* ist der Hex-Wert (dargestellt als zwei ASCII-Zeichen, z.B. `A9`) für das exklusive Oder aller Zeichen nach dem `$` und vor dem `*`.

Bevor das Meßsystem die Messung beginnt, wird der GPS-Empfänger vom Meßsystem (allerdings nachdem er bereits den *Time Transfer Mode* erreicht hat) mit folgender Sequenz initalisiert:

```
unlogall<cr>
log status<cr>
log taf<cr>
```

Hierbei werden mit dem `unlogall`-Befehl alle Nachrichten ausgeschaltet und nachfolgend die Nachrichten `status` und `taf` eingeschaltet. Diese beiden Nachrichten tragen die wichtigsten Informationen und werden im folgenden detailliert besprochen (bei Kommandos wird zwischen Groß– und Kleinschreibung nicht unterschieden, Ausgaben vom Empfänger erfolgen allerdings in Großbuchstaben):

`$STATUS,`*W*`,`*TOW*`,`*TR*`,`*DVS*`,`*NTS*`,`*HGD*`,`*FS*`*`*Prüfsumme*`<cr><lf>`

Variable	Beschreibung
W	GPS/UTC-Week; GPS/UTC-Wochennummer
TOW	GPS/UTC-Time of Week; Hundertstelsekunden seit Beginn einer GPS/UTC-Woche
TR	Time Reference; gewählte Bezugszeitskala: `GPS` oder `UTC`
DVS	Data Valid Status; Gültigkeit der Ausgangsignale (entspricht der grünen LED für DATA VALID): `DATA VALID` oder `DATA INVALID`
NTS	Number of Tracked Satellites; Anzahl der derzeit getrackten Satelliten: `TRACKING` *Anzahl* `SVs`
HGD	Have GPS Data Base; GPS-Datenbank (Almanach und Ephemeriden) komplett ?: `GPS DB COMPLETE` oder `GPS DB INCOMPLETE`
FS	Fault Status; Fehlerzustand: `FAULT xxxx`. Die `xxxx` sind Hex-Ziffern, die den BIT-Status (Built in Test Status) angeben. (`0000`: OK, `A000`: Antennenfehler, `0B00`: Empfängerfehler)

Beispiel. `$STATUS,726,49769588,GPS,DATA INVALID,TRACKING 4 SVs,GPS DB COMPLETE,FAULT 0000*4F<cr><lf>`

Zur Erläuterung: GPS bzw. UTC-Wochen beginnen immer am Sonntag um 00.00. Die erste GPS/UTC-Woche (Nr. 0) beginnt am 6 .Jänner 1980 00.00 (Natürlich ein Sonntag). Zu diesem Zeitpunkt unterschieden sich die beiden Zeitskalen in puncto Schaltsekunden noch nicht.

Die zweite interessante Nachricht ist `taf` (time and frequency):

`$TAF,`*W*`,`*TOW*`,`*TO*`,` *TU*`,` *FO*`,`*FUS*`,` *GTFOM*`,`*GFFOM*`*`*Prüfsumme*`<cr><lf>`

[1] `cr` kennzeichnet das ASCII-Zeichen 13, Carriage Return

Variable	Beschreibung
W	GPS/UTC-Week; GPS/UTC-Wochennummer
TOW	GPS/UTC-Time of Week; Hundertstelsekunden seit Beginn einer GPS/UTC-Woche
TO	Time Offset; Fehler des 1 PPS-Signals in Relation zur wahren Zeitskala (in ns)
TU	Time offset solution Uncertainty; Unsicherheit in der Bewertung des obigen Wertes (in ns)
FO	Frequency Offset: Fehler des 10 MHz-Signals gegenüber der wahren Zeitskala (in ps/s)
FU	Frequency offset solution Uncertainty; Unsicherheit in der Bewertung des obigen Wertes: (in ps/s)
GTFOM	Time Figure of Merit (0-9); Gütewert für 1 PPS-Signal
GFFOM	Frequency Figure of Merit (0-9); Gütewert für 10 MHz-Signal

Beispiel. `$TAF,726,49765207,-77719859,18,52456,112,9,9*41<cr><lf>`

Die Spezifikation aller Nachrichten vom und zum Empfänger sind im Benutzerhandbuch des Empfängers (Abs [1995]) zu finden.

6.2.3 NavSymm Timing and Frequency Receiver NTFR-S

Allgemeines

Auch dieses Modell ist ein speziell für die Bereitstellung von Zeitsignalen entwickelter GPS-Empfänger und besitzt neben einem 1 PPS-Ausgang auch einen 10 MHz-Frequenzausgang. Eine Besonderheit dieses Empfängers ist die Trennung zwischen der kombinierten Antennen–Empfangseinheit und einer sogenannten *Timing-Unit*, die die Zeitsignale ausgibt und die RS232-Schnittstelle bedient. Ungewöhnlich ist auch die Verbindung dieser beiden Einheiten über Lichtwellenleiter.

Dieser Empfänger kann nur stationär betrieben werden, da er zunächst sofort nach Inbetriebnahme (oder per Befehl) in den *Automatic Position Location*-Modus übergeht und dabei durch Mittelung über einen längeren Zeitraum (mehrere Stunden) versucht, die Position der Antenne möglichst genau zu bestimmen. Sobald diese Position vorliegt, geht der Empfänger in einen *Timing*-Modus über, in dem er erst in der Lage ist — sofern Satelliten getrackt werden — die korrekten Zeitsignale auszugeben. Diese Vorgänge können anhand der Farbe einer LED an der Frontplatte der Timing-Unit verfolgt werden.

Die GPS-Ellipsoidkoordinaten dieser ermittelten Position sind: Nördliche Breite: $48^\circ\ 11'46.''380$ N, Östliche Länge: $16^\circ\ 22'9.''000$ E, Höhe über dem GPS-Ellipsoid: 257.00 m

Schnittstellen

In Abbildung 6.5 ist die kombinierte wetterfeste Antennen–Empfangseinheit skizziert. An der Unterseite befinden sich zwei Anschlüsse für die Lichtwellenleiter zur Kommunikation nach unten bzw. nach oben (UP– und DOWN-LINK). Außerdem ist eine eigene Stromversorgung (12 V) und Erdung erforderlich.

Die interessanten Schnittstellen befinden sich auf der Timing-Unit, die als $19''$ Rack-Einschubkarte ausgeführt ist. Die Frontplatte ist in Abbildung 6.6 dargestellt.

Die Kontroll-LED zeigt in drei Farben den Systemzustand an:

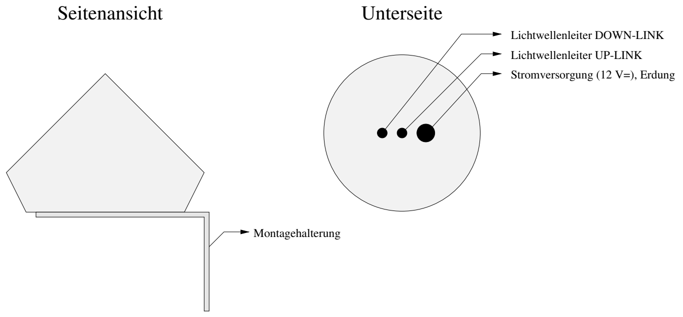

Abbildung 6.5: Die kombinierte Antennen–Empfangseinheit des NavSymm GPS-Empfängers

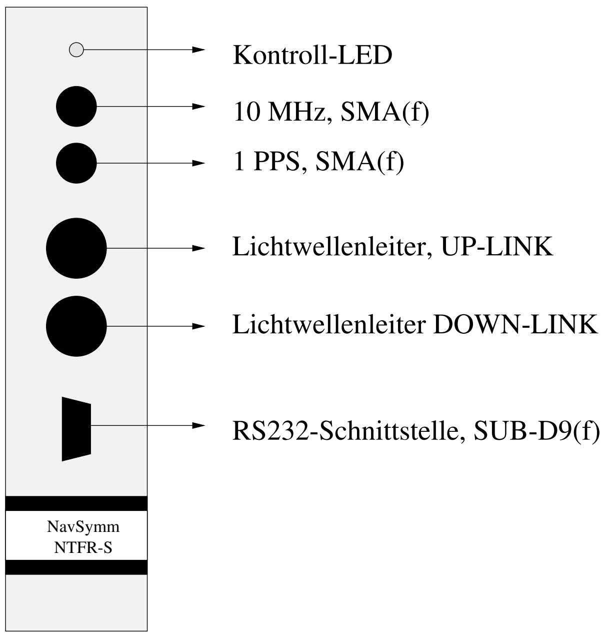

Abbildung 6.6: Die Frontplatte der Timing-Unit des NavSymm GPS-Empfängers

Farbe	Bedeutung
Grün	Die Signalausgänge (1 PPS und 10 MHz) sind gültig
Orange	Der 10 MHz-Ausgang ist gültig, jedoch der 1 PPS-Ausgang ist außerhalb der spezifizierten Genauigkeit
Rot	Weder 1 PPS– noch 10 MHz-Ausgang sind gültig: Dies kann ein Hinweis auf einen Fehler in der Verbindung zur Empfangseinheit sein

Folgende Buchsen sind vorhanden:

Bezeichnung	Buchsentyp	elektrische Spezifikation
1 PPS	SMA(f)	steigende Flanke bei TTL-Pegel (mit 2 mA belastbar)
10 MHz	SMA(f)	TTL-Pegel (mit 2 mA belastbar)
RS232-C	SUB-D9(f)	9polige RS232-C Schnittstelle für DCE (allerdings ist der 9. Pin mit der Versorgungsspannung der Timing-Unit belegt!)
Lichtwellenleiter DOWN-LINK	LWL-Buchse	
Lichtwellenleiter UP-LINK	LWL-Buchse	

Zu obiger Tabelle sind noch einige Anmerkungen notwendig. Der 10 MHz-Ausgang läßt sich über DIP-Schalter auch auf andere Frequenzen (4 kHz, 100 kHz, 500 kHz, 1 MHz, 5 MHz, 10 MHz) einstellen. Außerdem wurde die Platine der Timing-Unit so modifiziert, daß über dessen 96-poligen Steckerleiste die Stromversorgung zugeführt werden kann. (Und zwar über die 64-polige Buchsenleiste, siehe Abschnitt 6.2.1)

Für Details sei auf das Handbuch Nav [1995] verwiesen.

Empfangseinheit und Verbindungskabel

Statt eines Antennenkabels dienen als Verbindung zwischen der Empfangseinheit (Antenne und Empfänger) und der Timing-Unit zwei 20 m lange Lichtwellenleiter. Die Stromversorgung erfolgt über ein 10 m langes vieradriges Kabel. Im Handbuch wird die Einbindung eines Blitzschutzgerätes für die Stromversorgung empfohlen, was allerdings wegen der sehr geringen Gefahr eines naheliegenden Blitzes nicht erfolgt ist. (In diesem Fall müßten konsequenterweise auch die Antennen aller anderen Empfänger blitzgeschützt werden, was wegen des zu hohen Aufwandes nicht geschah.)

Kommunikation über RS232

Die Baudrate des Empfänger ist auf 2400 Baud bei Benutzung von 8 Datenbits, keinem Paritätsbit und einem Stopbit festgelegt. Modemstatusleitung (CTS, RTS) werden keine verwendet. Die Beschaltung der 9 poligen SUB-D9(f)-Buchse entspricht der eines DCE (*Data Communication Equipment*) **mit der Ausnahme, daß der 9-te Pin mit der Versorgungsspannung der Timing-Unit verbunden ist. (Dies um die Timing-Unit mit einem 1:1–Kabel von einem PC aus versorgen zu können). Wird die Timing-Unit von einer eigenen Spannungsversorgung gespeist, darf der 9. Pin nicht durchverbunden werden!**

Das verwendete Protokoll benutzt nur ASCII-Zeichen und hält die Nachrichten in einer für Menschen lesbaren Form. Die Nachrichten zum und vom Empfänger haben alle folgende Form:

`$PTELx`,*Sentence-ID*,*Werteliste*`<cr><lf>`

Optional können Nachrichten auch mit einer Prüfsumme versehen werden. Diese ist allerdings standardmäßig für Nachrichten vom Empfänger nicht aktiviert.

Das `x` steht für einen der folgenden Buchstaben `S, A, Q, R, P, E`. Die Bedeutung der Nachricht wird durch diesen Buchstaben und der *Sentence-ID* festgelegt:

Buchstabe	Bedeutung
`S`	Setze Systemparameter (Befehl)
`A`	Bestätigungsnachricht (A für Acknowledgement) für eine `$PTELS`-Befehl
`Q`	Abfragen der Werte von Systemparametern (Q für Query) (Befehl)
`R`	Antwort auf einen `$PTELQ`-Befehl
`P`	Periodische Nachricht vom Empfänger
`E`	Ereignisgesteuerte Nachricht vom Empfänger

Die *Sentence-ID* legt fest, auf welche Systemparameter oder –werte sich die Nachricht bezieht. Für den Einsatz im Meßsystem sind periodische Nachrichten von Interesse. Beinahe alle Antwortnachrichten auf Abfragebefehle (`$PTELR`) können auch periodisch ausgegeben werden. (Dann allerdings mit der Initialsequenz `$PTELP`):

Zwei *Sentence-IDs* sind hier von besonderem Interesse:

`$PTELP,GPS,`*a*`,`*b*`,`*c*`,`*d*`,`*OM*`,`*LAT*`,` *HEM*`,`*LONG*`,`*LP*`,`*AH*`,`*UTC-TIME*` ,`*UTC-DATE*`<cr><lf>`

Variable	Beschreibung
a,b,c,d	können ignoriert werden, Bedeutung siehe Handbuch des Empfängers
OM	Operating-Mode; `P`: Automatic Position Location Mode; `T`: Timing Mode
LAT	Latitude; (GPS-Ellipsoid–)Breite der Antenne in der Form: ddMM.mmm (dd: Grad, MM: Minuten, mmm: Minutenbruchteile)
HEM	Hemisphere; `N`: Nördliche Hemisphere, `S`: Südliche Hemisphere
LONG	Longitude; (GPS-Ellipsoid–)Länge der Antenne in der Form dddMM.mmm (ddd: Grad, MM: Minuten, mmm: Minutenbruchteile)
LP	Longitude Pole; `E`: Östlich Greenwich, `W`: Westlich Greenwich
AH	Antenna Height; Höhe der Antenne in Meter gegenüber MSL (Mean Sea Level)
UTC-TIME	UTC-Zeit in der Form hhmmss (hh: Stunden, mm: Minuten, ss: Sekunden)
UTC-DATE	UTC-Datum in der Form ddmmyy (dd: Tag, mm: Monat, yy: Jahr im Jahrhundert)

Beispiel. `$PTELP,GPS,V,1,1,T,T,4822.123,N,1611.123,E,220,122039,141295<cr><lf>`

Die zweite Nachricht, die wichtige Informationen, liefert ist folgende:

`$PTELP,SFM,`*a*`,`*b*`,`*c*`,`*d*`,`*ACC-POS*`,M,`*ACC-TIME*`,NS,`*GDOP*`,`*HDOP*`,`*TDOP*`,` *VER*`<cr><lf>`

Variable	Beschreibung
a,b,c,d	können ignoriert werden, Bedeutung siehe Handbuch des Empfängers
ACC-POS	Im Timing-Modus steht hier die Distanz zwischen den in den letzten zwei Mittelungsperioden ermittelten Positionen in Metern (als ganze Zahl mit maximale 5 Ziffern)
ACC-TIME	Hier steht einen ganze Zahl mit maximal fünf Ziffern, die die 2σ-Genauigkeit des 1 PPS-Ausgangs angibt. Eine Null steht für inakzeptable Genauigkeit bzw. nicht verfügbar
GDOP	General Dilution of Precision in der Form xx.y
HDOP	Horizontal Dilution of Precision in der Form xx.y
TDOP	Time Dilution of Precision in der Form xx.y
VER	Software Version in der Form x.y

Beispiel. `$PTELP,V,1,1,T,20,M,200,NS,1.5,1.2,1.4,2.9<cr><lf>`

Zu Beginn der Messung wird der Empfänger mit folgender Sequenz initialisiert:

```
$PTELS,CRR,SIV,0<cr><lf>
$PTELS,CRR,SAL,0<cr><lf>
$PTELS,CRR,GPS,1<cr><lf>
$PTELS,CRR,SFM,1<cr><lf>
```

Die Sentence-ID `CRR` dient zur Programmierung der periodischen Nachrichten. Hier werden lediglich die Nachrichten `SIV` und `SAL` ausgeschaltet und die Nachrichten `GPS` und `SFM` für eine Periodizität von 1 Hz programmiert.

6.2.4 Motorola VP Oncore Evaluation Kit

Allgemeines

Der VP Oncore ist ein relativ preisgünstiger Empfänger, der nicht eigens für Zeitanwendungen entwickelt wurde. Allerdings besitzt er einen 1 PPS-Ausgang und verspricht hohe Genauigkeiten: 130 ns (1σ), falls SA aktiv ist und 50 ns (1σ) für aktives SA im *position hold mode*.

Schnittstellen

Der VP Oncore Empfänger selbst besteht aus einer kleinen Platine mit den Ausmaßen 50,8 mm × 82,6 mm × 16,3 mm. Diese ist allerdings im VP Oncore Evaluation Kit in einem robusten schwarzen Aluminiumgehäuse untergebracht (Ausmaße 140 mm × 107 mm × 32 mm). Außerdem findet sich in diesem Gehäuse eine Spannungstabilisierung (12 V unstabilisert auf 5 V stabilisiert) und ein Pegelwandler für die RS232-C Schnittstelle des VP-Oncore.

Als Steckkontakte sind eine BNC(w)-Buchse für den Anschluß der Antenne und ein 9 poliger SUB-D9(m)-Stecker herausgeführt, die neben Stromversorgung und RS232 Schnittstelle auch den 1 PPS-Ausgang herausführt. Die genaue Belegung ist folgende:

Pin	Belegung
1	Schutzerde
2	RS232-RxD
3	RS232-TxD
4	*unbeschaltet*
5	1 PPS-Masse
6	5/12 V-Batteriezufuhr
7	Masse für 5/12 V Versorgung und RS232-Signale
8	12 V Versorgung
9	1 PPS-Signal

Um mit einheitlicheren Schnittstelle hantieren zu können, wurde der Motorola VP Oncore über eigene Kabel mit einer Rack-Einschubkarte verbunden (siehe Abschnitt 6.2.1), die über eine BNC(f)-Buchse den 1 PPS-Ausgang und über eine 25 polige SUB-D25(f)-Buchse die Kommunikationsleitungen der RS232-Schittstelle des Empfängers herausführt. Auf der Karte wird außerdem der 1 PPS-Puls mit dem IC 74LS125 getrieben.

Antenne und Antennenkabel

Die zylinderförmige Antenne mit einem Durchmesser von 102 mm und einer Höhe 22,6 mm ist steckerlos mit einem 6 m langen Antennenkabel (RG 58) verbunden, das mit einem BNC(m)-Stecker versehen ist. Da diese Länge für den Versuchsaufbau zu gering war, wurde er mit einer BNC-Kupplung und einem 4 m langen RG 58 Kabel verlängert (Beidseitig BNC(m)-Stecker). Die dadurch verursachte höhere Dämpfung ist (auch laut Hersteller) in der Praxis ohne weiteres tolerierbar. Über dieses Kabel wird die aktive Antenne mit 5 V versorgt (maximaler Stromverbrauch 25 mA).

Kommunikation über RS232

Der Empfänger bietet zur Kommunikation 3 Protokolle an:

2 ASCII-Protokolle (arbeiten nur mit ASCII-Zeichen) und ein Binärprotokoll:

Protokollname	Typ	Baud	Bits	Startbit	Paritätbits
Motorola	Binär	9600	8	1	Keine
NMEA	ASCII	4800	8	1	Keine
LORAN	ASCII	1200	8	1	Keine

Das interessanteste ist das Binärprotokoll, da es die größte Kontrolle erlaubt. Die anderen beiden entsprechen den Protokollstandards NMEA-0183 und LORAN.

Die nützlichste Nachricht im Motorola-Binärprotokoll ist die *Position-Status-Data*–Nachricht.

Die Form der Nachricht ist folgende:

`@@Ba` *Daten Prüfsumme*`<cr><lf>`

In Tabelle 6.2 ist das Format dieser Nachricht dargestellt (Die Länge wird in Einheiten von Bytes gemessen, Zahlen, die mehrere Bytes umfassen gehorchen dem Little-Endian-Format: Das höchstwertige Byte kommt hierbei zuerst; Negative Werte werden im Zweierkomplement dargestellt).

Diese Nachricht liefert also die Position der Antenne, den Sekundenanteil des 1 PPS-Pulses, die Zustände der Kanäle und den Status des Empfängers. Die Datenstruktur, die den Zustand jeweils eines Kanales beschreibt, ist im Handbuch Mot [1994] nachzuschlagen.

Das Bitfeld zum Status des Empfängers lautet wie folgt:

Bit	Bedeutung
7	Position Propagate Mode
6	Poor Geometry (DOP > 20)
5	3D fix
4	Altitude Hold (2D fix)
3	Acquiring Satellites/Position Hold
2	Differential
1	Insufficient visible satellites
0	Bad Almanac

Im Meßaufbau wird der Empfänger im sogenannten *Position Hold-Modus* betrieben. Dabei wird dem Empfänger die Position der Antenne mitgeteilt. Dadurch genügt das Tracken eines einzigen Satelliten um einen korrekten 1 PPS-Puls generieren zu können. (Dies entspricht dem Timing-Modus des Stellar– und des NavSymm-Empfängers, wobei diese die Position zuvor jedoch selbständig bestimmen).

Um diese genau Antennenposition zu erhalten, wurden die sekundenweise erhaltenen Positionen im Normalbetrieb (3D-fix-Mode) über einen Zeitraum von 24 Stunden gemittelt. Es wurden folgende GPS-Ellipsoidkoordinaten erhalten: Nördliche Breite: $48^\circ\ 11'46.''970$ N, Östliche Länge: $16^\circ\ 22'9.''509$ E, Höhe über GPS-Ellipsoid: 253.18 m.

Zur Initialisierung werden vier Befehle an den Empfänger gesandt:

- Weise Empfänger an, die Nachricht `@@Ba` (*Position-Data-Status*) periodisch jede Sekunde zu übermitteln (Befehl `@@Ba`)
- Setze Anwendungstyp auf Static[2] (Befehl `@@AB`)
- Setze Koordinaten für Position Hold-Modus auf obige Koordinaten (Befehl `@@As`)
- Schalte Position Hold-Modus ein (Befehl `@@At`)

Für tiefergehende Informationen schlage man im Handbuch Mot [1994] nach.

[2] In diesem Anwendungstyp geht der Empfänger davon aus, daß die Antenne ruht.

Bezeichnung	Länge	Wertebereich	Bedeutung
Monat	1	1 … 12	Monat des Jahres
Tag	1	1 … 31	Tag im Monat
Jahr	2	1900 … 2079	Jahreszahl
Stunden	1	0 … 23	Stunden am Tag
Minuten	1	0 … 59	Minuten in der Stunde
Sekunden	1	0 … 60	Sekunde in der Minute
Sekundenbruchteile	4	0 … 999999999	Sekundenbruchteile in ns
Breite	4	-324000000 … 324000000	GPS-Breite der Antenne in Tausendstel einer Bogensekunde
Länge	4	-648000000 … 648000000	GPS-Länge der Antenne in Tausendstel einer Bogensekunde
GPS-Höhe	4	-100000 … 1800000	GPS(Referenz-Ellipsoid)-Höhe der Antenne in cm
MSL-Höhe	4	-100000 … 1800000	MSL(Mean Sea Level)-Höhe der Antenne in cm
Geschwindigkeit	2	0 … 51400	Geschwindigkeit der Antenne in cm/s
Richtung	2	0 … 3599	Richtung der Geschwindigkeit in Zehntel Grad
DOP	2	0 … 999	Dilution of Precision in Zehntel
DOP-Typ	1	0 oder 1	0 = PDOP (im 3D-Modus), 1 = HDOP (im 2D-Modus)
Sichtbar	1	0 … 12	Anzahl der sichtbaren Satelliten
Getrackt	1	0 … 6	Anzahl der getrackten Satelliten
Kanal 1	4	Datenstruktur	Beschreibung des Zustands des Kanal 1
Kanal 2	4	Datenstruktur	Beschreibung des Zustands des Kanal 2
Kanal 3	4	Datenstruktur	Beschreibung des Zustands des Kanal 3
Kanal 4	4	Datenstruktur	Beschreibung des Zustands des Kanal 4
Kanal 5	4	Datenstruktur	Beschreibung des Zustands des Kanal 5
Kanal 6	4	Datenstruktur	Beschreibung des Zustands des Kanal 6
Status	1	Bitfeld	Zustand des Empfängers

Tabelle 6.2: `Ba`–Nachricht des Binärprotokolls des VP-Oncore

6.2.5 Trimble SVeeSix-CM2 Starter Kit

Allgemeines

Dieser preisgünstige Empfänger, der lediglich aus einer kleinen Platine besteht (nicht speziell für Zeitanwendungen entwickelt), besitzt lediglich einen 1 PPS-Ausgang. Seine Maße sind 82,6 mm × 46,5 mm × 17,8 mm.

Schnittstellen

Der Empfänger besteht aus einer Platine mit einem SMB(m)-Stecker zum Anschluß der Antenne und einem 8-poligen Feldstecker (2 × 4), der folgende Belegung besitzt:

Pin	Belegung
1	5 V= Stromversorgung für Antenne
2	5 V= Stromversorgung
3	TxD (TTL-Pegel)
4	Battery Backup
5	RxD (TTL-Pegel)
6	1 PPS (TTL-Pegel)
7	*nicht beschaltet*
8	Masse

Genauso wie beim Motorola wurde eine eigene Einschubkarte für das 19″-Rack angefertigt, um einheitliche Schnittstellen zu garantieren. Hier konnte die Platine aufgrund ihrer geringen Größe direkt auf der Trägerplatine untergebracht werden.

Dabei wurde der 1 PPS-Puls über einen Treiber geführt (74LS125), um ein ausreichend starkes Signal zu erhalten. Die Stromversorgung wurde wieder über einen 64 pol. VG-Stecker zugeführt (gleiche Belegung wie beim Motorola). Die RS232-Kommunikationsleitungen (mit TTL-Pegel) werden über die beim Starter-Kit mitgelieferte Platine geführt, um die gewünschten RS232-Pegel zu erhalten. Allerdings sind bei der 25 pol. SUB-D25(f)-Buchse gegenüber den anderen Rack-Einschüben die Leitungen RxD und TxD vertauscht.

Antenne und Antennenkabel

Als Antenne wurde nicht die mitgelieferte Antenne verwendet, da deren Kabellänge lediglich 5 m beträgt. Stattdessen wurde eine „Rooftop“-Antenne besorgt, die ein Antennenkabel mit einer Länge von 25 m besitzt. Die Rooftop-Antenne ist mit einer F(f)-Buchse versehen. Entsprechend besitzt das Antennenkabel beidseitig einen F(m)-Stecker. Zur Verbindung mit dem Empfänger wird bei der „Rooftop“-Antenne ein Koaxialübergang mitgeliefert, der auf einer Seite eine F(f)-Kabelbuchse und auf der anderen Seite eine SMB(f)-Kabelbuchse besitzt. Die aktiv Antenne wird über das Antennenkabel mit 5 V versorgt.

Kommunikation über RS232

Der Empfänger ermöglicht die Kommunikation über 3 Protokolle: TSIP (Trimble Standard Interface Protokoll), ein Binärprotokoll; TAIP (Trimble ASCII Interface Protokoll), ein ASCII-Protokoll; NMEA0183 (Standardprotokoll). Aufgrund der Komplexität des TSIP und der Tatsache, daß in diesem Modus der 1 PPS-Puls nicht exakt synchron zur Zeitskala läuft, sondern die Abweichung des Pulses über RS232 nachträglich bekanntgegeben wird, und der geringen Kontrollmöglichkeit beim NMEA0183 wurde das TAIP für die Verwendung im Versuchsaufbau ausgewählt.

Das TAIP arbeitet mit einer Baudrate von 4800 Baud, 8 Datenbits, einem Stopbit und keinem Paritätsbit.

Es sind zwei Nachrichten, die die wesentlichen Informationen tragen. Die erste ist die Time/Date `TM`-Nachricht:

`>RTM`*AABBCCDDDEEFFGGGGHHIJJKLLLLL*`<`

Das `>` und das `<` sind Begrenzer, `R` steht für Response und `TM` bezeichnet, welche Nachricht vorliegt.

Jeder kursive Buchstabe steht dabei für ein ASCII-Zeichen:

Zeichen	Bedeutung
AA	UTC-Stundenwert
BB	UTC-Minutenwert
CC.DDD	UTC-Sekundenwert (mit Tausendstel Bruchteilen)
EE	UTC-Tag (Tag im Monat)
FF	UTC-Monat
GGGG	UTC-Jahr
HH	GPS/UTC-Time Offset, der Differenzbetrag zwischen den beiden Zeitskalen durch die Schaltsekunden hervorgerufene Differenzbetrag
I	Current Fix Source, (0=2D GPS, 1=3D GPS, 2=2D DGPS, 3=3D DGPS, 6=DR (Direct Reference), 8=Degraded DR, 9=Unbekannt)
JJ	Anzahl verwendbarer Satelliten
K	GPS/UTC Offset-Flag, bestimmt, ob die angegebene UTC-Zeit gültig (=1) oder auch nicht (=0) ist
LLLLL	*reserviert*

Beispiel. `>RTM123445.0010104199510104100000<`

Die zweite Nachricht ist die Status `ST`-Nachricht:

`>RST`*AABCDDEFGG*`<`

Zeichen	Bedeutung
AA	(00=Doing Position fixes, 01=Don't have GPS time yet, 02=not used, 03=PDOP is too high, 08=No usable satellites, 09=Only 1 usable satellites, 0A=Only 2 usable satellites, 0B=Only 3 usable satellites
B	Error Code Nibble 1
C	Error Code Nibble 2
DD	Machine ID
E	Error Code Nibble 3
F	Error Code Nibble 4
GG	*reserviert*

Beispiel. `>RST00000000<`

Die genaue Beschreibung sei dem Handbuch entnommen (Tri [1994]).

Der Trimble-Empfänger wird nicht wie die Empfänger von Stellar, NavSymm und Motorola im Time Transfer-Modus (oder auch Position-Hold-Mode) betrieben. Er ermittelt nämlich Zeit und Position aus der regelmäßigen Vermessung von 4 Satelliten (4 Satelliten-Lösung).

6.2.6 Magellan OEM GPS Module PS

Allgemeines

Der Magellan-Empfänger ist sehr preisgünstig und besitzt neben einem 1 PPS-Puls auch einen 1 kPPS-Puls. Die Maße der Platine sind 81,5 mm × 188,0 mm × 23,9 mm.

Schnittstellen

Zur Aufnahme der Antennensignale besitzt die Platine eine SMB(m)-Buchse. Zwei Feldstecker (1×7 und 1×10) stellen die Schnittstelle zur Platine her. Die Belegung des einen (J1) ist folgende:

Pin	Belegung
1	VBat
2	1 PPS
3	VExt
4	GND
5	GND
6	1 kPPS
7	On/Off

Die Belegung des zweiten Feldsteckers (J2) ist folgende:

Pin	Belegung
1	NMEA Out
2	NMEA IN B/Ground
3	MMEA IN A
4	RS422 Rx IN +
5	RS422 Rx IN -
6	RS232 OUT/RS422 OUT +
7	RS232 IN/RS422 OUT -
8	RS232 OUT
9	Ground
10	RS232 IN/0-5 V CMOS

Auch für den Magellan wurde eine eigene Einschubkarte für das 19″-Rack angefertigt Die Platine mußte wegen ihrer Größe allerdings direkt am Rack montiert werden

Der 1 PPS-Puls wurde auf der Einschubkarte über einen Treiber geführt (74LS125), um ein ausreichend starkes Signal zu erhalten. Die Stromversorgung wurde über den 64 pol. VG-Stecker zugeführt. Die RS232-Kommunikationsleitungen wurden einfach an die 25 pol.SUB-D25(f)-Buchse durchgeschliffen.

Antenne und Antennenkabel

Die Antenne hat die Form eines halbierten Ellipsoides mit einer vertikalen Symmetrieachse. Der Durchmesser des Ellipsoids beträgt 9,2 cm, die Höhe 10,2 cm. An der Unterseite ist eine TNC(f)-Buchse herausgeführt. Mitgeliefert wurde ein Koaxialkabel mit der Länge von 15 m. Beidseitig ist das Kabel mit einem TNC(m)-Stecker versehen. Ein Übergangskabel mit TNC(f) und SMB(f) zum Anschluß an den Empfänger war im Lieferumfang enthalten.

Kommunikation über RS232

Die meisten Nachrichten können beim Magellan sowohl in Binärform als auch in ASCII-Form übermittelt werden. Der Magellan wird in der Default-Einstellung mit 9600 Baud, 8 Datenbits, keinem Paritätsbit und einem Stopbit betrieben.

Die Nachricht, die bei der Untersuchung verwendet wird, ist die *Time Transfer Message* im Binärformat.

Die Form der Nachricht ist folgende:

`$$K3` *Daten Checksumme* `<lf>`

Die ersten beiden `$$` kennzeichnen den Beginn einer Binärnachricht. `K3` ist die ID der Nachricht und `<lf>` ist ein Line-Feed (ASCII-10), das die Nachricht abschließt. Die *Checksumme* besteht aus einem Zeichen und ist das Exklusiv-Oder aller Bytes nach `$$` und vor der Checksumme. Die *Daten* bestehen aus 17 Bytes mit folgender Bedeutung (Daten hängen von der gewählten Zeitskala UTC oder GPS ab):

Länge in Bytes	Bedeutung
1	UTC/GPS-Stunde
1	UTC/GPS-Minute
1	UTC/GPS-Sekunde
1	Tag im Monat
1	Monat im Jahr
2	Jahr (MSB-Byte zuerst)
1	1=normal, 0=coasting
2	Satellitennummer und Signalrauschen Kanal 1
2	Satellitennummer und Signalrauschen Kanal 2
2	Satellitennummer und Signalrauschen Kanal 3
2	Satellitennummer und Signalrauschen Kanal 4
1	Time Figure of Merit, TFOM

Die Bedeutung von TFOM ergibt sich aus nachstehender Tabelle:

TFOM	Erwarteter Zeitfehler x
1	$0 < x < 1$ (ns)
2	$1 < x < 10$ (ns)
3	$10 < x < 100$ (ns)
4	$100 < x < 1000$ (ns)
5	$1 < x < 10$ (μs)
6	$10 < x < 100$ (μs)
7	$100 < x < 1000$ (μs)
8	$1 < x < 10$ (ms)
9	$10 < x < 100$ (ms)
10-14	*unbenutzt*
15	keine Information verfügbar

Genauso, wie der Trimble-Empfänger, ermittelt der Empfänger von Magellan Zeit und Position über eine 4 Satellitenlösung.

Für den Meßaufbau wurde übrigens die Zeitskala GPS gewählt.

6.2.7 Rockwell Microtracker

Allgemeines

Der Rockwell Microtracker ist eine sehr kleine Platine, die zwar einen Timing-Ausgang besitzt, der aber nicht mit einer Zeitskala synchron läuft. Dieser Ausgang ist vielmehr eine vom internen Quarz abgeleitete Frequenz von 1 Hz. Über RS232 wird der Offset des durch die Flanke dieses Timing-Ausgangs (Time Mark-Puls) definierten Zeitpunkts relativ zum Sekundenbeginn der gewünschten Zeitskala (GPST oder UTC) übermittelt. Dieser Empfänger ist eine Leihgabe der Fernuniversität Hagen.

Pin	Belegung
1	Masse
2	*reserviert*
3	Masse
4	*reserviert*
5	Run/Keep-Alive Control (mit 5 V zu beschalten)
6	Masse
7	Time Mark, Steigende Flanke ist relevant
8	Masse
9	Power Monitor
10-13	*reserviert*
14	Baudrate
15	Versorgung 5 V
16	Masse
17	Serial Data In (TTL-Pegel)
18	Masse
19	Serial Data Out (TTL-Pegel)
20	Masse
21	Versorgung 5 V
22	Masse
23	Versorgung Antenne (max. 12 V bei 100 mA) Hier mit 5 V
24	Masse
25	*reserviert*
26	Versorgung 5 V

Tabelle 6.3: Belegung des ZIF-Connectors des Rockwellempfängers

Schnittstellen

Die Platine hat die Ausmaße 25,4 mm × 71,1 mm × 13,5 mm. Die Antennensignale werden über eine MCx(f)-Buchse zugeführt. Die Antenne wird über diesen Kontakt mit 5 V versorgt. Daneben besitzt er einen ungewöhnlichen 26 poligen Miniaturstecker mit der Bezeichnung 1 MM PITCH FPC Connector Zero Insertion Force(ZIF)-Type mit 26 Kontakten (erhältlich von der Firma ELCO International).

Die Belegung des 26 poligen Steckers ist in Tabelle 6.3 wiedergegeben.

Die 5 V–Versorgungsspannungen müssen übrigens allesamt beschaltet werden, damit der Empfänger auch funktioniert. Der Microtracker konnte direkt auf einer 19” Rackeinschubkarte untergebracht werden. Dabei wurde ein Treiber (74LS125) für den Time Mark-Puls und ein Pegelwandler (TTL zu RS232–Pegeln) für die RS232-Kommunikationsleitungen eingebaut.

Antenne und Antennenkabel

Die für den Microtracker eingesetzte Antenne ist die L1 GPS Vehicle Tracking Antenna P/N 90LL13800. Diese aktive Antenne benötigt maximal 20 mA bei 5 V. Aus der Antenne ragt ein kurzes Kabelstück heraus, an dem eine MCx(f)-Kabelbuchse montiert ist. Mit einem 10 m langen RG 316 Kabel mit MCx(m)-Steckern an den Enden wird die Verbindung zwischen Antenne und Empfänger hergestellt.

Kommunikation über RS232

Über den Pin 14 des 26 poligen Steckers kann die Baudrate eingestellt werden. Im Meßaufbau ist er unbeschaltet. Dadurch ist die Baudrate für eingehende und ausgehende Nachrichten auf 9600 Baud festgelegt.

Es werden 8 Datenbits, ein Stopbit und ein Paritätsbit mit ungerader Parität benutzt.

Zur Ermittlung des Zeitanteils wird die alle Sekunden gelieferte Nachricht *Position, Velocity, Time Solutions and Status*-Message benutzt. Die für die Charakterisierung des Time-Mark-Pulses wichtigen Felder der Nachricht sind in folgender Tabelle dargestellt:

Bezeichnung	Offset/Bytes	Länge/Bytes	Erklärung
GPS Time of Week	10	6	Gleitkommazahl in Rockwell-eigenem Format. Kennzeichnet den genauen Zeitpunkt des Time-Mark-Pulses mit einer Auflösung von 1 μs
GPS Week Number	16	2	Integerwert für die GPS-Woche des Time-Mark-Pulses
UTC-Validity	18	2	Falls der Wert ungleich Null, Daten gültig

Genauso wie der Trimble-Empfänger ermittelt der Empfänger von Rockwell Zeit und Position über eine 4-Satellitenlösung.

Die Beschreibung des Empfängers ist im Dokument Roc [1993] enthalten.

6.2.8 Elektrische Spezifikation der 1 PPS-Pulse

In Tabelle 6.4 sind die elektrischen Spezifikationen der 1 PPS-Pulse der verschiedenen Empfänger wiedergegeben. Die angegebenen Daten beschreiben die Pulsweite und die Spannungspegel des Pulses, die Anstiegszeit der Flanke und welche der beiden Flanken die gewünschte Zeitskala repräsentiert (steigende oder fallende Flanke).

Empfänger	Pulsweite	elektrische Spez.	Anstiegszeit	Flanke
Stellar	1 μs	0 V $\rightarrow$ 10 V an 50 Ω	10 ns	steigend
NavSymm	250 ms	CMOS TTL	10 ns	steigend
Motorola	200 ms	0 V $\rightarrow$ 5 V	20–30 ns	steigend
Trimble	1 μs	CMOS TTL	20 ns	steigend
Magellan	100 ns	TTL	10 ns	steigend
Rockwell	20 μs	CMOS TTL	20 ns	steigend

Tabelle 6.4: Elektrische Spezifikation der 1 PPS-Pulse

6.2.9 Spezifizierte Genauigkeiten der 1 PPS-Pulse

Die Genauigkeiten zu den 1 PPS-Pulsen werden recht unterschiedlich angegeben. Zum einen gibt es Angaben, die für den Fall der Aktivierung des SA (siehe 3.6.3) gelten und andererseits solche die für inaktives SA relevant sind. Weiters ist es wichtig, zu wissen, welche Zeitskalen der Empfänger darzustellen vermag. Diese Daten sind in der Tabelle 6.5 zusammengefaßt.

Empfänger	Genauigkeit (mit SA)	Genauigkeit (ohne SA)	Zeitskala
Stellar	300 ns[a]	100 ns	GPST oder UTC
NavSymm	250 ns (2σ)		UTC
Motorola	50 ns (1σ)[b]		GPST
Trimble	typisch 1 μs, worst case 5 μs		GPST oder UTC
Magellan		$\pm$100 ns	GPST oder UTC
Rockwell		1 μs	GPST oder UTC

[a] Der Jitter des 1 PPS ist < 1 ns rms
[b] Diese Genauigkeit gilt allerdings nur im *position-hold-mode*. Für den normalen Betrieb mit Positionsbestimmung beträgt die Genauigkeit $\sigma = 130$ ns.

Tabelle 6.5: Genauigkeit der Pulse

6.2.10 Systematische Abweichungen der 1 PPS-Pulse

Initialpositionen

Die Empfänger Stellar, NavSymm und Motorola werden im *Time Transfer Modus* betrieben. Dabei gehen sie von einer bekannten *Initialposition* aus und benutzen diese zur Rekonstruktion der gewünschten Zeitskala. Die Empfänger berechnen diese Position automatisch durch langfristige Mittelung. Da sie unabhängig voneinander zu diesen Positionen gelangt sind, unterscheiden sich diese voneinander. Dies kann natürlich Ursache für systematische (und auch stochastische) Abweichungen sein.

Die von den Empfängern ermittelten Initialpositionen im WGS-84–Koordinatensystem sind in Tabelle 6.6 aufgelistet.

Empfänger	Breite	Länge	Höhe
Stellar	$48°11'46.''400$ N	$16°22'9.''500$ E	239.00 m
NavSymm	$48°11'46.''380$ N	$16°22'9.''000$ E	257.00 m
Motorola	$48°11'46.''970$ N	$16°22'9.''509$ E	253.18 m

Tabelle 6.6: Initialpositionen der Empfänger Stellar, NavSymm und Motorola im WGS-84

Errichtet man ein Koordinatensystem, dessen x-Achse nach Osten, y-Achse nach Norden und z-Achse in Richtung des Zenits weist und normiert man die Position des Stellar auf den Ursprung des Koordinatensystem, so erhält man die Position in der in Tabelle 6.7 gezeigten Form.

Empfänger	x	y	z
Stellar	0	0	0
NavSymm	-10 m	-0,6 m	+18 m
Motorola	+2 m	+18 m	+14 m

Tabelle 6.7: Initialpositionen der Empfänger Stellar, NavSymm und Motorola in einem topozentrischen Koordinatensystem

Berechnet man die gegenseitigen Abstände dieser Initialpositionen, so stellt man fest, daß sie in Form eines gleichseitigen Dreiecks mit der Seitenlänge 21 m angeordnet sind. Die maximal mögliche Abweichung durch die unterschiedlichen Initialpositionen beträgt demnach 60 ns (1 m entspricht 3 ns). Da sich die Satellitenposition normalerweise etwa gleich in positiver und negativer x und y-Richtung verteilen, in z-Richtung allerdings nur in positiver Achsenrichtung auftreten, wird der Hauptanteil systematischer Abweichungen durch die z-Komponente der Initalposition bestimmt: Die über diesen Weg ermittelte maximal mögliche systematische Abweichung zwischen dem Stellar-Empfänger und dem Motorola- bzw. NavSymm-Empfänger beträgt daher 54 ns bzw. 42 ns, zwischen den letzten beiden hingegen nur 12 ns. Da der Einfallswinkel der Satellitensignale nicht immer 90° beträgt, wird die tatsächliche Abweichung entsprechend kleiner sein.

Verzögerungen durch Antennenkabel

Manche Empfänger berücksichtigen die Verzögerung über das Antennenkabel automatisch. Dies ist der Fall für die Empfänger von Stellar, NavSymm und Magellan. Für den Motorola–, Trimble– und Rockwell–Empfänger müssen die durch das Kabel eingebrachten Verzögerungen (in den verwendeten Koaxialkabeln breiten sich die Signale mit 2/3 der Lichtgeschwindigkeit aus) in der Auswertung berücksichtigt werden. In Tabelle 6.8 sind diese gemeinsam mit den Kabellängen angegeben.

Empfänger	Kabellänge/m	Verzögerung/ns
Motorola	10	45
Trimble	25	110
Rockwell	10	45

Tabelle 6.8: Verzögerungen durch die Laufzeit in den Antennenkabeln

6.3 Signalvorverarbeitung

Das 10 MHz-Signal von der Referenzuhr und die 10 MHz-Signale der GPS-Empfänger müssen in geeigneter Weise auf eine niedrigere Frequenz herabgesetzt werden, um bei der Messung entsprechend weniger Flanken — und damit auch Daten — zu erhalten. Konkret werden aus dem 10 MHz Ausgangsignal der Referenzuhr 100 Hz (REF100), 1 Hz (REF1) und ein Triggersignal (TRIGGER) erzeugt (siehe Abbildung 6.1), dessen Periodendauer in Stufen von Zweierpotenzen über einen Jumper auf der Frequenzteilerschaltung einstellbar ist (mögliche Periodendauern: 2, 4, 8, 16, 32, 64, 128 s). In der vorliegenden Messung wurde eine Periodendauer von 8 s gewählt. Dieses Triggersignal dient, wie der Name vermuten läßt, der Triggerung der Logikanalysatoren im alternierenden Betrieb (siehe Signalanalyse).

Die 10 MHz-Frequenzausgänge der GPS-Empfänger werden auf eine Frequenz von 1 Hz herabgeteilt, damit sie genauso wie die 1 PPS-Signale behandelt werden können.

Neben diesen Frequenzumwandlungen waren auch einfache Signalanpassungen von 1 PPS und 10 MHz-Signalen notwendig, die aus Praktikabilitätsgründen auf derselben Schaltung realisiert wurden. Der schematische Aufbau dieser Frequenzteilerschaltung, die von Fritz Mörth im Rahmen des Projekts SynUTC angefertigt wurde, ist in Abbildung 6.7 ersichtlich. Die detaillierten Schaltpläne sind dem Anhang Abschnitt C.2 zu entnehmen.

Die Schaltung besitzt 10 BNC(f)-Buchsen als Eingänge für 1 PPS-Signale. Alle diese Eingänge werden — mit Ausnahme der ersten Eingangs 1PPS(1), der über einen Spannungsteiler geführt wird — unverändert an die beiden Logikanalysatoren weitergeführt. (Grund: der Stellar-Empfänger gibt über den 1 PPS-Ausgang nicht 5 sondern 10 V aus.) In der Abbildung nicht enthalten sind außerdem die Monoflops und LEDs, die von den 1 PPS-Signalen angesteuert werden und lediglich der optischen Kontrolle der anliegenden Signale dienen.

Das Referenzsignal von 10 MHz, das über eine BNC(f)-Buchse aufgenommen wird, ist mit einem Widerstand von 50 Ω abgeschlossen. Durch Frequenzteilung werden daraus die drei Signale REF100, REF1 und TRIGGER gewonnen.

Die 10 MHz-Signale der GPS-Empfänger gelangen ebenfalls über eine BNC(f)-Buchse in die Schaltung und werden auf 1 Hz herabgeteilt. Diese Signale und auch das REF1-Signal werden über Monoflops und LEDs optisch angezeigt.

Aus den 14 Eingangsignalen wurden damit 16 Ausgangsignale gewonnen, die über geeignete Stecker an die Logikanalysatoren weitergeleitet werden. Dabei wurde eine Zuordnung der Kanäle getroffen (für beide Logikanalysatoren gleich), wie sie in Tabelle 6.9 dargestellt ist.

Signal	GPS-Empfänger	Kanal im Logikanalysator
1PPS(1)	Stellar	6
1PPS(2)	NavSymm	8
1PPS(3)	Motorola	7
1PPS(4)	Trimble	10
1PPS(5)	Magellan	9
1PPS(6)	Rockwell	12
1PPS(7)	*unbenutzt*	11
1PPS(8)	*unbenutzt*	13
1PPS(9)	*unbenutzt*	14
1PPS(10)	*unbenutzt*	15
REF100		0
REF1		1
TRIGGER		2
10MHZ(1)	Stellar	3
10MHZ(2)	NavSymm	4
10MHZ(3)	*unbenutzt*	5

Tabelle 6.9: Zuordnung der Meßsignale zu den Kanälen der Logikanalysatoren

Um eine zeitliche Synchronisation mit dem Meß-PC zu erreichen, wurde das TRIGGER-Signal zusätzlich als open-emitter-Ausgang ausgeführt und über eine Cinch-Buchse herausgeführt. Dieser wird schließlich mit der interruptfähigen ACK-Leitung der parallelen Schnittstelle LPT1 verbunden, sodaß bei jeder steigenden Flanke des TRIGGER-Signals ein Interrupt im Meß-PC ausgelöst wird.

6.4 Signalanalyse

Von der Signalvorverarbeitungsstufe werden 16 Signale geliefert, deren Zeitverhalten erfaßt werden soll. Um eine lückenlose Datenerfassung sicherzustellen, werden zwei Logikanalysatoren HP16500B verwendet, wobei abwechselnd jeweils ein Gerät mißt, während aus dem anderen die Daten ausgelesen werden können. Dieses Abwechseln wird durch das TRIGGER-Signal und damit durch die Referenzuhr gesteuert (siehe Abbildung 6.8)

Der Meßvorgang der Logikanalysatoren in dem zum Einsatz kommenden Transitional Full Channel 125 MHz Mode ist folgender: Der Logikanalysator muß nachdem er initialisiert wurde, für jeden einzelnen Meßvorgang gestartet werden, so daß er überhaupt auf das Eintreten der sogenannten Triggerbedingung reagieren kann. Im vorliegenden Meßystem ist die Triggerbedingung derart, daß sie für den ersten Logikanalysator bei steigender Flanke des TRIGGER-Signals und entsprechend beim zweiten Logikanalysator bei fallender Flanke desselben erfüllt ist. Sobald die Triggerbedingung erfüllt wurde, registriert der Logikanalysator jede Signaländerung an irgendeinem Eingangskanal mit einer Granularität von 8 ns (Abtastfrequenz 125 MHz) und zeichnet den Zustand jedes Signals zusammen mit einem Zeitstempel in einem internen Speicher auf. Sobald dieser Speicher voll ist, stoppt der Logikanalysator automatisch den Aufzeichnungsvorgang. Der Logikanalysator wird übrigens im Pretriggermodus betrieben, sodaß er auch Daten kurz vor dem Trigger aufzeichnet.

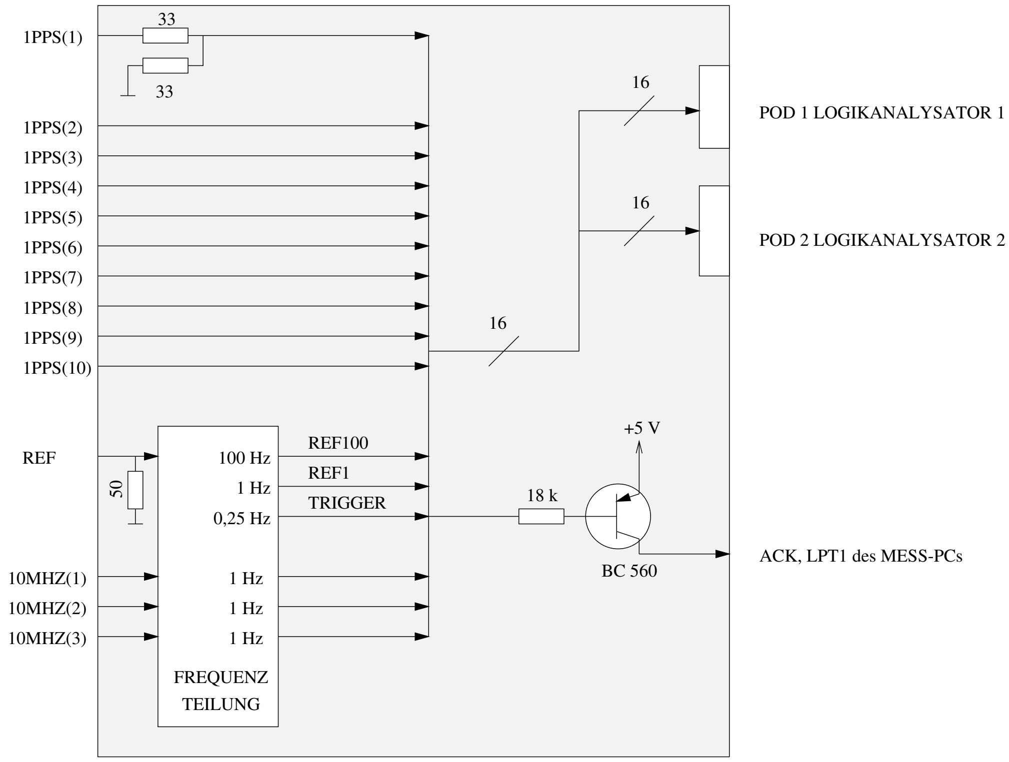

Abbildung 6.7: Die Frequenzteilerschaltung

Das Ansprechen der Logikanalysatoren erfolgt über eine HPIB-Bus Schnittstelle. Die drei wesentlichen Befehle, die über diese Schnittstelle gesandt werden müssen, sind:

- Starten des Logikanalysators
- Stoppen des Logikanalysators
- Auslesen der Daten im Logikanalysator

Die Steuerung der Logikanalysatoren erfolgt durch den Meß-PC. Aus den erhaltenen Daten müssen schließlich die von den lokalen Uhren der Logikanalysatoren erhaltenen Zeitstempel auf die Referenzuhr umgerechnet werden. Zu diesem Zweck kommt das REF100-Signal zum Einsatz. Diese Verarbeitung der Daten wird aber im Kapitel über Software behandelt.

6.5 Meßwertverarbeitung

6.5.1 8-fach serielle Schnittstellenkarte BOCA–BB1008

Die 8-fach serielle Schnittstellenkarte BB1008 von Boca wird zur Kommunikation mit den GPS-Empfängern und dem Server-PC eingesetzt. Dabei wurden die 8 Ports mit folgenden Kommunikationspartnern verbunden:

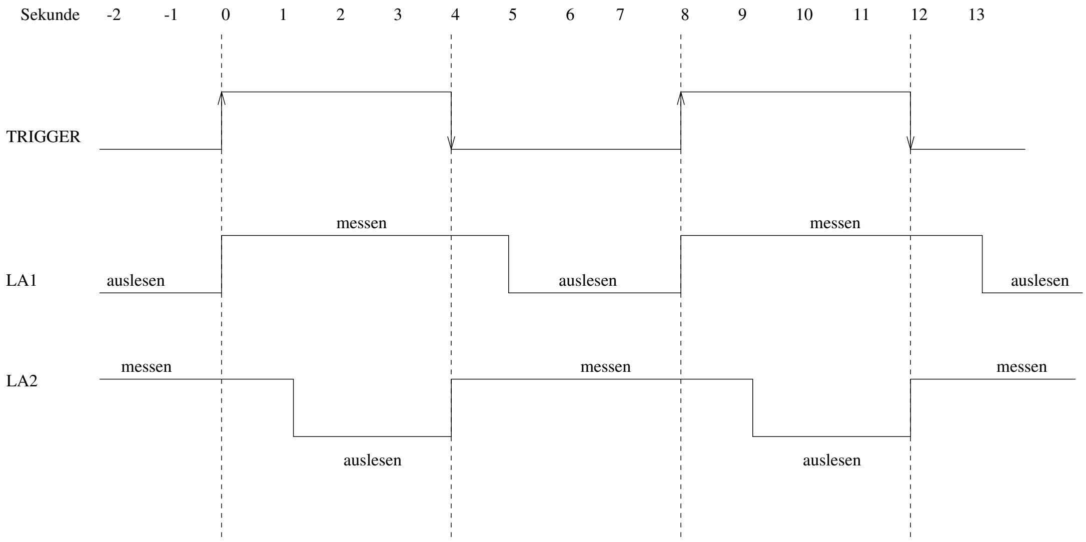

Abbildung 6.8: Der alternierende Betrieb der Logikanalysatoren

Port	Kommunikationspartner
1	Stellar
2	NavSymm
3	Motorola
4	Trimble
5	Magellan
6	Rockwell
7	*unbenutzt*
8	Server-PC

Die Ports sind als 6-polige RJ12-Buchsen ausgeführt. Mit der 8-fach Schnittelle wurden 8 Kabel mit beidseitig montierten 6-poligen RJ12-Steckern und 8 Übergänge auf 25 polige SUB-D25(m)-Stecker geliefert. Daher wurden die Rackeinschubkarten der Empfänger von Motorola, Trimble, Magellan und Rockwell mit 25 poligen SUB-D25(f) Steckern ausgestattet. Für die Empfänger von Stellar und NavSymm wurden die mitgelieferten Kabel einfach auf einer Seite mit den entsprechenden 9 poligen SUB-D9(m)-Steckern versehen.

6.5.2 Software

Auf dem Meß-PC läuft das Programm `GPSEVAL.EXE`, das die Meßwertverarbeitung und die Übermittlung der Daten über die RS232-Schnittstelle an den Server-PC übernimmt. Es ist im Kapitel 7 detailliert beschrieben.

6.6 Speicherung und Übermittlung

6.6.1 Ethernetkarte

Zur Übermittlung der Daten in das lokale Netz (TU-Net) besitzt der Server-PC eine Ethernetkarte. Der Rechner erhielt zur einfachen Übermittlung mittels FTP-Service eine eigene Internetadresse.

6.6.2 Software

Die Übernahme der Daten vom Meß-PC und die Speicherung auf der lokalen Festplatte geschieht am Server-PCs. Das Programm, das zu diesem Zwecke auf dem Server-PC läuft heißt `SERVER.EXE`. Auch dieses ist im Kapitel 7 beschrieben.

7 Software zum Betrieb des Meßsystems

Die verwendete Software besteht aus dem Programm `GPSEVAL.EXE` auf dem Meß-PC und dem Programm `SERVER.EXE` auf dem Server-PC, die beide mit Borland C++ 3.0 übersetzt und gelinkt wurden.

7.1 Programm `GPSEVAL.EXE`

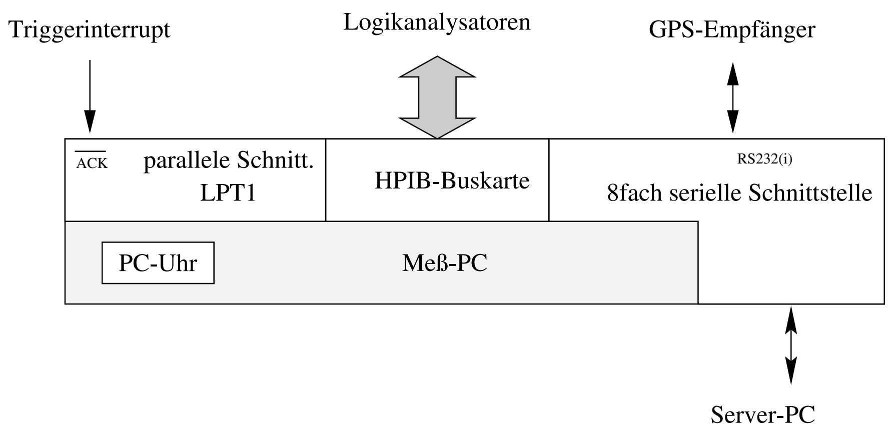

Abbildung 7.1: Schnittstellen zum Meß-PC

Das Programm `GPSEVAL.EXE` läuft auf dem Meß-PC, der eine Zahl von Geräten steuern muß, siehe Abbildung 7.1. Das Hauptmodul des Programms ist `projekt.cpp`. In Tabelle 7.1 ist eine Übersicht aller Module des Programms dargestellt.

In den nächsten Abschnitten wird die Funktionsweise des Programms näher beschrieben.

7.1.1 Initialisierung

Nach dem Start des Programms wird zunächst die Bildschirmmaske dargestellt, siehe Abbildung 7.2.

Modul	Zweck
`gpseval.cpp`	Hauptprogramm
`global.cpp`	Modul für globale Variable und Werte
`sys.cpp`	Modul zur Steuerung des Systems
`la.cpp`	Modul zur Verarbeitung der Daten von den Logikanalysatoren
`com.cpp`	Modul zur Verarbeitung der Daten von den GPS-Empfängern
`io.cpp`	Modul zur Bedienung des Triggerinterrupts über LPT1
`save.cpp`	Modul zur Zwischenspeicherung der gewonnen Pulsdaten
`timer.cpp`	Treibermodul zur Bedienung des PC-Timers
`serialbo.cpp`	Treibermodul zur Bedienung der 8-fach seriellen Schnittstellenkarte
`out.cpp`	Treibermodul zur einfachen Ausgabe am Textbildschirm des PC
`hp16500.cpp`	Treibermodul zur Steuerung der Logikanalysatoren
`ieee488.lib`	Bibliothek zur Kommunikation über die HPIB-Schnittstelle

Tabelle 7.1: Module des Programms `GPSEVAL.EXE`

Danach erfolgt die Initialisierung der einzelnen Systemkomponenten wie folgt (dies geschieht im Modul `sys.cpp`):

1. **Initialisierung der Logikanalysatoren.** Die Logikanalysatoren werden angewiesen, ihre Konfigurationsdateien zu laden.
2. **Starten der Logikanalysatoren.** Die Logikanalysatoren werden gestartet, um auf den ersten Triggerpuls zu warten und daraufhin Daten zu sampeln.
3. **RS232 und GPS-Empfänger initialisieren.** Die 8-fach RS232-Schnittstellenkarte wird initialisiert und den GPS-Empfängern werden über RS232 Befehle zugesandt, um sie anzuweisen, die gewünschten Nachrichten zu senden.
4. **Triggerinterrupt initialisieren.** Die parallele Schnittstelle LPT1 wird initialisiert und die ACK-Leitung der LPT1 mit einer Interruptroutine verbunden.

Wurden die Systemkomponenten initialisiert, so wird in der Folge auf das Eintreffen des ersten Triggerinterrupts gewartet. Tritt dieser nicht vor Ablauf von 8,01 s seit Aktivieren der Triggerinterruptroutine (Die Toleranz vom 10 ms ist erforderlich, da der Frequenzfehler der PC-Uhr noch nicht bekannt ist) auf, so muß etwas schiefgelaufen sein; das Programm bricht mit der Fehlermeldung `Timeout bei Warten auf 1. oder 2. Triggerinterrupt` ab. Entsprechend wird in der Folge auf den zweiten Triggerinterrupt gewartet. Sollte dieser nicht innerhalb von 8,01 s nach dem 1. Triggerinterrupt eintreffen, so bricht das Programm mit obiger Fehlermeldung ab.

Sind beide Triggerinterrupts korrekt eingelangt, so wartet das System auf das erste Datenpaket von einem der Logikanalysatoren und geht, sofern dieses rechtzeitig eintrifft, in den normalen Betriebsmodus über.

7.1.2 Vorverarbeiten der Logikanalysatordaten

Die Logikanalysatoren sampeln abwechselnd Daten für eine Dauer von jeweils 4 s. In Wirklichkeit werden allerdings auch Daten kurz vor der startenden und kurz nach der beendenden Triggerflanke mitgesampelt. Durch den Trigger wird lediglich festgelegt, wo die Daten beginnen und enden. Ein Beispiel für solchermaßen gesampelte Daten zeigt Abbildung 7.3.

Im Modul `la.cpp` erfolgt die Vorverarbeitung des Datenpakets. Die Hauptaufgabe hierbei ist, die mit den Uhren der Logikanalysatoren gemessenen Zeitstempel der verschiedenen Pulse auf die durch das REF100-Signal gegebene Zeitbasis der Rubidiumuhr umzurechnen. Die Frequenz des REF100-Signals beträgt 100 Hz. Das

```
                    GPS-Evaluation Software Version vom 09.11.1995 (a)
Pulsliste von Sekunde
Quelle    Ablage (ns) Sekunde(s) 1(ms) 2(ms)   Spezifische Daten
Stellar
NavSymm
Motorola
Trimble
Magellan
Rockwell
Stellar10
NavSymm10

Fehlpulsliste von Sekunde
Quelle    Ablage (ns) Sekunde(s) 1(ms) 2(ms)   Spezifische Daten






                    PulsTail            PulsSent            PulsHead
Sekunde:            FehlTail            FehlSent            FehlHead
La      :             PC-Zeit:
Trigger:              Lost: 0  PC-Uhr:
Logikanalysator initialisieren
```

Abbildung 7.2: Bildschirm nach dem Start von GPSEVAL.EXE

heißt, daß alle 10 ms ein Zeitstempel der Rb-Uhr zur Verfügung steht. Durch (lineare) Interpolation zwischen solchen Zeitstempeln kann somit jeder beliebige Zeitstempelwert — dieser basiert ja auf den Logikanalysatoruhren — auf die Rb-Zeitskala umgerechnet werden. Genau das geschieht im Modul `la.cpp`. Durch den Trigger wird dabei der Nullpunkt des Datenpakets ermittelt. Dieser Nullpunkt wird durch die erste steigende Flanke des REF100-Signals nach dem Trigger definiert. Die Rb-Zeitskala wird somit an sich alleine durch das REF100-Signal festgelegt. Der Trigger dient lediglich zur Identifizierung der jeweils ersten Flanke.

Folgende Fehler in der Struktur des LA-Datenpakets werden erkannt:

- Es ist nicht die richtige Anzahl REF100-Flanken zwischen startendem und beendenden Trigger vorhanden. Bei 8 s Triggerperiode müssen dies exakt 400 sein.
- Der Trigger besitzt den falschen Signalpegel.
- Keine startende Triggerflanke in den LA-Daten
- Es existieren zu viele Triggerflanken.
- Keine beendende Triggerflanke konnte gefunden werden.

Tritt einer der obigen Strukturfehler auf, so werden die Daten als ungültig gekennzeichnet. Es stehen für den Meßzeitraum des betreffenden Datenpakets von den Logikanalysatoren keinerlei Daten über die Signale der Empfänger zur Verfügung. Solche Ausfälle traten übrigens in der vorliegenden Messung insgesamt 42 mal (dies entspricht 168 s) auf.

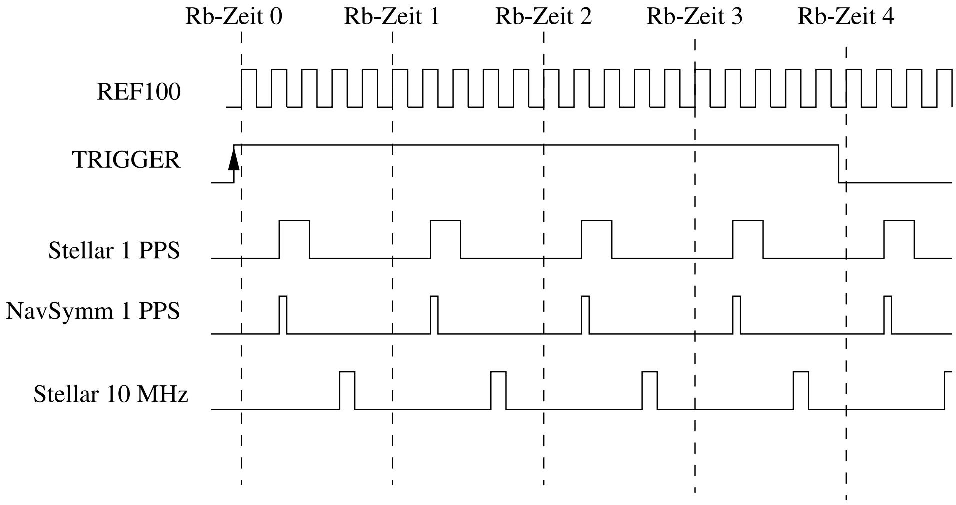

Abbildung 7.3: Die an den Logikanalysatoren analysierten Zeitsignale

Die fertig vorverarbeiteten Daten werden an den sogenannten *LA-FIFO* angehängt. Dieser dient der Zwischenspeicherung der fortlaufend anfallenden LA-Datenpakete.

7.1.3 Analysieren der GPS-Nachrichten

Im Modul `com.cpp` wird eine Klasse `Com` definiert, deren Methoden Befehle an die GPS-Empfänger sendet und Nachrichten von ihnen entgegennimmt und analysiert.

Bei der Initalisierung des Systems wird zunächst auch die Methode

```
int Com::Initialize(int Address,int Irq)
```

aufgerufen. Diese initialisiert einerseits die 8-fach serielle Schnittstellenkarte BB1008 und schickt andererseits Befehle zur Initalisierung an die Empfänger ab (siehe die Beschreibung der Empfänger im Hardware-Teil).

Nach der Initialisierung wird die Methode

```
int Com::Synchron()
```

immer wieder (in Polling-Manier) aufgerufen. Sie liest die RS232-Empfangsbuffer aus und analysiert (parst) sie nach Maßgabe der von den Empfängern geforderten Nachrichtenprotokollen. Die Informationen der Nachrichten werden an den *GPS-FIFO* des jeweiligen Kanals angehängt. Diese GPS-FIFOs sammeln die GPS-Nachrichten jedes Kanals. Sobald sie im Zuge der Kombination mit den Daten von den Logikanalysatoren benutzt wurden, werden sie wieder aus dem entsprechenden GPS-FIFO gelöscht.

7.1.4 Kombinieren von LA–Daten und GPS–Nachrichten

Liegen genügend Datenpakete von den Logikanalysatoren vor, so können diese mit den GPS-Daten kombiniert werden (siehe Abbildung 7.4). Das Ergebnis dieser Kombination ist eine *Pulsdatenliste*. Diese enthält für jede

Sekunde und jeden einem Empfängern zugeordneten Kanal die Ablage des betreffenden Pulses vom Sekundenbeginn der Rb-Zeit. Zustätzlich sind darin die Daten der GPS-Nachricht enthalten — sofern welche übermittelt wurden — und auch die Verzögerung, mit der die Nachricht nach dem Puls (über RS232) eingetroffen ist.

Hierbei wird einem Puls die jeweils erste GPS-Nachricht zugeordnet, die bis zu -10 ms vor dem Puls und noch 10 ms vor dem nächstfolgenden Puls eintrifft.

Treten an einem Kanal mehrere Pulse innerhalb einer Sekunde auf, so können natürlich nicht beide in der Pulsdatenliste untergebracht werden. Dies wird so gelöst, daß der erste Puls in die Pulsdatenliste gelangt und jeder nachfolgende Puls samt zugehöriger GPS-Nachrichten einen Eintrag in der *Fehlpulsliste* erzeugt.

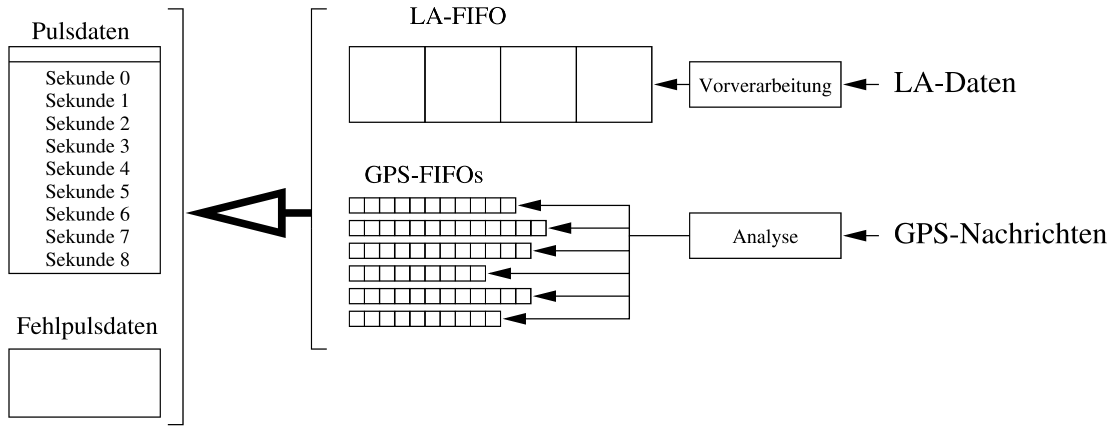

Abbildung 7.4: Kombinieren der Daten aus dem LA-FIFO und dem GPS-FIFO

7.1.5 Übertragung der Daten an den Kommunikations-PC

Die in Abschnitt 7.1.4 besprochene Pulsdatenliste liegt im lokalen Speicher. Da diese nur Pulsdaten für genau 2000 Sekunden fassen kann (wegen des Speichermangels im Real-Mode des PC), müssen die Daten rechtzeitig zum Server-PC übermittelt werden. Dasselbe gilt natürlich für die Fehlpulsliste mit einem Fassungsvermögen von 1000 Pulsen.

Deshalb werden die Pulsdaten jeder Sekunde in aufsteigender Reihenfolge an den Server-PC mit Prüfsumme und Header übermittelt. Die globale Variable `PulsSent` kennzeichnet dabei, welcher Puls zuletzt übermittelt wurde. Daraufhin wird auf eine Bestätigung vom Server-PC gewartet. Trifft diese ein, so weiß das Meßsystem, daß der Puls vom Server-PC übernommen und dort auf Festplatte gespeichert wurde. Deshalb kann der entsprechende Pulsdateneintrag aus der Liste gelöscht werden. Dieser Vorgang stoppt erst, wenn die Liste leer ist und startet von neuem, wenn wieder Pulse verfügbar sind.

Der momentane Umfang der Liste wird übrigens über die globalen Variablen `PulsTail` und `PulsHead` beschrieben. Sie geben an, zu welcher Sekunde der älteste (`PulsHead`) und der jüngste (`PulsTail-1`) Pulsdateneintrag in der Liste gemessen wurde. Diese Sekundenangabe erfolgt dabei relativ zur ersten vermessenen Sekunde. Diese hat natürlich den Sekundenwert 0.

Ein analoges Vorgang wird für die Übertragung der Fehlpulsliste verwendet. Die Variablen lautet dort allerdings `FehlSent`, `FehlTail` und `FehlHead`.

Ist die Übertragung zum Server-PC nicht möglich — weil etwa das Programm `SERVER.EXE` nicht läuft — wird die Pulsliste oder auch die Fehlpulsliste über kurz oder lang überlaufen. Sobald dies passiert, bricht das System mit der Meldung

```
Überlauf des Pulsfifos
```

beziehungsweise

```
Überlauf des Fehlpulsfifos
```

ab.

7.1.6 Low-Level-Treiber

Bedienung der 8-fach seriellen Schnittstelle BB1008 mit dem Modul `serialbo.cc`

Zur Benutzung der 8fach RS232-Karte BB1008 wurden die Klassen `boca` und `port` definiert. Zuerst muß der Konstruktor `boca()` aufgerufen werden. Daraufhin kann die Karte mit der Methode

```
int boca::Initialize(int CardAddress, int Irq)
```

initialisiert werden. Dabei muß die Hardwareadresse der Karte und die verwendete Interruptnummer angegeben werden. Ist das geschehen, so kann durch

```
port * boca::getPort(int Port, int Speed, int Parity, int Bits, int StopBits)
```

ein Port erzeugt werden. Dabei ist die Angabe einer Portnummer zwischen 1 und 8, die Angabe der Baudrate, die Anzahl der Paritybits, die Anzahl der Datenbits und die Anzahl der Stopbits erforderlich.

Durch die Methode

```
int port::GetFromReadBuf(char &c,unsigned long &t)
```

können dann die von einem Port eintreffenden Daten sukzessive gelesen werden. Das gelesene Zeichen wird dabei in der Variable `c` abgelegt. Die Variable `t` enthält den Zeitstempel des PC-Timers für den Zeitpunkt des Empfangens des Zeichens. Zum Aussenden von Daten über den Port ist die Methode

```
int port::PutToWriteBuf(char *string)
```

zu verwenden. Der zu sendende Text ist im nullterminierten String `string` enthalten.

Bedienung des Timers des Meß-PCs mit dem Modul `timer.cc`

Im Modul `timer.cc` ist eine Klasse `Tim`, definiert, die einen einfachen Zugriff auf den lokalen PC-Timer ermöglicht. Außerdem wurde zugleich eine Instanz `tim` dieser Klasse erzeugt.

Die Klasse muß mit dem Konstruktor `Tim()` initialisiert werden. Die Methode zur Ermittlung eines Zeitstempels des PC-Timers erfolgt durch die folgende Methode:

```
unsigned long Tim::getTime()
```

Die Auflösung des Zeitstempels beträgt 0,8381 μs. Durch die Beschränkung auf einen 32-Bitwert ist der Zeitstempel nur innerhalb einer Stunde eindeutig.

Beim Aufruf der obigen Methode dürfen die Hardwareinterrupts nicht gesperrt sein. Dementsprechend existiert eine Methode, die auch aufgerufen werden kann, wenn alle Hardwareinterrupts bereits gesperrt wurden:

```
unsigned long Tim::getTimed()
```

Zur einfachen Ermittlung der Zeitdifferenz zwischen zwei Zeitstempeln existiert die Methode

```
long Tim::diffTime(unsigned long t1, unsigned long t2)
```

Sie liefert — sofern die Ermittlung der Zeitstempel nicht mehr als 30 Minuten auseinanderlag — die vorzeichenrichtige Differenz zwischen zweitem Parameter und erstem Parameter `t2-t1`.

Die Methode

```
double Tim::transTime(unsigned long t)
```

liefert den in Sekunden umgerechneten Wert des Zeitstempels.

Bedienung des Bildschirm mit dem Modul `out.cc`

Zum einfachen Ansprechen des Bildschirms wurde eine Klasse `Out` mit einer bereits im Modul vorhandenen Instanz `out` definiert.

Nach der erforderlichen Initialisierung mit dem Konstruktor `Out()` kann auf einfache Weise auf den Bildschirm zugegriffen werden.

Mit der Methode

```
void Out::Clear()
```

wird der gesamte Bildschirm mit einem blauen Hintergrund gelöscht. Die Methode

```
void Out::Box(int x1, int y1, int x2, int y2, char typ)
```

gibt ein Rechteck mit den Eckkoordinaten `x1, y1, x2, y2` in der Farbe `typ` aus. Text kann mit den Methoden

```
void Out::PosText(int x, int y, const char *string, char typ)
void Out::PosTextB(int len,int x, int y, const char *string, char typ)
void Out::CenterText(int y, const char *string, char typ)
```

ausgegeben werden. Bei der ersten Form wird der Text an der Position `x,y` mit der Farbe `typ` ausgegeben. Der Text selbst wird durch einen nullterminierten String definiert. Die zweite Form unterscheidet sich, dadurch, daß das Ende des Strings nicht durch eine Nullzeichen, sondern die durch den zusätzlichen Parameter gegebene Länge `len` definiert ist. Die letzte Form schließlich gibt den nullterminierten String zentriert aus. Der Parameter `x` entfällt dann natürlich.

Bedienung der Logikanalysatoren mit dem Modul `hp16500.cpp`

Das Modul `hp16500.cpp` wurde vom Praktikanten Robert Klima am Institut für Computertechnik geschrieben. Es enthält die nowendigen Routinen zur Kommunikation mit den Logikanalysatoren. Zum Linken ist die Bibliothek `ieee488.lib` erforderlich.

Zur Initialisierung ist es erforderlich, die Routine

```
int HP_Init(struct HP_Initstrc * i_strc)
```

aufzurufen. Sie Initialisiert den Logikanalysator mit der lokalen Konfigurationsdatei. Danach können die Logikanalysatoren mit der Routine

```
int HP_Start(void)
```

gestartet werden. Über die Routine

```
int HP_GetData(struct HP_Info_Data **info)
```

kann dann im Pollingbetrieb überprüft werden, ob Daten vorliegen; im Erfolgsfalle werden diese Daten in einem Datenkonstrukt abgelegt. Diese Routine übernimmt alle notwendigen Schritte automatisch (etwa das Stoppen und Wiederstarten des gerade mit Daten vollgefüllten Logikanalysators).

Das Polling muß häufig genug geschehen, so daß früher gesampelte Daten nicht durch neue Daten überschrieben werden. Genaueres zu den Routinen ist in der Dokumentation des Modules zu finden. Diese ist im Anhang in Abschnitt C.1 abgedruckt.

7.1.7 Bedienung des Programms `GPSEVAL.EXE`

Nach dem Start des Programms erscheint der Bildschirm ähnlich wie in Abbildung 7.2. Dann folgen die zuvor bereits erwähnten Initialisierungen. Der Fortschritt derselben läßt sich am Eintrag in der letzten Zeile (am Bildschirm rot dargestellt) ablesen. Die hintereinander eingenommen Zustände sind folgende:

1. Initialiserung des Logikanalysators
2. Starten des Logikanalysators
3. RS232 und GPS-Empfänger initialisieren
4. Triggerinterrupt initialisieren
5. Warten auf 1. Trigger
6. Warten auf 2. Trigger
7. Warten auf 1. Datenpaket vom Logikanalysator
8. Normaler Betriebszustand

```
                     GPS-Evaluation Software Version vom 09.11.1995 (a)
Pulsliste von Sekunde 12
Quelle    Ablage (ns) Sekunde(s) 1(ms) 2(ms)    Spezifische Daten
Stellar    250984900  500000007      9   30 VFA
NavSymm    250985105  499999997    112  756 V   200 ns
Motorola   250984980  500000007     20          6 Sats  Status: 00
Trimble    250984769  499999997    234  256 V   010101010
Magellan   250985006  500000008     70      V   04
Rockwell   786943785  500000007     47      V
Stellar10  123488124  499999992
NavSymm10  344902232

Fehlpulsliste von Sekunde 12
Quelle    Ablage (ns) Sekunde(s) 1(ms) 2(ms)   Spezifische Daten






                    PulsTail        16  PulsSent        16  PulsHead       14
Sekunde:        24  FehlTail         0  FehlSent         0  FehlHead        0
La     :    0.822312  PC-Zeit: 2510.345566    2.004344
Trigger: 2508.341222  Lost: 0  PC-Uhr: 3.162E-6
normaler Betrieb
```

Abbildung 7.5: Bildschirm während des normalen Betriebs von `GPSEVAL.EXE`

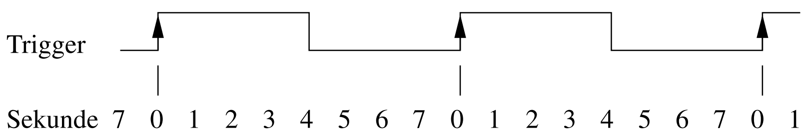

Abbildung 7.6: Sekundenwert innerhalb der Triggerperiode

Im gleichen Wortlaut wie in dieser Liste werden diese Zustände auch in der letzten Zeile dargestellt.

Im normalen Betrieb sieht der Bildschirm ähnlich wie in Abbildung 7.5 aus. Zunächst sei die Bedeutung der Zeilen im untersten Bildschirmbereich erklärt:

`PC-Zeit`: Dieses Feld kennzeichnet den momentanen — nur innerhalb einer Stunde eindeutigen — Zeitstempelwert der lokalen PC-Uhr (Dieser Wert umfaßt einen Wertebereich von 0 bis 3600 Sekunden). Zieht man davon den Zeitstempelwert des letzten Triggers ab (s.u.), so erhält man den Sekundenwert innerhalb der 8 s dauernden Triggerperiode, siehe Abbildung 7.6.

`PC-Uhr`: Hier wird der Frequenzfehler der PC-Uhr angezeigt. Diese kann aus den PC-Zeitstempelwerten des Triggerinterrupts ermittelt werden, wobei über mehrere Triggerperioden gemittelt wird.

`Trigger`: Hier steht der PC-Zeitstempel des letzten Triggers. Falls der Triggerimpuls keinen Interuppt auslösen sollte, so wird künstlich anhand der Kenntnis des PC-Uhrenfrequenzfehlers ein Zeitstempel für den Trigger erzeugt (8 Sekundenperiode). Solche Triggerverluste werden gleich daneben unter dem Titel `Lost` dargestellt.

Dabei wird die Anzahl der hintereinander verlorengegangenen Triggerverluste angezeigt. Tritt der Trigger wieder auf, so wird der Wert wieder auf 0 zurückgesetzt. Treten zuviele Triggerverluste (mehr als 11) auf, so wird das Programm fatal beendet.

`Lost`: siehe `Trigger`

`Sekunde`: Dieser Wert gibt die momentane Meß-Sekunde an, in der sich das Meßsystem befindet. Der Wert wird allerdings nur alle 4 s aktualisiert. Dies deshalb, weil die Datenpakete von den Logikanalysatoren 4 Sekunden umfassen.

`La`: Hier steht der Sekundenwert innerhalb der 8 Sekunde-Triggerperiode, zu dem das La-Datenpaket vollständig übermittelt wurde. In einer Triggerperiode (8 s) werden immer zwei solche Werte angezeigt. Der erste ungefähr zum Sekundenwert 1 und der andere zum Sekundenwert 5. Das heißt, daß ungefähr eine Sekunde verstreicht, bist ein LA-Datenpaket nach Triggerende über den HPIB-Bus eintrifft.

`PulsTail, PulsSent, PulsHead`: siehe Abschnitt 7.1.5

`FehlTail, FehlSent, FehlHead`: siehe Abschnitt 7.1.5

Nun sollen die Pulsdaten einer Sekunde selbst beschrieben werden: Unter dem Titel `Pulsliste von Sekunde xx` für `xx` wird hier die momentan dargestellte Sekunde angezeigt. Darunter befindet sich eine Beschreibungszeile zur Kennzeichnung der Tabellenspalten.

Die erste Spalte (Quelle) enthält den Namen des GPS-Empfängers von dem der 1 PPS-Puls geliefert wird. Ausnahmen sind `Stellar10` und `NavSymm10`, die für die auf 1 Hz heruntergeteilten 10 MHz-Signale stehen.

Die zweite Spalte (Ablage(ns)) gibt die Ablage des betreffenden Pulses von der durch die Referenzuhr definierten Zeitbasis an. Der Wert kann übrigens auch kleine negative Werte annehmen (aber größer als -10 ms).

Die dritte Spalte (Sekundes(s)) enthält den Sekundenanteil des Pulses. Dieser wird periodisch als Nachricht über RS232 geliefert. Dieser Sekundenwert ist definiert als Anzahl der Sekunden die seit dem 6.Januar 1980 in der betreffenden Zeitskala (GPS oder UTC) verstrichen sind.

Die vierte und fünfte Spalte (1(ms), 2(ms)) geben die Verzögerungswerte der ersten und zweiten Nachrichten (in Millisekunden) relativ zu dem zuvor aufgetretenen Puls an. Welche der beiden Nachrichten welche Information trägt, ist vom Empfänger abhängig.

Zwischen fünfter und sechster Spalte befinden sich Flags, die die Gültigkeit der Signale (durch `V` dargestellt), Antennenfehler (`A`) oder Empfängerfehler (`F`) anzeigen. Zu beachten ist, daß für den Motorola-Empfänger das `V`-Flag nicht benutzt wird.

7.2 Programm `SERVER.EXE`

Das Programm `SERVER.EXE` läuft auf dem Server-PC. Es dient dazu, die über die RS232-Schnittstelle vom Meß-PC einlangenden Puls– und Fehlpulsdaten auf die lokale Festplatte zu speichern. Um die Daten dann manuell über ftp ins TU-Net übertragen zu können, muß das Programm jederzeit abbgebrochen und wieder gestartet werden können. Die Module des Programms sind in Tabelle 7.2 aufgelistet.

Die einlangenden Daten werden unmittelbar auf die Festplatte geschrieben. Dabei werden Pulsdaten über jeweils 3600 Sekunden zu einer Datei zusammengefaßt. Der Name einer solchen Datei ist `Gxxxxxxx.PLS`. Die `x` stehen hierbei für eine siebenstellige Dezimalzahl mit führenden Nullen. Sie gibt an, um die wievielte Datei es sich handlet (beginnend mit 0). Die erste Datei lautet also `G0000000.PLS`.

Fehlpulse werden direkt nacheinander in die Datei `FEHLPULS.PLS` geschrieben.

Modul	Zweck
server.cpp	Hauptprogramm
timer.cpp	Treibermodul zur Bedienung des PC-Timers
serialbo.cpp	Treibermodul zur Bedienung der 8-fach seriellen Schnittstellenkarte
out.cpp	Treibermodul zur einfachen Ausgabe am Textbildschirm des PC

Tabelle 7.2: Module des Programms SERVER.EXE

Die Bedienung des Programms ist relativ simpel: Beim Starten des Programm synchronisiert es sich zunächst mit GPSEVAL.EXE auf dem Meß-PC. Dabei ermittelt das Programm den zuletzt geschriebenen Puls und ist damit in der Lage, dem in Abschnitt 7.1.5 beschriebenen Protokoll zu folgen. Durch Drücken der Escape-Taste wird das Programm einfach abgebrochen.

8 Format der Meßdaten

Während der Langzeitmessung über 64 Tage und 17 Stunden vom 12. Dezember 1995 11:51:07 UTC bis zum 15. Februar 1996 04:51:16 UTC wurden die Pulsdaten der 6 Empfänger für jede Sekunde aufgezeichnet. Die Meßzeit betrug somit 5,590.800 Sekunden.

Allerdings mißlang die Übertragung der Daten von den Logikanalysatoren während dieser Zeit in 42 Fällen (zu je 4 Sekunden). Über 168 Sekunden liegen daher keine Daten vor. Effektiv liegen damit Daten über 5,590.632 Sekunden vor. Für einfache statistische Auswertung spielen diese fehlenden Daten keine Rolle. (Lediglich 0,003 % der Daten fehlen.)

Die Daten wurden als Dateien gespeichert, die Pulsdaten für jeweils eine Stunde enthalten. Es liegen damit 1553 Dateien mit einer Länge von je 525.600 Bytes vor; insgesamt somit 778 MByte. Die Namen der Dateien laufen von `G0000000.PLS` bis `G0001552.PLS`. Die Zahl gibt dabei die Anzahl der seit Meßbeginn verstrichenen Stunden an. Diese Dateien wurden zur Archivierung in komprimierter Form (218 MB) auf einer beschreibaren CD untergebracht.

Eine Datei `G000xxxx.PLS` besteht aus 3600 Datensätzen, die Daten (von 8 Kanälen) zu jeweils einer Sekunde umfassen. Der Aufbau eines solchen Datensatzes ist in Tabelle 8.1 dargestellt. Der erste Feldwert eines solchen Datensatzes ist der Gültigkeitswert. Ist dieser gleich Null, so liegen gültige Daten vor, ansonsten waren die Daten defekt oder wurden nicht übertragen (wie oben erwähnt kam dies für 168 solcher Datensätze vor).

Feld	Länge/Bytes
Gültigkeitswert	2
1 PPS-Pulsdaten des Stellar-Empfängers	18
1 PPS-Pulsdaten des NavSymm-Empfängers	18
1 PPS-Pulsdaten des Motorola-Empfängers	18
1 PPS-Pulsdaten des Trimble-Empfängers	18
1 PPS-Pulsdaten des Magellan-Empfängers	18
Time-Mark-Pulsdaten des Rockwell-Empfängers	18
10 MHz-Pulsdaten des Stellar-Empfängers	18
10 MHz-Pulsdaten des NavSymm-Empfängers	18

Tabelle 8.1: Aufbau eines Datensatzes zu einer Sekunde

Die Pulsdaten der einzelnen Kanäle haben etwas unterschiedliche Bedeutung. Die allgemeine Form der Pulsdaten besitzt die in Tabelle 8.2 beschriebene Form (Zahlen, die mehrere Bytes umfassen, sind im Intelformat gespeichert, bei dem das niederwertigste Byte zuerst angeordnet ist). Das mit *Ablage* bezeichnete Feld kennzeichnet die Verzögerung des Pulses des betreffenden Kanals von der durch das REF100-Signal definierten

Feld	Länge/Bytes	Typ
Flags	2	16 Bit, Bitfeld
Ablage (in ns)	4	32 Bit (signed)
Sekundenanteil	4	32 Bit (unsigned)
Verzögerung Nachricht 1 (in ms)	2	16 Bit (signed)
Verzögerung Nachricht 2 (in ms)	2	16 Bit (signed)
Freier Datenwert	4	32 Bit

Tabelle 8.2: Pulsdaten für einen Kanal

Zeitskala (Rb-Zeit). Sie wird in Einheiten von Nanosekunden angegeben. Die Felder *Verzögerung von Nachricht x* kennzeichnen hingegen die Verzögerung der Nachricht vom aufgetretenen Puls in der Einheit Millisekunden.

Der *Sekundenanteil* enthält den Zeitwert für die Flanke des zugehörigen Pulses. Er wird durch die seit der GPS-Startepoche (6. Jänner 1980) in der betrachteten Zeitskala (GPS oder UTC) vergangenen Sekunden definiert. Dieser Wert wurde aus den Zeitnachrichten der Empfänger berechnet. Da der NavSymm den Sekundenanteil relativ zu UTC angibt, ist dort der Sekundenanteil um die zwischen GPS und UTC bestehende Zahl von Schaltsekunden geringer.

Ist einer der bis jetzt besprochenen Werte nicht verfügbar, weil etwa kein Puls aufgetreten ist (keine Ablage) oder die entsprechende Zeitnachricht nicht kam (kein Sekundenanteil, kein Verzögerungswert), so enthalten die Felder definierte ungültige Werte. Diese sind für 16-Bit Zahlen der Wert `0x8000` und für 32-Bit Zahlen der Wert `0x80000000`.

Die untersten 8 Bits des *Flags*-Feld haben für die 1 PPS und den Time-Mark-Puls eine einheitliche Bedeutung, die in Tabelle 8 wiedergegeben ist. Die obersten 8 Bits sind normalerweise alle Null, sie werden nur beim Magellan benutzt.

Bitnummer	Bedeutung
0	Gibt an, ob überhaupt ein Puls aufgetreten ist.
1	Gibt an, ob die GPS-Daten oder Pulse gültig sind; Wird je nach Empfänger durch unterschiedliche Kriterien entschieden
2	Gibt an, ob ein interner Empfängerfehler erkannt wurde.
3	Gibt an, ob ein Fehler an der Antenne erkannt wurde
4	Gibt an, ob die erste Nachricht fehlerhaft war (falsche Prüfsumme).
5	Gibt an, ob die zweite Nachricht fehlerhaft war (falsche Prüfsumme).
6,7	*unbenutzt*
8	Gibt an, ob auf diesem Kanal während der Aufzeichnungszeit des Logikanalysators für diese Sekunde unsinnig viele Pulse (>= 20) auftraten („Jumpy Channel“)

Tabelle 8.3: Bedeutung der Bits des Flags-Felds der Pulsdaten eines Kanals

Die spezifischen Unterschiede der Pulsdaten für die einzelnen Kanäle sind in den folgenden Abschnitten wiedergegeben:

8.1 Stellar 1 PPS und Stellar 10 MHz

Beim Stellar sind zwei Nachrichten für die Generierung der Pulsdaten verwendet worden (siehe Abschnitt 6.2.2):

Die erste Nachricht ist die STATUS-Nachricht. Aus ihr ist zu entnehmen, ob ein Antennen– oder Empfängerfehler aufgetreten ist, oder ob die Zeitausgänge gültig sind (DATA VALID). Diese Informationen wurden alle in die entsprechenden Bits des Flag-Feldes eingetragen. Die Nachricht enthält weiters die Informationen zur Berechnung des Sekundenanteils des Pulses.

In der zweiten Nachricht TAF stehen interessante Informationen über den Zeit– und Frequenzausgang. Da der Platz in den Pulsdaten für den 1 PPS-Puls des Stellars nicht ausreichte, wurden diese zum Teil in die ohnehin unbenutzten Felder für die Pulsdaten des 10 MHz-Pulses des Stellars untergebracht. Insbesondere besitzt die Nachricht einen eigenen Zeitstempelwert, der den Zeitpunkt der Ermittlung der Daten in TAF angibt. Er ist im Sekundenanteil des 10 MHz-Pulses untergebracht.

Da beide Nachrichten mit Prüfsummen versehen sind, wurde die Information, ob diese korrekt sind, in den entsprechenden Bits des Flag-Feldes eingetragen.

Wegen der von der Standardform stark abweichenden Struktur der Pulsdaten des Stellar seien diese nochmals dargestellt. In Tabelle 8.4 sind die Pulsdaten für den 1 PPS-Puls und in Tabelle 8.5 die des 10 MHz-Pulses des Stellars dargestellt. Die genaue Bedeutung der Felder aus der TAF-Nachricht sei dem Abschnitt 6.2.2 entnommen.

Feld	Länge/Bytes	Typ
Flags	2	16 Bit, Bitfeld
Ablage 1 PPS-Puls (in ns)	4	32 Bit (signed)
Sekundenanteil	4	32 Bit (unsigned)
Verzögerung Nachricht 1 (in ms)	2	16 Bit (signed)
Verzögerung Nachricht 2 (in ms)	2	16 Bit (signed)
Time Offset (TAF-Nachricht)	4	32 Bit

Tabelle 8.4: 1 PPS-Pulsdaten des Stellar

Feld	Länge/Bytes	Typ
Flags	2	16 Bit
Ablage 10 MHz-Puls (in ns)	4	32 Bit (signed)
Sekundenanteil der TAF-Nachricht	4	32 Bit (unsigned)
Untere 16 Bit des TU-Felds der TAF-Nachricht	2	16 Bit
Obere 16 Bit des TU-Felds der TAF-Nachricht	2	16 Bit
Frequenz Offset des 10 MHz-Pulses (TAF-Nachricht)	4	32 Bit

Tabelle 8.5: 10 MHz-Pulsdaten des Stellar

8.2 NavSymm 1 PPS und NavSymm 10 MHz

Beim NavSymm werden zwei Nachrichten benutzt: Die erste ist die GPS-Nachricht. Aus ihr wird der Sekundenanteil berechnet (speziell aus den Feldern UTC-TIME und UTC-DATE der GPS-Nachricht). Da der NavSymm nur Zeitwerte nach der UTC-Zeitskala ausgibt, ist der Sekundenanteil gegenüber denen der anderen Empfänger um die entsprechende Anzahl Schaltsekunden geringer (10 s vor dem 01.01.1996 und 11 s danach).

Die zweite Nachricht SFM enthält die interessante Information, mit welcher Genauigkeit des 1 PPS-Pulses zu rechnen ist. Der verwendete Wert ACC-TIME gibt den erwarteten 2σ Wert für die Abweichung in Nanosekunden an. Dieser Wert wird als freier Datenwert der 1 PPS-Pulsdaten des NavSymm eingetragen. Ist dieser Wert Null, so ist die Genauigkeit laut Handbuch des NavSymm inakzeptabel. Diese Kriterium wird benutzt, um ungültige Daten anzuzeigen (im entsprechende Bit des Flags-Feldes).

Die Pulsdaten des 10 MHz-Ausgang des NavSymm enthalten nur die Ablage des Pulses, alle anderen Felder bleiben unbenutzt.

8.3 Motorola 1 PPS

Beim Motorola–Empfänger wird nur eine Nachricht (`Position-Status-Data`) benutzt. Aus ihr kann der Sekundenanteil berechnet werden. Da kein einfaches Kriterium zur Beurteilung der Gültigkeit des Pulses oder der Daten existiert, wurden lediglich wichtige Statusdaten im freien Datenwert der Pulsdaten untergebracht. Die Bits 0–7 des freien Datenwerts enthalten den Status des Empfängers (siehe Abschnitt 6.2.4). Die Bits 8–15 enthalten die Anzahl der getrackten Satelliten (0–6).

8.4 Trimble 1 PPS

Die erste Nachricht des Trimble ist die `TM`-Nachricht. Sie enthält Informationen zur Berechnung des Sekundenanteils (Feld *AA, BB, CC.DDD, EE, FF* und *GGGG* der Nachricht `TM`, siehe Abschnitt 6.2.5). Genauso wie beim NavSymm ist dieser Sekundenanteil an sich auf die UTC-Zeitskala bezogen. Da in der Nachricht auch die Schaltsekundendifferenz zwischen UTC und GPST enthalten ist, liegt auch der Sekundenanteil bezüglich GPST vor. Weiters enthält sie interessante Informationen, die im freien Datenwert eingetragen wird. Wir der freie Datenwert dezimal angeschrieben, so beschreiben die zwei Ziffern mit niedrigstem Stellenwert die Anzahl von Schaltsekunden zwischen GPST und UTC, die Ziffern drei und vier geben die Anzahl verwendbarer Satelliten an und die fünfte Ziffer gibt das Feld *K* der Nachricht `TM` an (siehe Abschnitt 6.2.5). Sie hat den Zahlenwert 1, im Falle daß die Informationen zur Berechnung des Sekundenanteils gültig sind, ansonsten ist sie 0.

Die zweite Nachricht `ST` wird nur verwendet, um das Gültigkeitsflag zu setzen. Das dazu benutzte Feld *AA* ist genau dann 00, falls der Empfänger Positionsbestimmungen durchführt, genau dann ist nämlich auch das Zeitsignal gültig.

8.5 Magellan 1 PPS

Der Magellan übermittelt die Nachricht `K3`, siehe Abschnitt 6.2.6. In ihr ist neben den Informationen zur Bestimmung des Sekundenanteils, auch ein Bytewert vorhanden, der angibt, ob der Empfänger im normalen Betriebsmodus oder im *coasting*-Betrieb ist. Im *coasting*-Betrieb trackt der Empfänger zuwenig Satelliten, um einen korrekten 1 PPS-Puls auszugeben zu können. Dies wird benutzt, um das Gültigkeitsflag zu setzen. Eine andere interessante Information ist der TFOM-Wert (Time Figure of Merit, siehe Abschnitt 6.2.6). Dieser Wert wird in den obersten 8 Bits des flags-Feldes eingetragen.

8.6 Rockwell 1 PPS

Die einzige beim Rockwell verwendete Nachricht ist die *Position, Velocity, Time Solutions and Status*-Nachricht. Sie enthält zwei interessante Werte; den genauen GPS-Zeitwert des Time-Mark-Pulses — dieser Puls ist mit keiner Zeitskala synchron und wird direkt vom Quarz der Empfängerplatine abgeleitet — mit 1 μs Auflösung (GPS Time of Week und GPS Week Number) und ein Flag zur Kennzeichnung der Gültigkeit (UTC-Validity-Flag), siehe Abschnitt 6.2.7. Der ganzzahlige Anteil des GPS-Zeitwerts wird als Sekundenanteil in die Pulsdaten, der Restbetrag in Einheiten von Nanosekunden in den freien Datenwert eingetragen. Das Gültigkeitsflag der Pulsdaten wurde durch das UTC-Validity-Flag bestimmt.

Mit dem genauen GPS-Zeitwert des Time-Mark-Pulses kann der jeweilige Sekundenbeginn der GPS-Zeitskala rekonstruiert werden. Wenn im folgenden daher von der 1 PPS-Ablage des Rockwell die Rede ist, so ist damit die Ablage des mit Hilfe der Nachricht rekonstruierten Sekundenbeginns der Zeitskala die Rede.

8.7 Fehlpulsdaten

Etwaige Fehlpulse der Empfänger — das sind Pulse die innerhalb einer Sekunde mehrfach auftreten — werden in der Fehlpulsdatei `FEHLPULS.PLS` gespeichert. Sie enthält in chronolgischer Reihenfolge die aufgetretenen Fehlpulse. Das Format ist in Tabelle 8.6 dargestellt.

Feld	Länge/Bytes	Typ
Kanalnummer	2	16 Bit (signed)
Sekundenummer	4	32 Bit (unsigned)
Sekundenanteil	4	32 Bit (unsigned)
Pulsdaten	18	siehe Tabelle 8.2

Tabelle 8.6: Datensatz der Fehlpulsdatei

Die *Kanalnummer* kennzeichnet den Kanal (0=Stellar 1 PPS, 1=NavSymm 1 PPS, 2=Motorola 1 PPS, 3=Trimble 1 PPS, 4=Magellan 1 PPS, 5=Rockwell 1 PPS, 6=Stellar 10 MHz, 7=NavSymm 10 MHz), in dem der Fehlpuls aufgetreten ist. Die *Sekundenummer* gibt Aufschluß darüber, zu welcher Sekunde seit Meßbeginn — beginnend mit 0 — das Fehlpulsereignis stattgefunden hat. Die restlichen Daten gestalten sich genauso wie bei einem gewöhnlichen Puls.

9 Auswertung

9.1 Berechnung der Referenzzeit

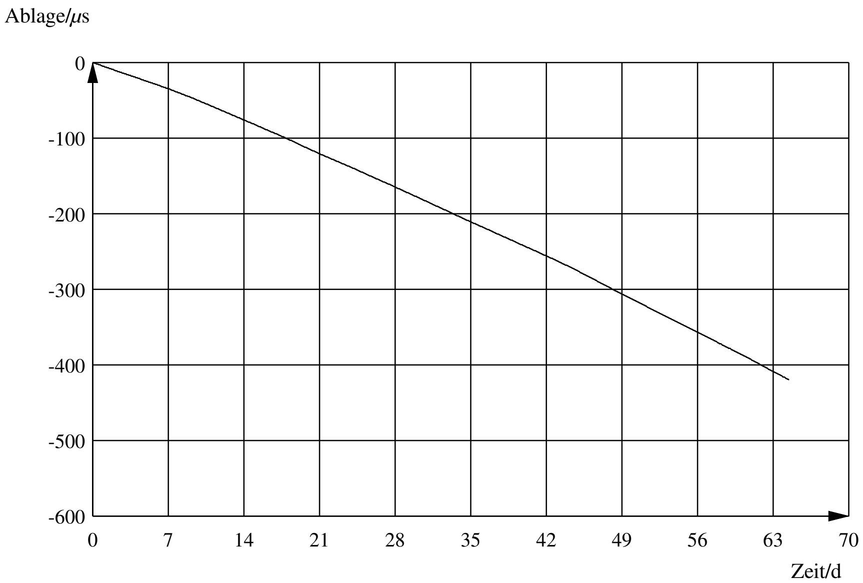

Abbildung 9.1: Ablage des 1 PPS-Pulses des Stellarempfängers

Um das Verhalten der Empfänger beurteilen zu können, ist es erforderlich, die zugrundeliegende Zeitskala GPS-Time möglichst genau zu reproduzieren. Zu diesem Zweck soll der beste Empfänger (Stellar GPS Model 100A) benutzt und dessen Zeitsignale mit einem Rubidium-Frequenznormal (FRS-C von Ball Efratom) gemittelt werden. Damit sollen die durch stochastische Fehlereinflüsse (z.B. SA) erzeugten Schwankungen möglichst eliminert werden.

Um sich ein Bild vom Verhalten des gegen die Rubidiumuhr vermessenen 1 PPS-Pulses des Stellarempfängers zu machen, ist die Ablage des Pulses über die gesamte Meßdauer in Abbildung 9.1 dargestellt. Ein positiver Wert der Ablage bedeutet eine Verspätung gegenüber der durch die Rubidiumuhr definierten Zeit.

Wie aus der Abbildung zu erkennen ist, zeigt die Ablage annähernd lineares Verhalten. Die Ursache ist der Frequenzfehler der Rubidiumuhr. Da die Drift der Rubidiumuhr relativ gering ist, ist sie aus dem Graphen nicht auszumachen. Versucht man allerdings eine quadratische Funktion an den Graphen anzupassen, indem man drei Stützwerte (einer am Beginn $t = 0$ d, einer in der Mitte $t = 32$ d und einen am Ende $t = 64$ d der Meßreihe) als Grundlage für die Interpolation der quadratischen Funktion verwendet, so erhält man folgende Werte für Frequenzfehler und Drift der Rubidiumuhr:

$$y_0 = -6{,}3 \cdot 10^{-11} \quad D = 2{,}2 \cdot 10^{-18} \mathrm{s}^{-1}. \tag{9.1}$$

Das statistische Verhalten der Ablage läßt sich aus der Allan-Varianz ablesen. Sie ist in Abbildung 9.2 dargestellt.

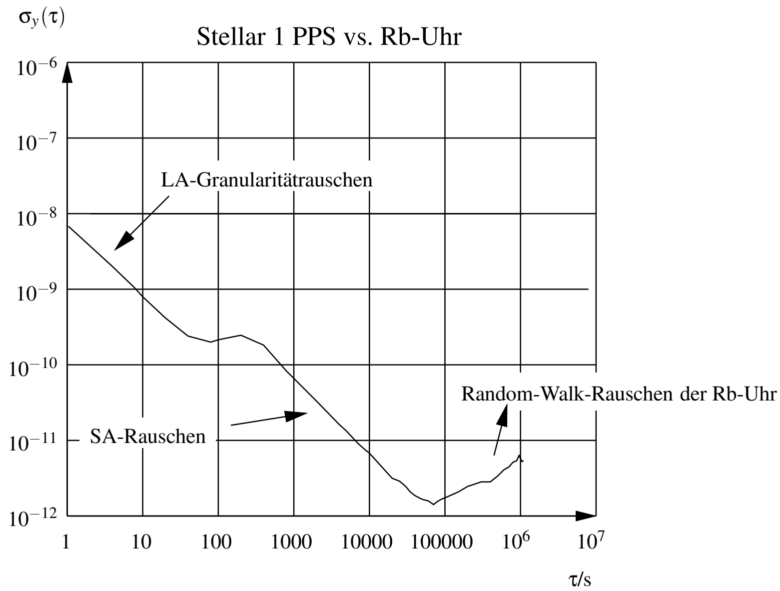

Abbildung 9.2: Allanvarianz des 1 PPS-Pulses des Stellarempfängers

Es lassen sich dabei drei wesentliche Rauschbeiträge ablesen: Für sehr kurze Mittelungszeiten (bis 100 s) erzeugt die Granularität der Logikanalysatoren eine Art Quantisierungsrauschen (± 16 ns). Wie für weißes Phasenrauschen typisch fällt dieses mit τ^{-1} ab. Für Mittelungszeiten von 100 s bis 70.000 s herrscht das durch stochastische Schwankungen des 1 PPS-Pulses (vor allem SA-Effekte) erzeugte Rauschen vor, das es auszumitteln gilt. Ab 70.000 s dominiert das für Rb-Uhren typische Random-Walk-Rauschen (siehe auch Lesage und Audoin [1979]) mit seinem $\tau^{1/2}$-Verhalten.

Drifteffekte der Rubidiumuhr müssen nicht berücksichtigt werden, da diese für die interessierenden Mittelungszeiten $\tau < 10^6$ s gegenüber dem Rauschen der Rubidiumuhr vernachlässigbar sind: Die Drift erscheint in der Allan-Varianz in Form einer mit τ^1 ansteigenden Geraden. Für die Mittelungszeit $\tau = 10^6$ s betrüge die durch die Drift hervorgerufene Allan-Varianz (siehe Allan [1987])

$$\sigma_y(\tau) = \frac{D\tau}{\sqrt{2}} = \frac{2{,}2 \cdot 10^{-18} \times 10^6}{\sqrt{2}} = 1{,}5 \cdot 10^{-12}. \tag{9.2}$$

Dieser Beitrag geht in dem dort herrschenden Random-Walk-Rauschen unter.

Mit dem Diagramm kann man die Allan-Varianz der Rubidiumuhr nach oben abschätzen. Laut Spezifikation beträgt die Allan-Varianz für 1 Sekunde $\sigma(1\ \mathrm{s}) = 10^{-10}$. Bis zum Mittelungswert $\tau = 100$ s fällt dieser Wert nach einem $\tau^{-1/2}$-Gesetz ab (siehe Tabelle 6.1). Berücksichtigt man das Random-Walk-Rauschen für sehr lange

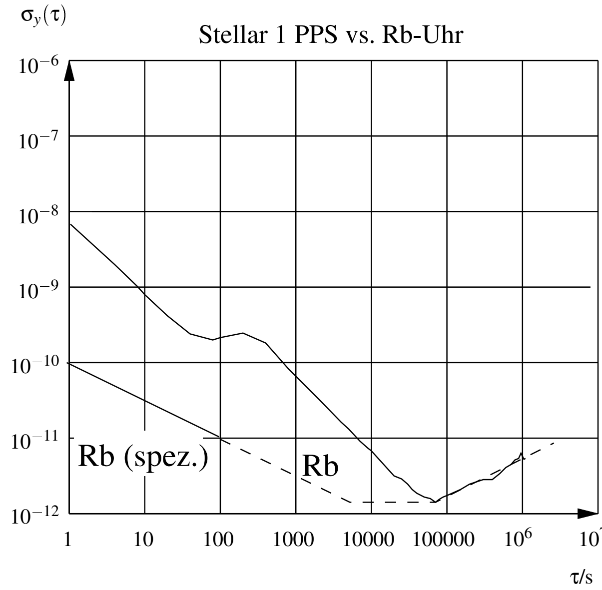

Abbildung 9.3: Abschätzung der Allanvarianz der Rb-Uhr

Mittelungszeiten und geht davon aus, daß das Rauschen der Rubidiumuhr durch Rauscharten mit einfachen Potenzgesetzen beschrieben werden kann (siehe Abschnitt 2.2.4 und Artikel Lesage und Audoin [1979]), so stellt sich die Allan-Varianz der Rubidiumuhr im schlechtesten Falle so wie in der Abbildung 9.3 gezeigt dar.

Um den Rechenaufwand für die Mittelung bewältigen zu können, ist ein einfaches Verfahren erforderlich. Das hier zum Einsatz kommende besteht aus einer arithmetischen Mittelung aller Ablagen T Sekunden vor und nach dem betrachteten Zeitpunkt. Dieses Fenster umfaßt $2T+1$ Werten mit dem zu ermittelnden Bezugszeitpunkt in der Mitte des Fensters und wird nun Schritt für Schritt über das Datenmaterial geschoben. Diese gemittelten Werte dienen dann als verbesserte Referenzzeit. Formal läßt sich die aus der Ablage des 1 PPS-Pulses des Stellars (t kennzeichnet den Zeitpunkt eines Ereignisses)

$$x_{\text{Stellar}}(t) = t(\text{1 PPS-Puls des Stellars}) - t(\text{1 PPS-Puls der Rb-Uhr}) \tag{9.3}$$

gewonnene Ablage der Referenzzeit

$$x_{\text{Ref}}(t) = t(\text{gemittelter 1 PPS-Puls des Stellars}) - t(\text{1 PPS-Puls der Rb-Uhr}) \tag{9.4}$$

so anschreiben:

$$x_{\text{Ref}}(t) = \frac{1}{2T+1} \sum_{i=-T}^{T} x_{\text{Stellar}}(t+i). \tag{9.5}$$

Der Parameter T ist so zu wählen, daß das Mittelungsfenster einerseits die SA-Effekte möglichst eliminiert, andererseits aber starke Einflüsse vom Rauschens des Rubidiumoszillators vermeidet. Es is also ein Kompromiß zwischen großem (möglichst umfangreiche Vernichtung des SA-Rauschens) und kleinem Mittelungsfenster (möglichst geringer Einfluß des langfristig sehr großen Rauschens der Rb-Uhr) zu finden. Für eine exakte Behandlung wäre die exakte Kenntnis des statistischen Verhaltens der Rb-Uhr un der SA-Effekte vonnöten. Da diese nicht vorliegen, muß auf anderem Wege ein günstiger Parameter T ermittelt werden. Zu diesem Zweck betrachtet man den nach der Mittelungszeit τ erhaltenen Zeitfehler (1σ) durch das Rauschen der Rb-Uhr: Im Bereich von 5.000 s und 70.000 s (Flicker-Frequenz-Rauschen, siehe Allan-Varianz der Rb-Uhr in Abbildung 9.3) ergibt sich dieser zu (vergleiche Seite 22):

$$\sigma_{\Delta x}(\tau) = 1,2\tau \cdot \sigma(\tau). \tag{9.6}$$

Die Allan-Varianz ist dort konstant $\sigma_y(t) = 1,4 \cdot 10^{-12}$. Die folgende Tabelle gibt die Zeitfehler für verschiedene Werte von τ an:

τ	$\sigma_{\Delta x}(\tau)$	4σ
5.000 s	9 ns	36 ns
10.000 s	18 ns	72 ns
15.000 s	27 ns	108 ns
20.000 s	36 ns	144 ns
30.000 s	54 ns	216 ns
40.000 s	72 ns	288 ns

Als günstiger Wert erscheint $t = 20.000$ s. Der Fehler durch die Uhr ist einerseits noch klein ($\approx$144 ns für $99,99$ % Vertrauensintervall), andererseits kann der größte Teil der SA-Fehler eliminiert werden, da die Fenstergröße bereits 40.000 ns beträgt. Schätzt man noch die zurückbleibenden SA-Fehler großzügig ab, so erhält man etwa Δx=200 ns.

Die im folgenden beschriebenen Ablagen sind, wenn nicht anders erwähnt auf diese durch Mittelung gewonnene Referenzzeit bezogen.

9.2 Übersicht über die Ablagen der Empfänger

Im Anhang sind zur Übersicht über das Verhalten der GPS-Empfänger die Ablagen der Empfänger einmal für einen Ausschnitt von 1000 s und einmal über die gesamte Meßdauer dargestellt. Der Ausschnitt über die gesamte Meßdauer ist natürlich mit Vorsicht zu genießen, da zwischen zwei benachbarten Bildpunkten dieses Graphen etwa 5.500 Pulse unberücksichtigt bleiben. Trotzdem stellt er eine gut Übersicht über das Verhalten der Empfänger dar. Aus den 1000 s–Graphen ist deutlich zu erkennen, daß der Stellarempfänger über kurze Zeiträume ein wesentlich geringeres Schwankungsverhalten als alle anderen Empfänger zeigt. Der Grund dafür liegt in der hohen Qualität des eingebauten Quarzoszillators (ein OXCO).

Abbildung A.1 auf Seite 125 zeigt die Ablagen der 1 PPS-Pulse des Stellar, NavSymm und Motorola für einen Bereich von 1000 s.

Die Abbildungen A.2 bis A.4 auf den Seiten 125 bis 126 zeigen die entsprechenden Ablagen für Trimble, Magellan und Rockwell.

Die über die gesamte Meßdauer erstreckten Übersichtsgraphen seien den Abbildungen A.5 bis A.10 (Seiten 127 bis 129) entnommen.

9.3 Verzögerung der GPS-Nachrichten

In Tabelle 9.1 sind die Kennwerte für die Verzögerung der Nachrichten gegenüber dem zugehörigen Puls wiedergegeben. Die Spalten *Min* und *Max* enthalten die minimal bzw. maximal aufgetretenen Verzögerungen in

Nachricht	Min. (ms)	Max. (ms)	Mittel (ms)	Std.abweichung (ms)
Stellar `STATUS`	6	79	9	14
Stellar `TAF`	86	165	95	144
NavSymm `GPS`	41	192	177	177
NavSymm `SFM`	292	859	443	449
Motorola	11	26	21	21
Trimble `TM`	-10	989	271	312
Trimble `ST`	-10	989	227	278
Magellan	1	438	103	104
Rockwell	39	678	73	73

Tabelle 9.1: Kennwerte für die Verzögerung der Nachrichten

Millisekunden. Die Spalte Feld *Mittel* beschreibt dem Mittelwert der Verzögerung und in Spalte *Std.abweichung* ist die Standardabweichung der Verzögerung enthalten.

Bis auf den Trimble besitzen alle Empfänger eingrenzbare Intervalle innerhalb derer die Nachrichten auftreten. Beim Trimble trifft die Nachricht zu beliebigen Zeitpunkten ein. Mit Hilfe der mitgelieferten Sekundenbruchteile konnte aber durch Runden der korrekte Sekundenanteil ermittelt werden.

9.4 Fehlverhalten der Empfänger

Es zeigt sich, daß die Empfänger von NavSymm und Magellan ein relativ ausgeprägtes Fehlverhalten zeigen. Das Verhalten der anderen Empfänger ist hingegegen bis auf Kleinigkeiten relativ regulär. Das Fehlverhalten aller Empfänger ist in den folgenden Abschnitten dargestellt:

9.4.1 Stellar

10 MHz-Ausgang

Vergleicht man die Ablage des 10 MHz-Pulses mit der des 1 PPS-Pulses (siehe auch Abschnitt 9.5), so kann festgestellt werden, daß die Differenz der Ablagen bis auf das Logikanalysator-Granularitätsrauschen ($\sigma_x =$ 6 ns) konstant bleibt (siehe Abbildung A.47). Dies läßt den Schluß zu, daß der 1 PPS-Ausgang lediglich eine herabgeteilte Form des 10 MHz-Ausgangs darstellt.

Pulsauslassungen

Sowohl der 10 MHz-Puls als auch der 1 PPS-Puls traten in jeder Sekunde auf.

Fehlpulse

Beim Stellar-Empfänger traten insgesamt 12 Fehlpulse, alle bis auf einen knapp nach dem korrekten — synchron liegenden — Puls, auf. In Tabelle 9.2 sind die Daten der Fehlpulse wiedergegeben. Dabei ist, neben der Angabe für die Ablage des korrekten Pulses, die Ablage des Fehlpulses vom korrekten Puls in Nanosekunden angegeben (diese ist negativ, falls der Fehlpuls vor dem korrekten Puls auftrat). Gelegentlich kamen mehrere Fehlpulse knapp hintereinander. Das könnte ein Hinweis auf einen möglichen Störeinfluß durch das Interface sein.

Nr.	Datum (GPST)	Meßsekunde	Abl. Puls (ns)	Abl. Fehlpuls gg. Puls (ns)
1	1996 Jan. 08 06:35:31	2313854	250041221	43275100
2	1996 Jan. 08 09:54:20	2325783	250040373	693806145
3	1996 Jan. 08 09:54:20	2325783	250040373	693806241
4	1996 Jan. 10 13:19:59	2510922	250026583	590930528
5	1996 Jan. 10 13:20:00	2510923	250026583	185173534
6	1996 Jan. 17 13:08:20	3115023	249980681	425755826
7	1996 Jan. 18 12:52:49	3200492	249974337	373310791
8	1996 Jan. 18 12:52:49	3200492	249974337	373310847
9	1996 Jan. 18 12:52:49	3200492	249974337	373310911
10	1996 Jan. 18 12:52:49	3200492	249974337	373311007
11	1996 Jan. 18 12:57:39	3200782	249974345	19877770
12	1996 Jan. 30 13:57:07	4241150	249891366	-78588353

Tabelle 9.2: Fehlpulse des Stellar

Fehlende Nachrichten

6 mal ging die `STATUS`-Nachrichten allein verloren. Die zweite Nachricht `TAF` ging 1 mal alleine verloren und 2 mal gingen beide Nachrichten zugleich verloren.

Gültigkeit und Sekundenanteil

Jedesmal wenn die `STATUS`-Nachricht ankam, war der 1 PPS Puls als gültig (`DATA VALID`) gekennzeichnet. Außerdem war der Sekundenanteil in dieser Nachricht immer korrekt.

9.4.2 NavSymm

Irreguläres Verhalten und Gültigkeit

In den Meßdaten konnten 8 Fälle gefunden werden, in denen es zu plötzliche Pulssprüngen kam. Der Vorgang lief dabei folgendermaßen ab: Zuerst änderte sich der Wert ACC-TIME der `SFM`-Nachricht des NavSymm (dieser gibt die erwartete 2σ–Genauigkeit des Pulses an) auf den Wert Null und zeigte damit eine inakzeptable Genauigkeit des Pulses an. Einige Sekunden später sprang die Ablage des Pulses um mehrere Millisekunden von der korrekten Ablage weg, wobei der ACC-TIME während dieser starken Ablage plötzlich wieder Werte ungleich Null annahm und damit eine nichtvorhandene Genauigkeit von einigen 100 ns Abweichung gegen UTC vorgab. Am Schluß kehrte die Ablage langsam wieder zu normalen Werten zurück. In Abbildung 9.4 ist einer dieser 8 Pulssprünge dargestellt. Die langsame Rückkehr der Ablage ist in Abbildung 9.5 genauer dargestellt. (Die Pulsspur ist zur Verdeutlichung jeweils fett hervorgehoben.) In Tabelle 9.3 ist dieser Sprung dokumentiert. Dabei kennzeichnet die letzte Spalte *ACC-TIME*, das entsprechende Feld der `SFM`-Nachricht. In den Zeilen, in denen das Feld *ACC-TIME* bzw. *Ablage* leer ist, ist zu dem zugehörigen Puls keine Nachricht übermittelt worden bzw. trat kein Puls auf. Spätestens 130 s nach Auftreten der ersten Irregularität (ACC-TIME=0) war der ganze Spuk wieder vorbei.

Tabelle 9.4 listet die wichtigsten Daten für jeden der 8 Fälle auf. Dabei kennzeichnet die Spalte *Beginn*, den Zeitpunkt, zu dem das erste Mal ACC-TIME gleich Null war. Die Spalte *Gesamtdauer* beschreibt die Anzahl der Sekunden seit dem Beginn der Anomalie bis zu dem Zeitpunkt, da die Ablage wieder normale Werte annahm. In der Spalte *Dauer1* ist die Zeitdauer angegeben, zwischen dem Zeitpunkt als der Wert ACC-TIME wieder normale Werte anzeigte und dem Zeitpunkt, da der Puls tatsächlich wieder korrekte Ablagen hatte. Die

Spalte *Dauer2* beschreibt die Dauer zwischen dem Zeitpunkt als die Ablage betragsmäßig kleiner als 2 μs war und dem Zeitpunkt als wieder korrekte Ablagen gemessen wurden. In der letzten Spalte schließlich ist die erste Ablage des Pulses nach dem Pulssprung eingetragen.

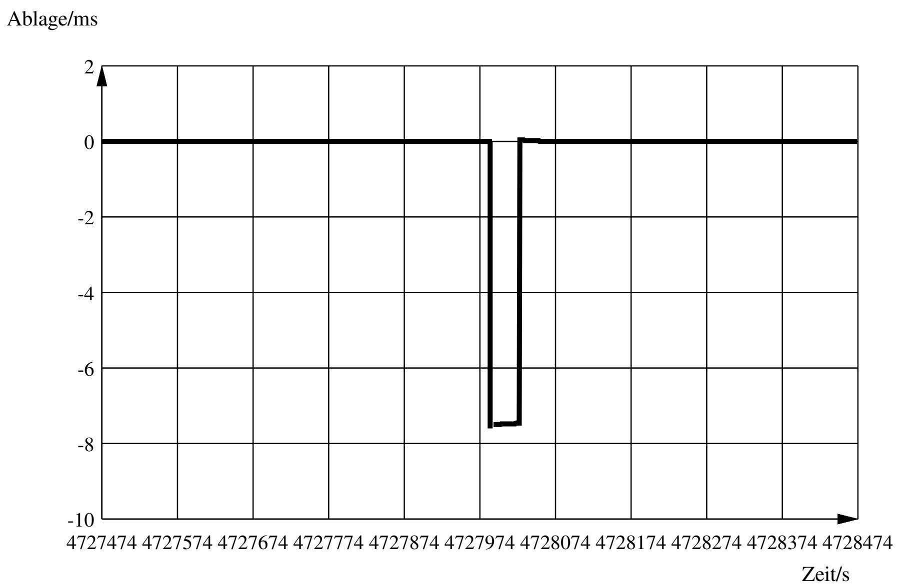

Abbildung 9.4: Einer der 8 Pulssprünge des NavSymm

Bei der zweiten auftretenden Anomalie des NavSymm nimmt ACC-Time — wie bei den Pulssprüngen — den Wert Null an. Dann läuft der Puls langsam von der korrekten Ablage weg und erreicht eine maximale Ablagen in der Größenordnung von 1 μs. Kurz danach nimmt ACC-TIME wieder korrekte Werte an, worauf die Ablage des Pulses langsam zur korrekten Lage zurückkehrt. Dieses Phänomen trat genau 6 mal auf. Abbildung 9.6 stellt diesen Vorgang für einen der Fälle graphisch dar. In Tabelle 9.5 ist er dokumentiert. Eine Übersicht über alle 6 Fälle zeigt Tabelle 9.6. Hierbei bedeutet die Spalte *Beginn* in der Tabelle wiederum den Zeitpunkt als ACC-TIME erstmals den Wert Null annahm. Die *Gesamtdauer* beschreibt die Dauer seit diesem Beginn bis zu dem Zeitpunkt als die Ablagen wieder korrekt waren. *Dauer1* bezeichnet die Dauer vom Beginn bis zum Zeitpunkt als ACC-TIME korrekte Werte anzeigte. Die letzte Spalte *extremste Ablage* schließlich enthält den extremsten (betragsmäßig größten) auftretenden Ablagenwert.

Neben diesen zwei Anomalien trat noch ein 10 Sekunden währendes Phänomen auf (Beginn 1995 Dec. 17 12:10:10 GPST), wobei der Wert ACC-TIME Null wurde, die Ablage des Pulses allerdings überhaupt keine Veränderung zeigte und ACC-TIME nach 10 Sekunden wieder korrekte Werte annahm.

Es kam nur während einer der ersten beiden erwähnten Anomalien zu Pulsauslassungen. Insgesamt gingen dabei 6 Pulse mitsamt der Nachricht `GPS` verloren. Zu Nachrichtenverluste kam es nur während einer der drei Anomalien. Dabei gingen insgesam 305 `GPS`-Nachrichten und 299 `SFM`-Nachrichten verloren. Dies entspricht einer relativen Häufigkeit von etwa $5,4 \cdot 10^{-6}$. In den `SFM`-Nachrichten war das ACC-TIME–Feld in 538 Fällen (rel. Häufigkeit $9,6 \cdot 10^{-5}$) Null.

Das anomale Pulsverhalten erschwert natürlich die statistische Analyse. Deshalb müssen Pulse, die während solcher Phasen auftreten ausgeklammert werden. Dazu wurde der ACC-TIME–Wert in der entprechenden Nachricht benutzt. In allen Fällen zeigte dieser Wert noch vor der Anomalie einen Wert von Null. Allerdings nahm er, bevor der Puls wieder synchron mit UTC war, bereits normale Werte an. Deswegen wurde ein *gültiger* Puls folgendermaßen definiert: Alle Pulse, deren letzten 150 Vorgängerpulse kein ACC-TIME–Wert von Null zeigen, sind gültige Pulse. Eine Analyse ergab, daß mit dieser Festlegung 2.839 Pulse (rel. Häufigkeit $0,5 \cdot 10^{-4}$) ungültig waren. In den folgenden Auswertungen wurden wegen des extrem irregulären Verhaltens ungültiger Pulse nur mehr gültige Pulse betrachtet (es wird allerdings jeweils darauf hingewiesen).

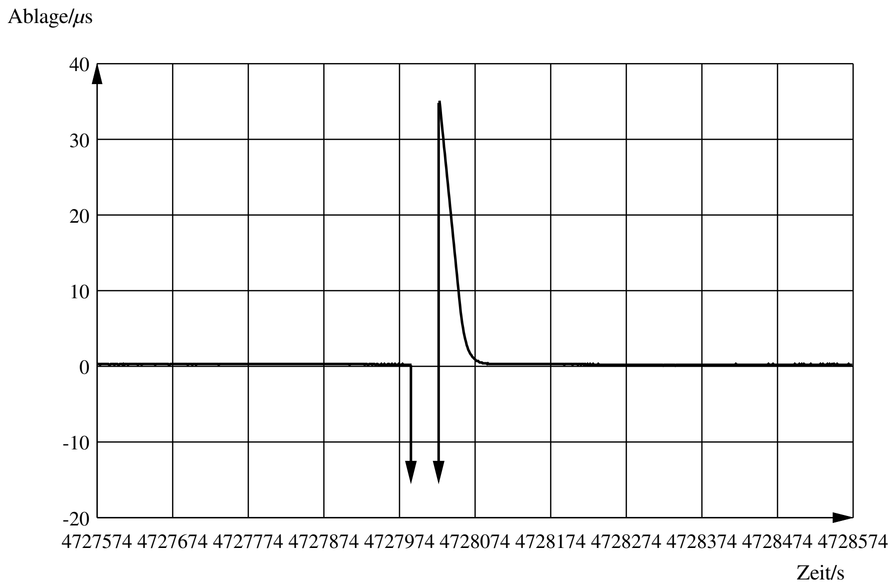

Abbildung 9.5: Rückkehr des 1 PPS-Pulses des NavSymm zur korrekten Ablage

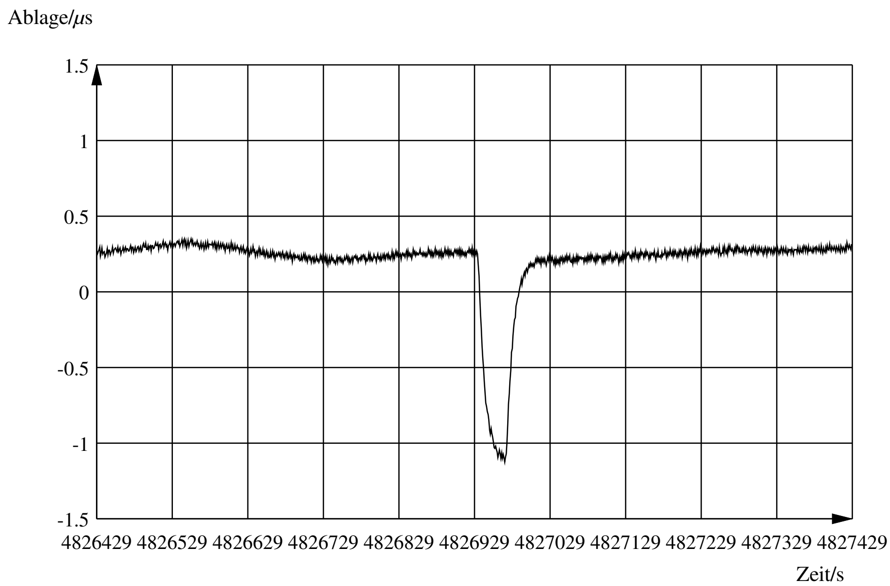

Abbildung 9.6: Einer der 6 Pulsabweichungen des NavSymm

Datum(GPST)	Ablage	ACC-TIME
1996 Feb. 05 05:10:53	220	0
1996 Feb. 05 05:10:54	204	0
⋮		
1996 Feb. 05 05:11:06	217	0
1996 Feb. 05 05:11:07	217	0
1996 Feb. 05 05:11:08		
1996 Feb. 05 05:11:09	-7499753	0
1996 Feb. 05 05:11:10	-7498923	0
1996 Feb. 05 05:11:11	-7497883	0
⋮		
1996 Feb. 05 05:11:34	-7474906	0
1996 Feb. 05 05:11:35	-7473914	0
1996 Feb. 05 05:11:36	-7472930	0
1996 Feb. 05 05:11:37	-7471935	0
1996 Feb. 05 05:11:38	-7470943	
1996 Feb. 05 05:11:39	-7469912	450
1996 Feb. 05 05:11:40	-7468928	450
1996 Feb. 05 05:11:41	-7467935	450
1996 Feb. 05 05:11:42	-7466945	450
1996 Feb. 05 05:11:43	-7465903	450
1996 Feb. 05 05:11:44	35071	450
1996 Feb. 05 05:11:45	34066	450
1996 Feb. 05 05:11:46	33058	450
⋮		
1996 Feb. 05 05:12:03	16109	300
1996 Feb. 05 05:12:04	15101	300
1996 Feb. 05 05:12:05	14074	300
⋮		
1996 Feb. 05 05:12:21	2539	300
1996 Feb. 05 05:12:22	2227	300
1996 Feb. 05 05:12:23	1955	300
1996 Feb. 05 05:12:24	1795	300
⋮		
1996 Feb. 05 05:12:38	533	300
⋮		
1996 Feb. 05 05:12:54	318	300
1996 Feb. 05 05:12:55	302	300

Tabelle 9.3: Einer der 8 Pulssprünge des NavSymm

Beginn (GPST)	Gesamtdauer/s	Dauer1/s	Dauer2/s	Pulssprung/ns
1995 Dec. 23 08:10:53	81	51	15	9000245
1995 Dec. 28 07:50:53	58	30	6	10750166
1996 Jan. 02 07:30:53	94	48	5	-7499865
1996 Jan. 04 07:21:29	111	65	23	24750268
1996 Jan. 14 06:40:53	78	50	25	-2999719
1996 Jan. 19 06:20:53	119	73	32	-5249701
1996 Feb. 05 05:10:53	130	84	40	-7499753
1996 Feb. 10 04:50:53	89	73	43	2250153

Tabelle 9.4: Übersicht über die Pulssprünge des NavSymm

10 MHz-Ausgang

Vergleicht man die Ablage des 10 MHz-Pulses mit der des 1 PPS-Pulses, so kann festgestellt werden, daß die Differenz der Ablagen — dies gilt auch für die Pulssprünge — immer innerhalb eines Intervalls von 112 ns schwanken (siehe Abbildung A.48). Der 10 MHz-Puls dürfte daher zumindest langfristig dem 1 PPS-Puls nachgesteuert werden.

Sekundenanteil

Der Empfänger NavSymm zeigte einen weiteren kuriosen Fehler: 31 mal (rel. Häufigkeit $5{,}5 \cdot 10^{-6}$ war der Tageswert des Datums um eins zu klein. Die genauen Fehler des Sekundenanteils sind in Tabelle 9.7 wiedergegeben. Zu keinem anderen Zeitpunkt war der Sekundenanteil falsch.

9.4.3 Motorola

Pulsauslassungen und fehlende Nachrichten

10 mal ging der Puls des Motorola immer gemeinsam mit der Nachricht verloren.

Gültigkeit und Sekundenanteil

Immer wenn die Nachricht kam, war das Statusbyte des Empfängers auf 8 gesetzt (*Position Hold*), siehe 6.2.4. Außerdem war der Sekundenanteil jedesmal korrekt.

Die in der Nachricht ebenfalls enthaltene Information, wieviele Satelliten getrackt wurde, zeigt die in Tabelle 9.8 dargestellte Verteilung. Die relative Häufigkeit ergibt sich dabei durch Dividieren der absoluten Häufigkeit durch die Anzahl der aufgetretenen Nachrichten 5,590.622.

Wie aus der Tabelle zu sehen ist, wurden immer zumindest 2 Satelliten getrackt, was für den position-hold-mode ausreicht (1 Satellit erforderlich). Daß 225 Sekunden auftreten, in denen weniger als 4 Satelliten getrackt wurden, liegt vermutlich daran, daß die Dämpfung über das zusätzlich eingefügte 4 m lange Koaxialkabel für Satelliten mit kleinem Höhenwinkel (und entsprechend geringer Signalstärke am Empfangsort) ein wenig zu groß ist.

Datum(GPST)	Ablage	ACC-TIME
1996 Feb. 06 08:40:06	238	0
1996 Feb. 06 08:40:07	270	0
1996 Feb. 06 08:40:08	246	0
1996 Feb. 06 08:40:09	268	0
1996 Feb. 06 08:40:10	251	0
1996 Feb. 06 08:40:11	179	0
1996 Feb. 06 08:40:12	107	0
1996 Feb. 06 08:40:13	-25	
1996 Feb. 06 08:40:14	-153	0
1996 Feb. 06 08:40:15	-233	0
1996 Feb. 06 08:40:16	-361	0
1996 Feb. 06 08:40:17	-436	0
⋮		
1996 Feb. 06 08:40:39	-1047	0
1996 Feb. 06 08:40:40	-1071	0
1996 Feb. 06 08:40:41	-1098	0
1996 Feb. 06 08:40:42	-1066	0
1996 Feb. 06 08:40:43	-1098	
1996 Feb. 06 08:40:44	-1074	
1996 Feb. 06 08:40:45	-1095	200
1996 Feb. 06 08:40:46	-1126	200
1996 Feb. 06 08:40:47	-1094	200
1996 Feb. 06 08:40:48	-1070	200
1996 Feb. 06 08:40:49	-1001	200
1996 Feb. 06 08:40:50	-873	200
1996 Feb. 06 08:40:51	-745	200
1996 Feb. 06 08:40:52	-673	200
1996 Feb. 06 08:40:53	-590	200
1996 Feb. 06 08:40:54	-518	200
1996 Feb. 06 08:40:55	-398	200
1996 Feb. 06 08:40:56	-374	200
1996 Feb. 06 08:40:57	-290	200
1996 Feb. 06 08:40:58	-224	200
1996 Feb. 06 08:40:59	-185	200
1996 Feb. 06 08:41:00	-168	200
1996 Feb. 06 08:41:01	-93	
1996 Feb. 06 08:41:02	-69	200
1996 Feb. 06 08:41:03	-45	200
1996 Feb. 06 08:41:04	-29	200
1996 Feb. 06 08:41:05	0	200
1996 Feb. 06 08:41:06	32	200
1996 Feb. 06 08:41:07	48	200
⋮		
1996 Feb. 06 08:41:24	186	200

Tabelle 9.5: Einer der 6 Pulsabweichungen des NavSymm

Beginn (GPST)	Gesamtdauer/s	Dauer1/s	extremste Ablage/ns
1996 Feb. 06 08:40:06	72	39	-1126
1996 Feb. 07 08:40:05	71	40	-1988
1996 Feb. 09 08:30:05	70	40	-1031
1996 Feb. 11 08:20:06	69	39	-286
1996 Feb. 12 08:20:05	68	40	-406
1996 Feb. 14 08:10:05	76	40	-210

Tabelle 9.6: Übersicht über die Pulsabweichungen des NavSymm

korrekter Sekundenanteil (UTC)	fehlerhafter Sekundenanteil
1995 Dec. 15 00:00:00	1995 Dec. 14 00:00:00
1995 Dec. 15 00:00:01	1995 Dec. 14 00:00:01
1995 Dec. 15 00:00:02	1995 Dec. 14 00:00:02
1995 Dec. 19 00:00:00	1995 Dec. 18 00:00:00
1995 Dec. 19 00:00:01	1995 Dec. 18 00:00:01
1995 Dec. 19 00:00:02	1995 Dec. 18 00:00:02
1995 Dec. 25 00:00:00	1995 Dec. 24 00:00:00
1995 Dec. 25 00:00:01	1995 Dec. 24 00:00:01
1995 Dec. 25 00:00:02	1995 Dec. 24 00:00:02
1995 Dec. 25 00:00:02	1995 Dec. 24 00:00:02
1996 Jan. 08 00:00:00	1996 Jan. 07 00:00:00
1996 Jan. 14 00:00:00	1996 Jan. 13 00:00:00
1996 Jan. 14 00:00:01	1996 Jan. 13 00:00:01
1996 Jan. 20 00:00:00	1996 Jan. 19 00:00:00
1996 Jan. 20 00:00:01	1996 Jan. 19 00:00:01
1996 Jan. 24 00:00:00	1996 Jan. 23 00:00:00
1996 Jan. 24 00:00:01	1996 Jan. 23 00:00:01
1996 Jan. 27 00:00:00	1996 Jan. 26 00:00:00
1996 Jan. 27 00:00:01	1996 Jan. 26 00:00:01
1996 Jan. 31 00:00:00	1996 Jan. 30 00:00:00
1996 Jan. 31 00:00:01	1996 Jan. 30 00:00:01
1996 Feb. 03 00:00:00	1996 Feb. 02 00:00:00
1996 Feb. 03 00:00:01	1996 Feb. 02 00:00:01
1996 Feb. 04 00:00:00	1996 Feb. 03 00:00:00
1996 Feb. 04 00:00:01	1996 Feb. 03 00:00:01
1996 Feb. 07 00:00:00	1996 Feb. 06 00:00:00
1996 Feb. 07 00:00:01	1996 Feb. 06 00:00:01
1996 Feb. 12 00:00:00	1996 Feb. 11 00:00:00
1996 Feb. 12 00:00:01	1996 Feb. 11 00:00:01
1996 Feb. 14 00:00:00	1996 Feb. 13 00:00:00
1996 Feb. 14 00:00:01	1996 Feb. 13 00:00:01

Tabelle 9.7: Der Fehler des NavSymm bei der Generierung des Sekundenanteils

getrackte Satelliten	abs. Häufigkeit	rel. Häufigkeit
0	0	0
1	0	0
2	1	$1,7 \cdot 10^{-7}$
3	224	$4 \cdot 10^{-5}$
4	12488	$2,2 \cdot 10^{-3}$
5	355239	6,4 %
6	5222670	93,4 %

Tabelle 9.8: Verteilung der Anzahl getrackter Satelliten

9.4.4 Trimble

Pulsauslassungen

Kein einziger Puls wurde ausgelassen.

Fehlende Nachrichten

Die erste Nachricht `TM` ging insgesamt 509 mal (rel. Häufigkeit $9 \cdot 10^{-5}$) verloren. Die zweite Nachricht `ST` kam beachtliche 25.144 mal (rel. Häufigkeit 0,45 %) nicht an.

Gültigkeit und Sekundenanteil

Die Nachricht `ST` zeigte 7.923 mal an, daß der Empfänger nur 3 verwendbare Satelliten tracken konnte. Die restlichen Male wurde gemeldet, daß sich der Empfänger im korrekten Betriebsmodus *doing-position-fixes* befindet.

Der Sekundenanteil beim Trimble steht in zwei Formen zur Verügung: Einmal in Bezug auf GPST, das anderemal bezüglich UTC. Der Sekundenanteil nach GPS war in jeder `TM`-Nachricht immer korrekt. Für den Sekundenanteil nach UTC verhielt sich der Empfänger zum Jahreswechsel 1995/1996 fehlerhaft: Die zu diesem Zeitpunkt in UTC einzufügende Schaltsekunde wird dort nämlich zu früh eingefügt. Bereits am 31. Dezember 1995 um 23:59:51 UTC (ist gleich 1. Januar 1996 00:00:01 GPST) betrug beim Trimble die Schaltsekundendifferenz zwischen GPST und UTC 11 s. Damit war der Sekundenanteil bis zum tatsächlichen Einfügungspunkt 1. Jänner 1996 um 00:00:00 UTC um 1 zu klein.

Die Anzahl der verwendbaren Satelliten verteilte sich laut `TM`-Nachrichten nach der in Tabelle 9.9 angegebenen Verteilungsgunktion.

Verwendbare Satelliten	absolute Häufigkeit	relative Häufigkeit
1	0	0 %
2	0	0 %
3	33067	0,6 %
4	5557056	99,4 %

Tabelle 9.9: Verteilung verwendbarer Satelliten des Trimble

Dauer/s	absolute Häufigkeit
1	228
2	80
3	18
4	10
5–9	17
10–19	10
20–49	39
50–99	15
100–149	17
150–199	23
200–249	41
268	1
272	1
401	1
838	1
994	1

Tabelle 9.10: Verteilung der Dauer des Coastingmodus des Magellan

9.4.5 Magellan

Pulsauslassungen und fehlende Nachrichten

Drei 1 PPS-Pulse und 116 Nachrichten gingen verloren.

Irreguläres Verhalten und Gültigkeit

18.984 der Nachrichten vom Magellan zeigten an, daß sich der Empfänger im *coasting*-Modus *In diesem Modus erhält der Empfänger aus irgendeinem Grund — etwa, weil zu wenig Satelliten vorhanden sind — keine Meßwerte und der 1 PPS-Puls wird mit dem bekannten Frequenzfehler der lokalen Uhr weiter ausgegeben.* befindet. Dies entspricht einer relativen Häufigkeit von $0,34$ %. Dieser Modus trat meist in Blöcken, das heißt ohne Unterbrechung mehrere Sekunden dauernd, auf. Die Verteilung der Dauer dieser Blöcke ist in Tabelle 9.10 dargestellt.

Die Verteilung des TFOM-Werts, der den erwarteten Zeitfehler angibt, ist aus Tabelle 9.11 zu entnehmen.

Betrachtet man die Ablagen der Pulse während Phasen schlechter Genauigkeit (TFOM>4) (539.812 Fälle, rel Häufigkeit 9,9 %) und im *coasting*-Betrieb (18.984 Fälle, relative Häufigkeit 0,3 %), so stellt man fest, daß der Empfänger sich sehr irregulär verhält. Es treten einerseits Pulssprünge mit einer Sprungweite von 10 μs, und 1 ms auf. Andererseits driftet die Ablage während, sich der Empfänger im *coasting* Betrieb befindet, linear von der Referenzzeit weg und springt bei Erreichen des Normalzustands wieder zur korrekten Ablage zurück. Wegen dieses ungünstigen Verhaltens wurden — wie beim NavSymm — *gültige Pulse* definiert: Wird über die Nachricht `K3` ein TFOM=3 oder TFOM=4 angezeigt und ist der Empfänger laut Nachricht nicht im *coasting*-Betrieb, so liegt ein gültiger Puls vor. Die Anzahl der korrespondierenden ungültigen Pulse beträgt 551.511. Diese traten meist in Blöcken auf. Die zugehörige Verteilung der Dauer dieser Blöcke ist in Tabelle 9.12 wiedergegeben. Wegen des ungünstigen Verhalten ungültiger Pulse werden im folgenden nur mehr gültige Pulse betrachtet.

Trotz der Einschränkung auf gültige Pulse traten noch einige größere Ablagen auf: Jeweils vier Pulssprünge um ziemlich genau 1 ms traten völlig isoliert auf. 7 Ablagen um ungefähr 3,3 μs traten hintereinander nach

TFOM	Erwarteter Zeitfehler x	absolute Häufigkeit	relative Häufigkeit
1	$0 < x < 1$ (ns)	0	0
2	$1 < x < 10$ (ns)	0	0
3	$10 < x < 100$ (ns)	317458	5,7 %
4	$100 < x < 1000$ (ns)	4733240	84,7 %
5	$1 < x < 10$ (μs)	538427	9,6 %
6	$10 < x < 100$ (μs)	20	$3{,}6 \cdot 10^{-6}$
7	$100 < x < 1000$ (μs)	86	$1{,}5 \cdot 10^{-5}$
8	$1 < x < 10$ (ms)	0	0
9	$10 < x < 100$ (ms)	0	0
10-14	*unbenutzt*	0	0
15	keine Information verfügbar	1279	$2{,}3 \cdot 10^{-4}$

Tabelle 9.11: Verteilung für den TFOM der Magellan-Nachricht

einem Block im Coastingmodus auf. Dasselbe geschah für zwei hintereinander auftretende 10 μs–Ablagen. In beiden Fällen wurde offenbar zu früh das Coasting-Flag zurückgesetzt. Diese Pulse (im folgenden als Ausreißer bezeichnet) werden bei der statistischen Auswertung nicht berücksichtigt.

Sekundenanteil

Der Magellan-Empfänger produziert relativ häufig fehlerhafte Sekundenanteile: Insgesamt 5.799 mal (relative Häufigkeit 0,1 %) war der Sekundenanteil inkorrekt. Tabelle 9.13 zeigt den typischen Fall eines solchen Fehlers. Dabei traten fehlerhafte Sekundenanteile während ungültiger Pulse gehäuft auf (nämlich insgesamt 4.667 mal). Diese Sekundenanteilsfehler traten meist mehrfach hintereinander (in Blöcken) auf. Dabei zeigte die Dauer dieser Blöcke, die in Tabelle 9.14 dargestellte Verteilung. Der längste dieser Blöcke war 993 Sekunden lang!

9.4.6 Rockwell

Ablage des 1 PPS-Pulses

Wie in Abschnitt 9.5 beschrieben schwankt die Ablage des 1 PPS-Pulses des Rockwell über eine Breite von 10 μs! um die korrekte Lage.

Pulsauslassungen

Kein einziger Puls wurde ausgelassen.

Fehlende Nachricht, Sekundenanteil und Gültigkeit

192 mal ging die Nachricht verloren. Eine empfangene Nachricht hatte eine falsche Prüfsumme. Die durch das Flag *UTC-Validity* bestimmte Gültigkeit war in ankommenden Nachrichten immer gesetzt. Außerdem war der Sekundenanteil immer korrekt.

Dauer/s	absolute Häufigkeit
1	277
2	73
3	16
4	10
5	5
6–9	6
10–24	147
25–49	176
50–99	93
100–149	171
150–199	83
200–249	83
250–499	213
500–999	226
1000–1499	109
1500–1999	20
2000–2999	26
3000–3999	7
4008	1
4419	1
4809	1

Tabelle 9.12: Verteilung der Dauer der Phasen mit ungültigen Pulsen des Magellan

korrekter Sekundenanteil (GPS)	Sekundenanteil des Magellan
1995 Dec. 13 10:41:52	1995 Dec. 13 10:41:52
1995 Dec. 13 10:41:53	1995 Dec. 13 10:41:52
1995 Dec. 13 10:41:54	1995 Dec. 13 10:41:53
1995 Dec. 13 10:41:55	1995 Dec. 13 10:41:55

Tabelle 9.13: Fehlerhafter Sekundenanteil beim Magellan

Dauer/s	absolute Häufigkeit
1	6
2	170
3	19
4	5
5	1
7	1
8	1
10–29	5
30-99	7
100–299	18
401	1
993	1

Tabelle 9.14: Dauer von Sekundenanteilsfehler beim Magellan

Empfänger	Pulsauslassungen	Ungültige Pulse	Verlust Nachricht 1	Verlust Nachricht 2
Stellar	0	0	8	3
NavSymm	6	2.839	305	299
Motorola	10	0	10	
Trimble	0	0	509	25.144
Magellan	3	551.511	106	
Rockwell	0	0	193	

Tabelle 9.15: Übersicht über das Fehlerverhalten der Empfänger

9.4.7 Übersicht

Die Tabelle 9.15 zeigt eine Übersicht über Pulsausslassungen, die Anzahl ungültiger Pulse und die Anzahl der verlorenen Nachrichten 1 bzw. 2.

9.5 Verteilungsfunktionen für Ablagen gegen die Referenzzeit

Die Verteilungsfunktionen der gegen die Referenzzeit gemessenen Ablagen x der 1 PPS-Pulse, sind im Anhang wiedergegeben. Tatsächlich enthalten sie allerdings nur die statistischen Anteile. Zur Erhöhung der Übersicht wurde nämlich die mittlere Ablage $\overline{x}$ der jeweiligen Pulse vorher subtrahiert. Damit sind die in der Tabelle und den Abbildungen sichtbaren Mittelwerte alle Null. Die Graphen der Verteilungsfunktion sind in den Abbildungen A.11 bis A.16 (Seiten 130–135) zu finden. Die tabellarische Auflistung der Verteilungsfunktion kann den Tabellen A.1 bis A.6 (Seiten 130–135) entnommen werden. Die Tabellen enthalten neben der Standardabweichung σ der Verteilung, auch die linke und rechte Abschrankung ε_- und ε_+ der auftretenden Ablagen von den Referenzzeit. Diese Werte geben das kleinste Intervall $(\varepsilon_-, \varepsilon_+)$ an, in dem alle Ablagen enthalten waren.

Die charakteristischen Werte der Verteilungsfunktionen der Pulsablagen gegen die Referenzzeit sind als Übersicht in Tabelle 9.16 aufgelistet. Neben dem Mittelwert $\overline{x}$ und der Standardabweichung σ_x sind auch die linke und rechte Abschrankung ε_- und ε_+ angeschrieben.

Signal	Mittelwert $\overline{x}$/ns	Std.abweichung σ_x/ns	ε_-/ns	ε_+/ns
Stellar 1 PPS	1	38	-189	193
NavSymm 1 PPS[a]	245	46	14	610
Motorola 1 PPS	108	46	-133	360
Trimble 1 PPS	310	309	-993	1.655
Magellan 1 PPS[b]	60	155	-1.539	1.739
Rockwell 1 PPS	7.933	2.380	2.495	12.499

[a] Nur für gültige Pulse
[b] Nur für gültige Pulse, die keine Ausreißer sind (Ablage $< 2{,}0\,\mu$s).

Tabelle 9.16: Charakteristische Werte der Verteilungsfunktion

9.6 Verteilungsfunktionen der 1 PPS–Ablagen für kurze Mittelungszeiten

Im Projekt SynUTC ist die Frage, wie sich die 1 PPS-Pulse eines Empfängers über kürzere Mittelungszeiten ($\tau = 1, 10, 30, 100$ s) verhalten, von großem Interesse. Zu diesem Zweck wird die Verteilungsfunktion $\mathcal{V}(\Delta x(t,\tau))$ der Differenz zwischen τ Zeiteinheiten auseinanderliegenden Ablagen der 1 PPS-Pulse $x(t)$ und $x(t,\tau)$ betrachtet.

Ziel dieser Untersuchung ist es, absolute Grenzen für die maximale über die Zeitdauer τ akkumulierte Zeitabweichung $\Delta x(\tau)$ der Signale zu erhalten. Mit dieser Abschrankung kann auch der Frequenzfehler eingegrenzt werden. Die maximale mittlere Frequenzabweichung beträgt dann nämlich $\rho(\tau) = \Delta x(\tau)/\tau$. Daneben sind natürlich auch die charakteristischen Werte der Verteilung von Interesse.

Im Anhang sind die Graphen und Tabellen zu den Verteilungsfunktionen für Mittelungszeiten $\tau = 1, 10, 30, 100$ s für alle 6 Empfänger wiedergegeben (Für den NavSymm wurden nur gültige Pulse, für den Magellan nur gültige Pulse, die keine Ausreißer sind, berücksichtigt). Die Auswertungsergebnise sind in den Abbildungen A.17 bis A.40 und den Tabellen A.7 bis A.30 abgedruckt.

In Tabelle 9.17 können die Standardabweichungen $\sigma_{\Delta x}(\tau)$, der Betrag der maximalen Abweichung $\varepsilon(\tau)$ und die maximale mittlere Frequenzabweichung $\rho(\tau)$ der Verteilungsfunktion abgelesen werden.

Puls	$\sigma_{\Delta x}(\tau)$/ns	$\varepsilon(\tau)$/ns	$\rho(\tau)$/ns
Stellar $\tau = 1$ s	6,5	29	$2,9 \cdot 10^{-8}$
Stellar $\tau = 10$ s	7,2	40	$4,0 \cdot 10^{-9}$
Stellar $\tau = 30$ s	10,9	80	$2,7 \cdot 10^{-9}$
Stellar $\tau = 100$ s	27,2	196	$2,0 \cdot 10^{-9}$
NavSymm $\tau = 1$ s	22,1	104	$1,0 \cdot 10^{-7}$
NavSymm $\tau = 10$ s	23,3	152	$1,5 \cdot 10^{-8}$
NavSymm $\tau = 30$ s	27,1	201	$6,7 \cdot 10^{-9}$
NavSymm $\tau = 100$ s	47,5	399	$4,0 \cdot 10^{-9}$
Motorola $\tau = 1$ s	44,0	184	$1,8 \cdot 10^{-7}$
Motorola $\tau = 10$ s	43,9	284	$2,8 \cdot 10^{-8}$
Motorola $\tau = 30$ s	44,8	252	$8,4 \cdot 10^{-9}$
Motorola $\tau = 100$ s	51,7	290	$2,9 \cdot 10^{-9}$
Trimble $\tau = 1$ s	234,7	1147	$1,1 \cdot 10^{-6}$
Trimble $\tau = 10$ s	418,0	1672	$1,7 \cdot 10^{-7}$
Trimble $\tau = 30$ s	383,5	1688	$5,6 \cdot 10^{-8}$
Trimble $\tau = 100$ s	413,6	1715	$1,7 \cdot 10^{-8}$
Magellan $\tau = 1$ s	44,6	1280	$1,3 \cdot 10^{-6}$
Magellan $\tau = 10$ s	51,2	1848	$1,8 \cdot 10^{-7}$
Magellan $\tau = 30$ s	70,6	1896	$6,3 \cdot 10^{-8}$
Magellan $\tau = 100$ s	139,1	1865	$1,9 \cdot 10^{-8}$
Rockwell $\tau = 1$ s	205,8	6460	$6,5 \cdot 10^{-6}$
Rockwell $\tau = 10$ s	193,2	6666	$6,6 \cdot 10^{-7}$
Rockwell $\tau = 30$ s	210,4	6685	$2,2 \cdot 10^{-7}$
Rockwell $\tau = 100$ s	251,6	6997	$7,0 \cdot 10^{-8}$

Tabelle 9.17: Charakteristika der Verteilungsfunktionen der Inter-1 PPS-Ablagen

Puls	$\sigma_{\Delta x}(\tau)$/ns	$\varepsilon(\tau)$/ns	$\rho(\tau)$/ns
Stellar 10 MHz $\tau = 10$ s	6,1	34	$3,4 \cdot 10^{-9}$
Stellar 10 MHz $\tau = 30$ s	10,2	84	$2,8 \cdot 10^{-9}$
Stellar 10 MHz $\tau = 100$ s	27,0	200	$2,0 \cdot 10^{-9}$
NavSymm 10 MHz $\tau = 10$ s	26,2	146	$1,5 \cdot 10^{-8}$
NavSymm 10 MHz $\tau = 30$ s	29,7	224	$7,4 \cdot 10^{-9}$
NavSymm 10 MHz $\tau = 100$ s	49,2	394	$3,9 \cdot 10^{-9}$

Tabelle 9.18: Abschrankung der akkumulierten Zeitabweichung und der Frequenzabweichung für die 10 MHz-Ausgänge

9.7 Verteilungsfunktionen der 10 MHz-Ausgänge

Für das Projekt SynUTC ist das Frequenzverhalten der 10 MHz-Ausgänge von Interesse. Es ist insbesondere wichtig zu wissen, wie stark die Frequenzausgänge für Mittelungszeiten von etwa $\tau = 10$ s bis $\tau = 100$ s schwanken. Ähnlich wie bei der Untersuchung der Ablagen der 1 PPS-Pulse für verschiedene Mittelungszeiten wird hierbei die Verteilungsfunktion $\mathcal{V}(\Delta x(t,\tau))$ der Differenz zwischen τ Zeiteinheiten auseinanderliegenden Ablagen $x(t)$ und $x(t,\tau)$ betrachtet.

Ebenso wie dort ist es das Ziel, absolute Grenzen für die maximale, über die Zeitdauer τ akkumulierte Zeitabweichung $\Delta x(\tau)$ der Signale zu erhalten. Mit dieser Abschrankung wird wieder die Frequenz eingegrenzt. Die maximale mittlere Frequenzabweichung beträgt dann $\rho(\tau) = \Delta x(\tau)/\tau$.

In den im Anhang abgedruckten Abbildungen A.41 bis A.46 und den Tabellen A.31 bis A.36 sind die Verteilungsfunktionen der über verschiedene Mittelungszeiten $\tau = 10, 30, 100$ s akkumulierten Zeitabweichungen der 10 MHz-Pulse von Stellar und NavSymm dargestellt. Das Maximum der Absolutbeträge $\varepsilon(\tau)$in den Tabellen enthaltenen linken und rechten Schranken ε_- und ε_+ stellt die gewünschte Abschrankung dar. In Tabelle 9.18 sind diese neben den Werten $\sigma_{\Delta x}(\tau)$ und $\rho(\tau)$ in einer Übersicht aufgelistet.

9.8 Resümee

Die Auswertungen der Daten von etwa 2 Monaten (beinahe) kontinuierlicher Messung zeigen deutlich, daß GPS-Empfänger ein sehr vielfältiges Fehlverhalten an den Tag legen können. Obzwar einige Empfänger ein sehr günstiges Verhalten aufweisen (Stellar, Motorola und zum Teil Trimble) und innerhalb der Meßmöglichkeiten des Meßaufbaus ihre spezifizierten Genauigkeiten tadelos erreichen konnten, ist man offenbar nicht davor gefeit, daß auch relativ teure Empfänger — etwa NavSymm — ein schwerwiegendes Fehlverhalten aufweisen. Schwer wiegt dabei unter anderem auch, daß der 1 PPS-Puls der zuerst ausgelieferten Antennen-Empfänger–Einheit des NavSymm um ganze 250 μs von UTC ablag. Nach Reklamation (Der Hersteller gab, daß dies ein bekannter Softwarefehler sei) wurde die Antennen–Empfängereinheit gegen ein Exemplar ohne Softwarefehler ausgetauscht, die in der vorliegenden Evaluation vermessen wurde.

Für das Projekt als völlig ungeeignet erwiesen haben sich der Rockwell-Empfänger wegen seiner großen Ablage (10 μs) und der Magellan-Empfänger, welcher ein zu stark ausgeprägtes Fehlverhalten zeigt.

9.8.1 Verbesserungsvorschläge für zukünftige Messungen

Die vorliegende Messung zeigt noch deutliche Schwächen. Einerseits war die Kontinuität der Messung nicht vollständig gewährleistet (168 s ohne Meßdaten), andererseits wäre eine Steigerung der Genauigkeit der Referenzzeit wünschenswert. Insbesondere systematische Abweichung konnten nicht gemessen werden. Diese und andere Schwächen in zukünftigen Messungen vermeiden zu helfen, soll folgender Verbesserungskatalog dienen:

Verbessertes Interfacing

Die Fehlpulse des Stellars zeigen vermutlich die Schwächen des Interfacings auf. In zukünftigen Messungen sollte daher mehr Wert auf dieses Bereich des Meßaufbaus gelegt werden. Insbesondere sollten alle Eingänge der Frequenzteilerschaltung opto-entkoppelt und über zusätzliche Treiber geführt werden.

Genaue Bestimmung der Initialpositionen

Die ungenaue Kenntnis der Antennenpositionen verursacht systematische und stochastische Abweichung in der Größenordnung von 60 ns. Durch genaue Vermessung der Antennenpositionen mit Hilfe eines DGPS-Empfängers soll dieses Manko behoben werden. Mit der erhaltenen Position sollten sodann die im Time-Transfer Modus betriebenen Empfänger programmiert werden.

Getrennte Aufzeichnung von GPS-Nachrichten und Logikanalysatordaten

Um den Aufwand der Meßsoftware zu verkleinern, und damit ein stabileres Meßsystem zu erhalten, sollten die Pulsdaten und Daten der GPS-Nachrichten getrennt aufgezeichnet und abgespeichert werden. Die Kombination dieser Daten müßte dann natürlich in der Auswertungsphase erfolgen.

Vergleichsmessung mit einer hochwertigen Atomuhr

Ein wesentliches Problem der Messung ist die vage Kenntnis der Genauigkeit der Referenzzeit (sowohl betreffend systematische und stochastische Fehler). Um bessere Aussagen über sie machen zu können, sollte der Meßaufbau einige Wochen mit einer Atomuhr verglichen werden (eine realistische Möglichkeit wäre die Vermessung gegen die Atomuhr der TU Graz, die auch an der Darstellung der TAI und UTC-Zeitskalen beteiligt ist, und deren Ablage gegen diese Zeitskalen genau bekannt ist). Dies würde auch den schwerwiegenden Nachteil beseitigen, daß die GPS-Empfänger mit dem GPS-System selbst vermessen wurden (nämlich im wesentlichen mit dem besten aller verfügbaren GPS-Empfänger).

A Auswertungsergebnisse (Graphen und Tabellen)

Die folgenden Abbildungen und Tabellen enthalten die Ergebnisse der Auswertung. Die nachstehenden Tabellen geben eine Übersicht über alle Abbildungen in diesem Anhang mit Angabe ihrer Abildungs– und Seitennummer.

Bezeichnung	Abbildungs– und Seitennummer
Verteilung Motorola 1 PPS für Mittelungszeit $\tau = 1$ s	siehe Abb. A.25 auf Seite 144
Verteilung Motorola 1 PPS für Mittelungszeit $\tau = 10$ s	siehe Abb. A.26 auf Seite 145
Verteilung Motorola 1 PPS für Mittelungszeit $\tau = 30$ s	siehe Abb. A.27 auf Seite 146
Verteilung Motorola 1 PPS für Mittelungszeit $\tau = 100$ s	siehe Abb. A.28 auf Seite 147

Bezeichnung	Abbildungs– und Seitennummer
Verteilung Trimble 1 PPS für Mittelungszeit $\tau = 1$ s	siehe Abb. A.29 auf Seite 148
Verteilung Trimble 1 PPS für Mittelungszeit $\tau = 10$ s	siehe Abb. A.30 auf Seite 149
Verteilung Trimble 1 PPS für Mittelungszeit $\tau = 30$ s	siehe Abb. A.31 auf Seite 150
Verteilung Trimble 1 PPS für Mittelungszeit $\tau = 100$ s	siehe Abb. A.32 auf Seite 151

Bezeichnung	Abbildungs– und Seitennummer
Verteilung Magellan 1 PPS für Mittelungszeit $\tau = 1$ s	siehe Abb. A.33 auf Seite 152
Verteilung Magellan 1 PPS für Mittelungszeit $\tau = 10$ s	siehe Abb. A.34 auf Seite 153
Verteilung Magellan 1 PPS für Mittelungszeit $\tau = 30$ s	siehe Abb. A.35 auf Seite 154
Verteilung Magellan 1 PPS für Mittelungszeit $\tau = 100$ s	siehe Abb. A.36 auf Seite 155

Bezeichnung	Abbildungs– und Seitennummer
Verteilung Rockwell 1 PPS für Mittelungszeit $\tau = 1$ s	siehe Abb. A.37 auf Seite 156
Verteilung Rockwell 1 PPS für Mittelungszeit $\tau = 10$ s	siehe Abb. A.38 auf Seite 157
Verteilung Rockwell 1 PPS für Mittelungszeit $\tau = 1$ s	siehe Abb. A.39 auf Seite 158
Verteilung Rockwell 1 PPS für Mittelungszeit $\tau = 10$ s	siehe Abb. A.40 auf Seite 159

Bezeichnung	Abbildungs– und Seitennummer
Verteilung Stellar 1 PPS für Mittelungszeit $\tau = 10$ s	siehe Abb. A.41 auf Seite 160
Verteilung Stellar 1 PPS für Mittelungszeit $\tau = 30$ s	siehe Abb. A.42 auf Seite 161
Verteilung Stellar 1 PPS für Mittelungszeit $\tau = 100$ s	siehe Abb. A.43 auf Seite 162

Bezeichnung	Abbildungs– und Seitennummer
Verteilung NavSymm 1 PPS für Mittelungszeit $\tau = 10$ s	siehe Abb. A.44 auf Seite 163
Verteilung NavSymm 1 PPS für Mittelungszeit $\tau = 30$ s	siehe Abb. A.45 auf Seite 164
Verteilung NavSymm 1 PPS für Mittelungszeit $\tau = 100$ s	siehe Abb. A.46 auf Seite 165

Bezeichnung	Abbildungs– und Seitennummer
Verteilung Stellar 1 PPS gg. Stellar 10 MHz	siehe Abb. A.47 auf Seite 166
Verteilung NavSymm 1 PPS gg. NavSymm 10 MHz	siehe Abb. A.48 auf Seite 167

A.1 Übersicht der 1 PPS-Pulse über einen Ausschnitt von 1000 s

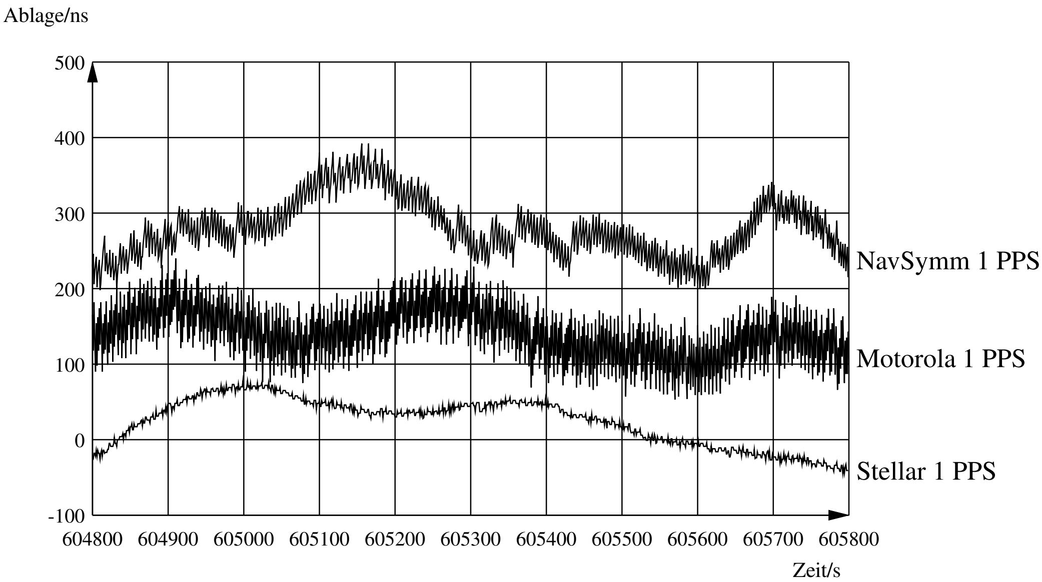

Abbildung A.1: Ablagen der Empfänger Stellar, NavSymm und Motorola für einen 1000 s–Ausschnitt

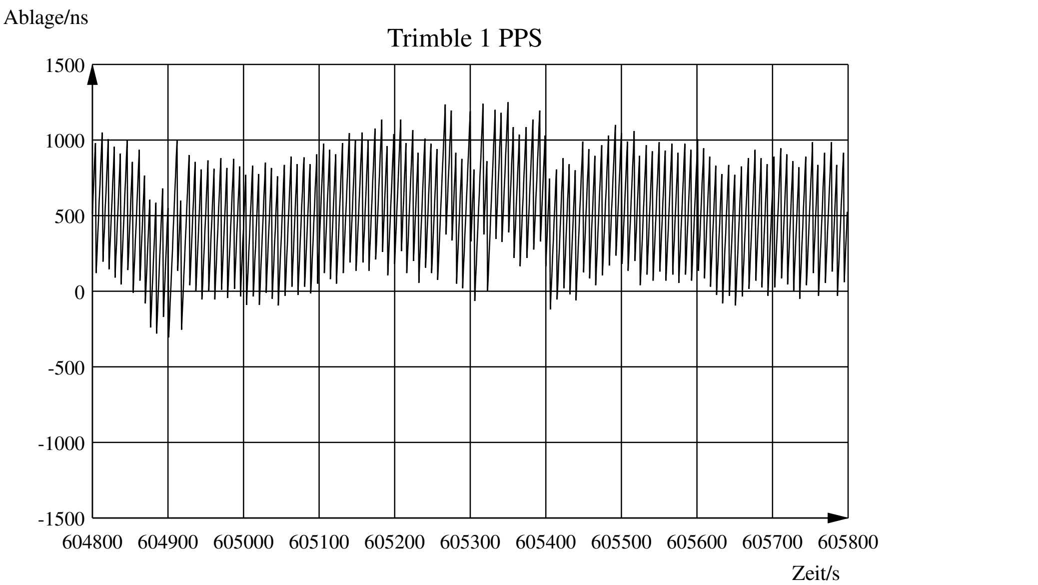

Abbildung A.2: Ablage des Empfängers Trimble für einen 1000 s–Ausschnitt

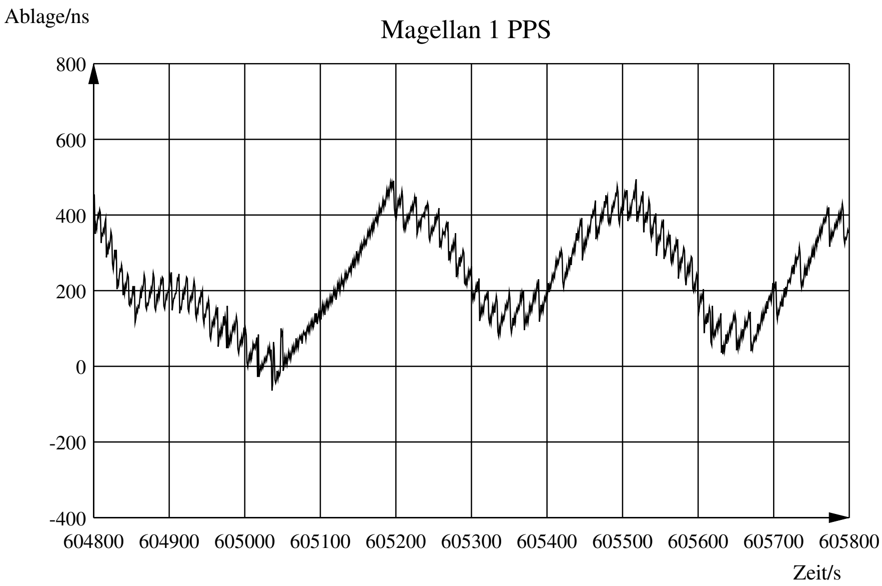

Abbildung A.3: Ablage des Empfängers Magellan für einen 1000 s–Ausschnitt

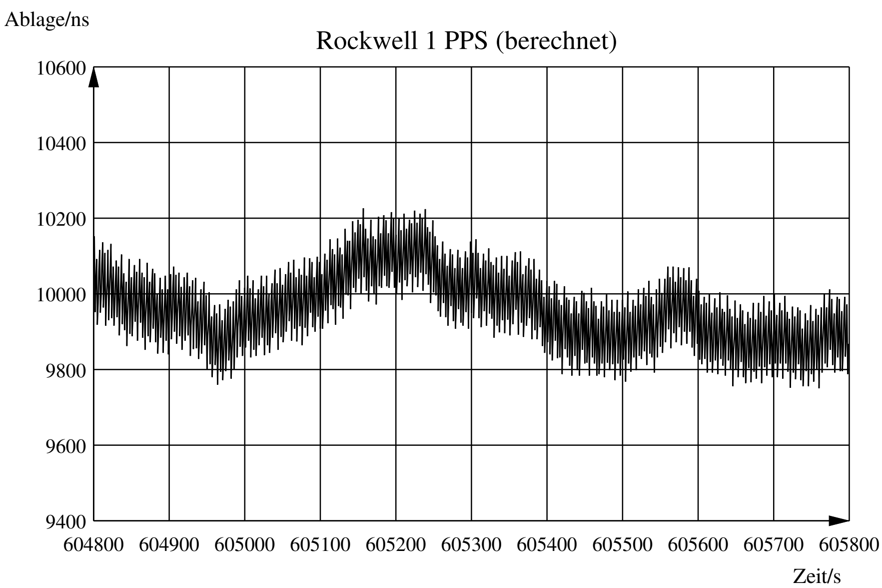

Abbildung A.4: Ablage des Empfängers Rockwell für einen 1000 s–Ausschnitt

A.2 Übersicht der 1 PPS-Pulse über die gesamte Meßdauer

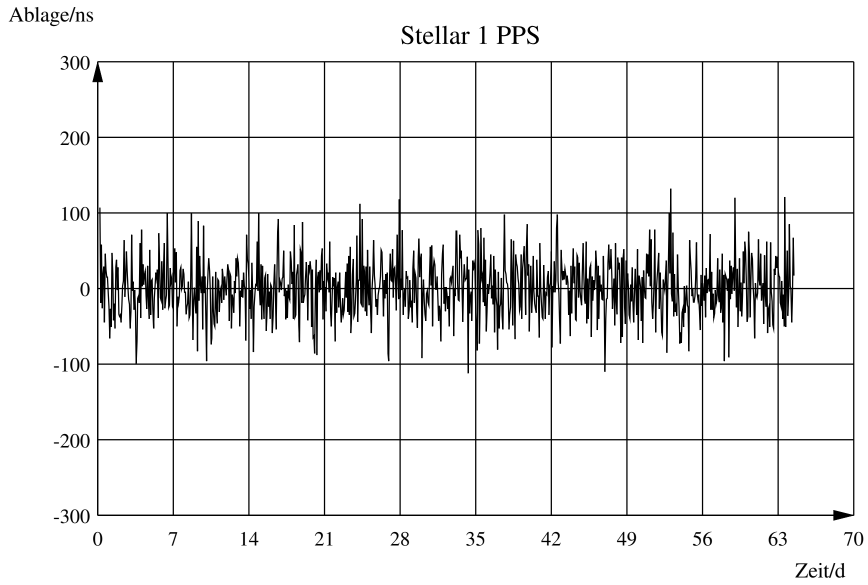

Abbildung A.5: Ablage Stellar 1 PPS gg. die Referenzzeit über die gesamte Meßdauer

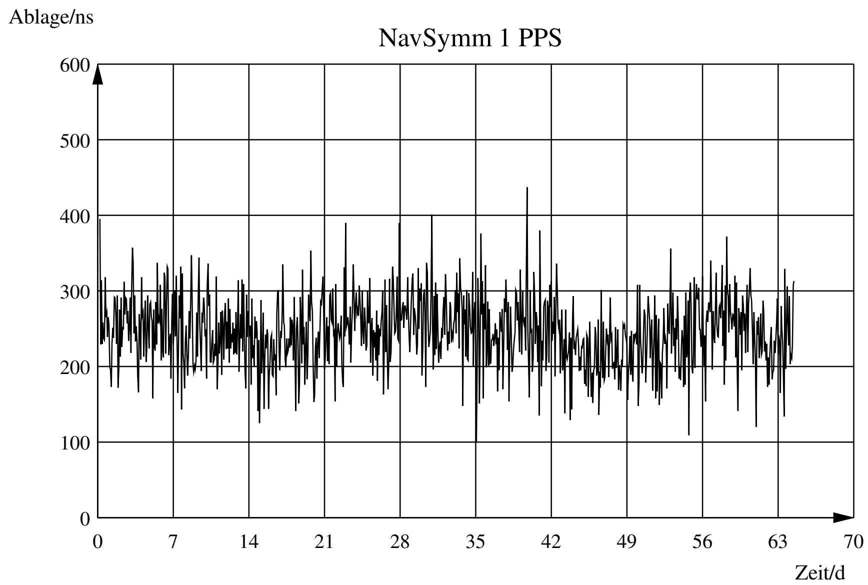

Abbildung A.6: Ablage NavSymm 1 PPS gg. die Referenzzeit über die gesamte Meßdauer

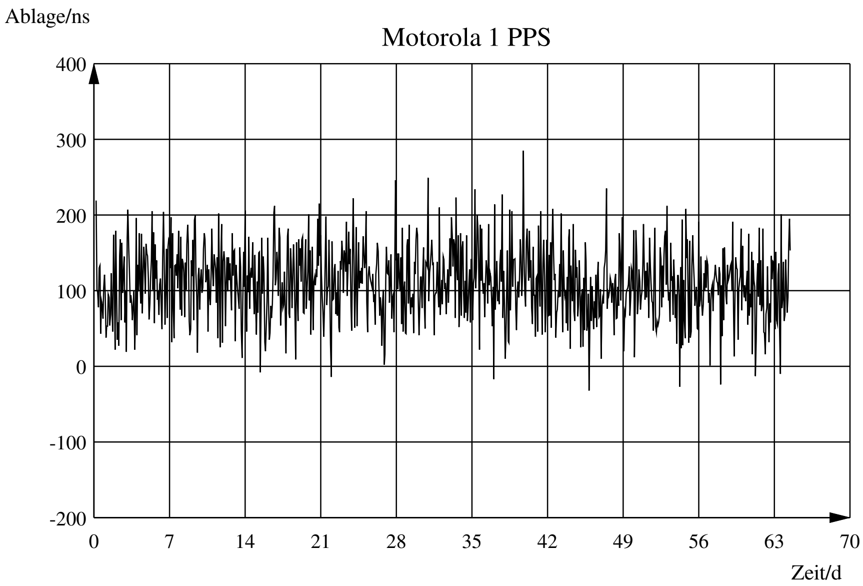

Abbildung A.7: Ablage Motorola 1 PPS gg. die Referenzzeit über die gesamte Meßdauer

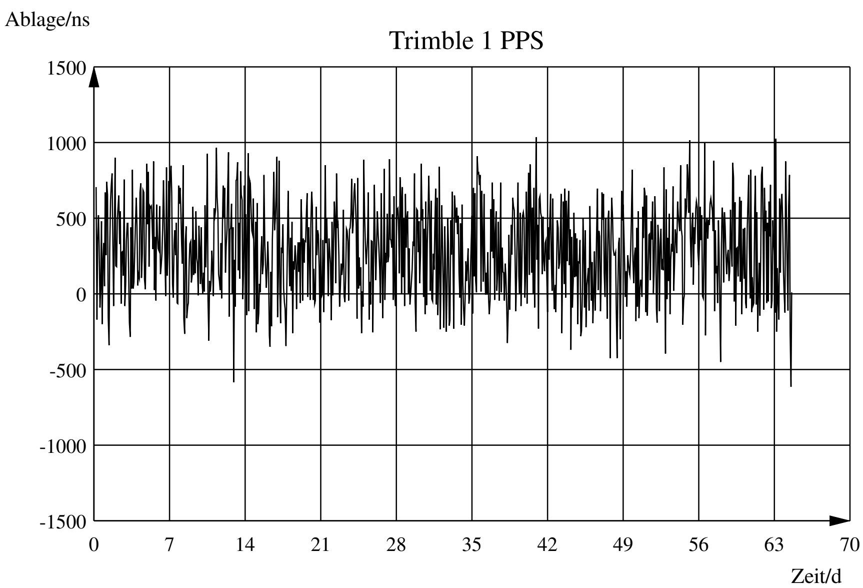

Abbildung A.8: Ablage Trimble 1 PPS gg. die Referenzzeit über die gesamte Meßdauer

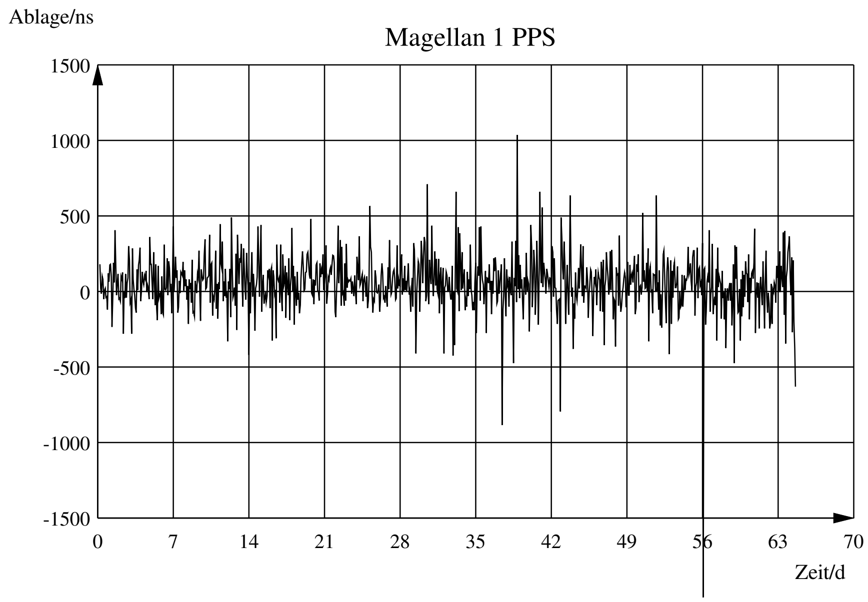

Abbildung A.9: Ablage Magellan 1 PPS gg. die Referenzzeit über die gesamte Meßdauer

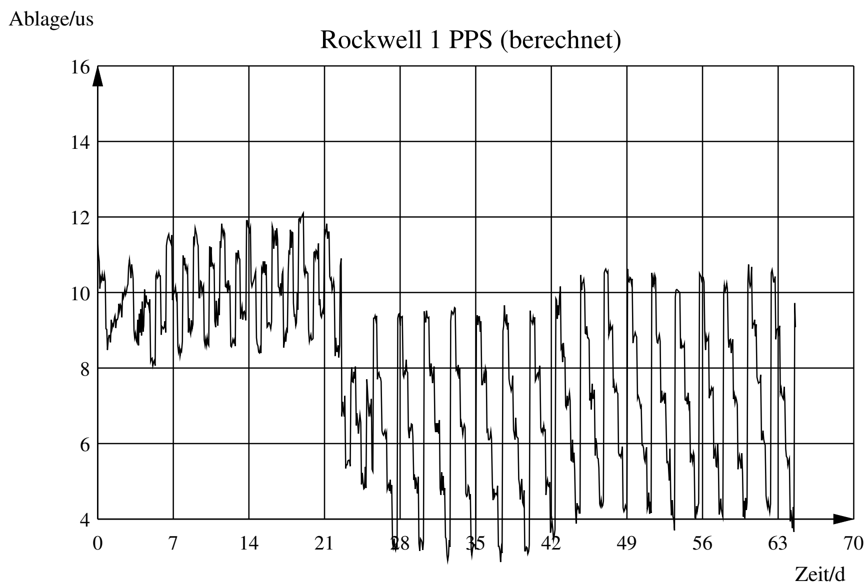

Abbildung A.10: Ablage Rockwell 1 PPS gg. die Referenzzeit über die gesamte Meßdauer

A.3 Ablagen 1 PPS–Pulse gegen die Referenzzeit

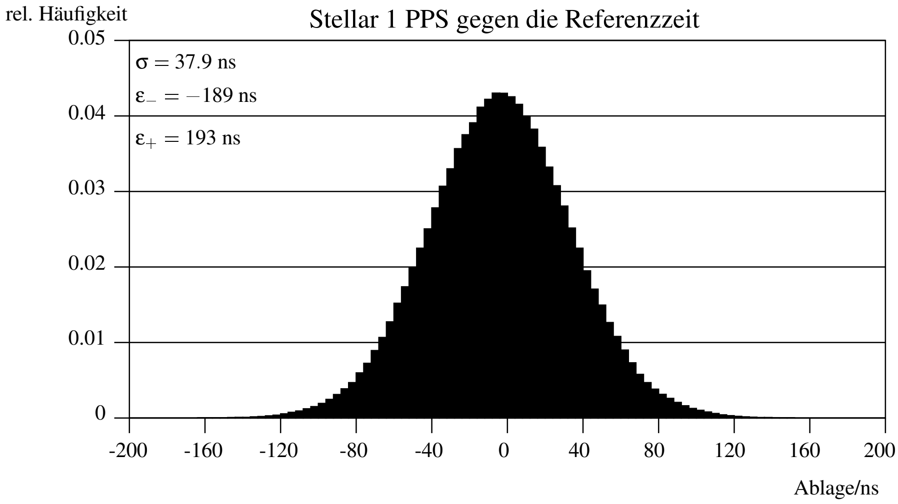

Abbildung A.11: Verteilungsfunktion Stellar 1 PPS gg. die Referenzzeit

Stellar 1 PPS gegen die Referenzzeit		
$\sigma = 37.9$ ns, $\varepsilon_- = -189$ ns, $\varepsilon_+ = 193$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -200	0	0 %
-200 … -180	38	$6.8 \cdot 10^{-6}$
-180 … -160	300	$5.4 \cdot 10^{-5}$
-160 … -140	1598	$2.9 \cdot 10^{-4}$
-140 … -120	6582	$1.2 \cdot 10^{-3}$
-120 … -100	28089	$5.0 \cdot 10^{-3}$
-100 … -80	90118	1.6 %
-80 … -60	255056	4.6 %
-60 … -40	559408	10.0 %
-40 … -20	921080	16.5 %
-20 … 0	1165924	20.9 %
0 … 20	1107608	19.8 %
20 … 40	781252	14.0 %
40 … 60	419420	7.5 %
60 … 80	171386	3.1 %
80 … 100	60023	1.1 %
100 … 120	18075	$3.2 \cdot 10^{-3}$
120 … 140	3792	$6.8 \cdot 10^{-4}$
140 … 160	661	$1.2 \cdot 10^{-4}$
160 … 180	187	$3.3 \cdot 10^{-5}$
180 … 200	35	$6.3 \cdot 10^{-6}$
> 200	0	0 %

Tabelle A.1: Verteilungsfunktion Stellar 1 PPS gg. die Referenzzeit

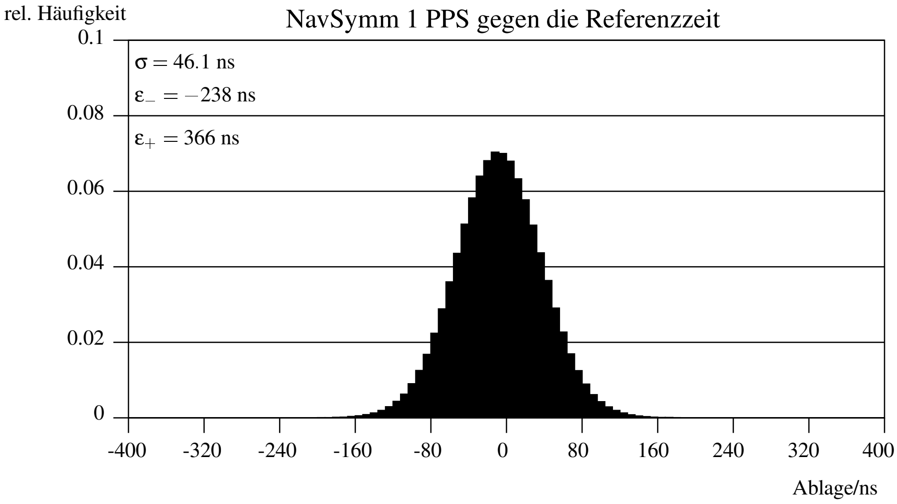

Abbildung A.12: Verteilungsfunktion NavSymm 1 PPS gg. die Referenzzeit

NavSymm 1 PPS gegen die Referenzzeit		
$\sigma = 46.1$ ns, $\varepsilon_- = -238$ ns, $\varepsilon_+ = 366$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -400	0	0 %
-400 … -360	0	0 %
-360 … -320	0	0 %
-320 … -280	0	0 %
-280 … -240	2	$3.6 \cdot 10^{-7}$
-240 … -200	451	$8.1 \cdot 10^{-5}$
-200 … -160	4746	$8.5 \cdot 10^{-4}$
-160 … -120	42566	$7.6 \cdot 10^{-3}$
-120 … -80	274379	4.9 %
-80 … -40	1018127	18.2 %
-40 … 0	1848060	33.1 %
0 … 40	1586002	28.4 %
40 … 80	657385	11.8 %
80 … 120	135663	2.4 %
120 … 160	17904	$3.2 \cdot 10^{-3}$
160 … 200	2034	$3.6 \cdot 10^{-4}$
200 … 240	272	$4.9 \cdot 10^{-5}$
240 … 280	71	$1.3 \cdot 10^{-5}$
280 … 320	89	$1.6 \cdot 10^{-5}$
320 … 360	36	$6.4 \cdot 10^{-6}$
360 … 400	0	0 %
> 400	0	0 %

Tabelle A.2: Verteilungsfunktion NavSymm 1 PPS gg. die Referenzzeit

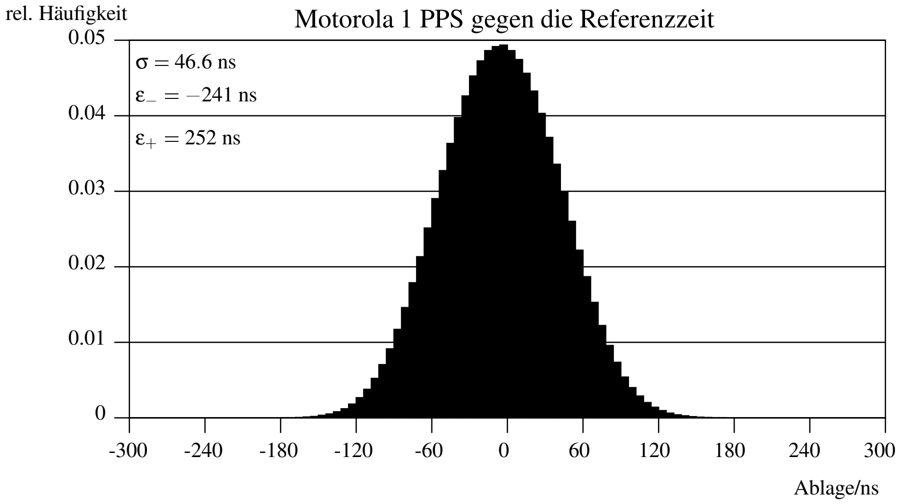

Abbildung A.13: Verteilungsfunktion Motorola 1 PPS gg. die Referenzzeit

Motorola 1 PPS gegen die Referenzzeit		
$\sigma = 46.6$ ns, $\varepsilon_- = -241$ ns, $\varepsilon_+ = 252$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -300	0	0 %
-300 … -270	0	0 %
-270 … -240	1	$1.8 \cdot 10^{-7}$
-240 … -210	18	$3.2 \cdot 10^{-6}$
-210 … -180	179	$3.2 \cdot 10^{-5}$
-180 … -150	2490	$4.5 \cdot 10^{-4}$
-150 … -120	26264	$4.7 \cdot 10^{-3}$
-120 … -90	156017	2.8 %
-90 … -60	507306	9.1 %
-60 … -30	1009208	18.1 %
-30 … 0	1340190	24.0 %
0 … 30	1259607	22.5 %
30 … 60	832536	14.9 %
60 … 90	353112	6.3 %
90 … 120	88230	1.6 %
120 … 150	13881	$2.5 \cdot 10^{-3}$
150 … 180	1379	$2.5 \cdot 10^{-4}$
180 … 210	166	$3.0 \cdot 10^{-5}$
210 … 240	35	$6.3 \cdot 10^{-6}$
240 … 270	3	$5.4 \cdot 10^{-7}$
270 … 300	0	0 %
> 300	0	0 %

Tabelle A.3: Verteilungsfunktion Motorola 1 PPS gg. die Referenzzeit

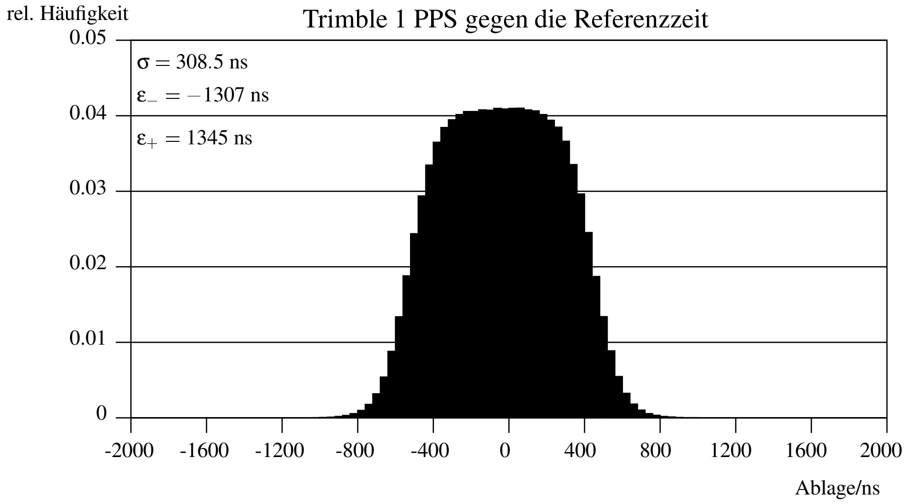

Abbildung A.14: Verteilungsfunktion Trimble 1 PPS gg. die Referenzzeit

Trimble 1 PPS gegen die Referenzzeit		
$\sigma = 308.5$ ns, $\varepsilon_- = -1307$ ns, $\varepsilon_+ = 1345$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -2000	0	0 %
-2000 ... -1800	0	0 %
-1800 ... -1600	0	0 %
-1600 ... -1400	0	0 %
-1400 ... -1200	17	$3.0 \cdot 10^{-6}$
-1200 ... -1000	381	$6.8 \cdot 10^{-5}$
-1000 ... -800	6884	$1.2 \cdot 10^{-3}$
-800 ... -600	113017	2.0 %
-600 ... -400	667929	11.9 %
-400 ... -200	1091083	19.5 %
-200 ... 0	1140435	20.4 %
0 ... 200	1138360	20.4 %
200 ... 400	993529	17.8 %
400 ... 600	397035	7.1 %
600 ... 800	39399	$7.0 \cdot 10^{-3}$
800 ... 1000	2386	$4.3 \cdot 10^{-4}$
1000 ... 1200	171	$3.1 \cdot 10^{-5}$
1200 ... 1400	6	$1.1 \cdot 10^{-6}$
1400 ... 1600	0	0 %
1600 ... 1800	0	0 %
1800 ... 2000	0	0 %
> 2000	0	0 %

Tabelle A.4: Verteilungsfunktion Trimble 1 PPS gg. die Referenzzeit

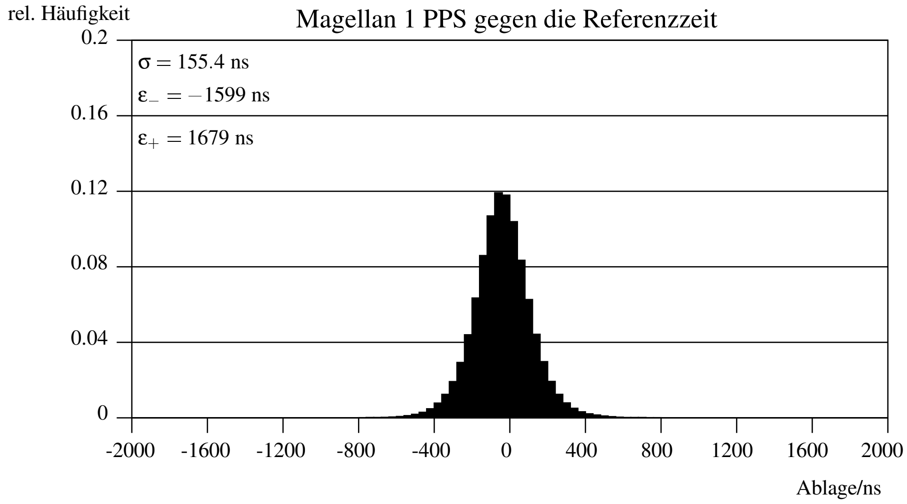

Abbildung A.15: Verteilungsfunktion Magellan 1 PPS gg. die Referenzzeit

Magellan 1 PPS gegen die Referenzzeit		
$\sigma = 155.4$ ns, $\varepsilon_- = -1599$ ns, $\varepsilon_+ = 1679$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -2000	0	0 %
-2000 ... -1800	0	0 %
-1800 ... -1600	19	$3.8 \cdot 10^{-6}$
-1600 ... -1400	47	$9.3 \cdot 10^{-6}$
-1400 ... -1200	56	$1.1 \cdot 10^{-5}$
-1200 ... -1000	669	$1.3 \cdot 10^{-4}$
-1000 ... -800	733	$1.5 \cdot 10^{-4}$
-800 ... -600	7401	$1.5 \cdot 10^{-3}$
-600 ... -400	58947	1.2 %
-400 ... -200	569280	11.3 %
-200 ... 0	2488093	49.4 %
0 ... 200	1634701	32.4 %
200 ... 400	242690	4.8 %
400 ... 600	29535	$5.9 \cdot 10^{-3}$
600 ... 800	4571	$9.1 \cdot 10^{-4}$
800 ... 1000	1787	$3.5 \cdot 10^{-4}$
1000 ... 1200	608	$1.2 \cdot 10^{-4}$
1200 ... 1400	11	$2.2 \cdot 10^{-6}$
1400 ... 1600	16	$3.2 \cdot 10^{-6}$
1600 ... 1800	5	$9.9 \cdot 10^{-7}$
1800 ... 2000	0	0 %
> 2000	0	0 %

Tabelle A.5: Verteilungsfunktion Magellan 1 PPS gg. die Referenzzeit

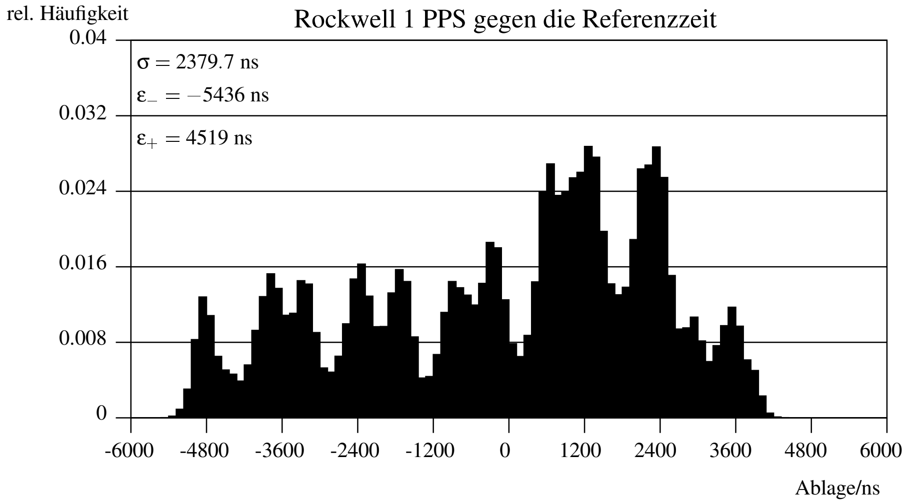

Abbildung A.16: Verteilungsfunktion Rockwell 1 PPS gg. die Referenzzeit

Rockwell 1 PPS gegen die Referenzzeit		
$\sigma = 2379.7$ ns, $\varepsilon_- = -5436$ ns, $\varepsilon_+ = 4519$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -6000	0	0 %
-6000 … -5400	132	$2.4 \cdot 10^{-5}$
-5400 … -4800	141304	2.5 %
-4800 … -4200	173209	3.1 %
-4200 … -3600	316952	5.7 %
-3600 … -3000	333838	6.0 %
-3000 … -2400	231117	4.1 %
-2400 … -1800	345328	6.2 %
-1800 … -1200	264442	4.7 %
-1200 … -600	330475	5.9 %
-600 … 0	421083	7.5 %
0 … 600	343523	6.1 %
600 … 1200	703191	12.6 %
1200 … 1800	577806	10.3 %
1800 … 2400	640572	11.5 %
2400 … 3000	392371	7.0 %
3000 … 3600	241920	4.3 %
3600 … 4200	132461	2.4 %
4200 … 4800	714	$1.3 \cdot 10^{-4}$
4800 … 5400	0	0 %
5400 … 6000	0	0 %
> 6000	0	0 %

Tabelle A.6: Verteilungsfunktion Rockwell 1 PPS gg. die Referenzzeit

A.4 Ablagen 1 PPS–Pulse für verschiedene Mittelungszeiten

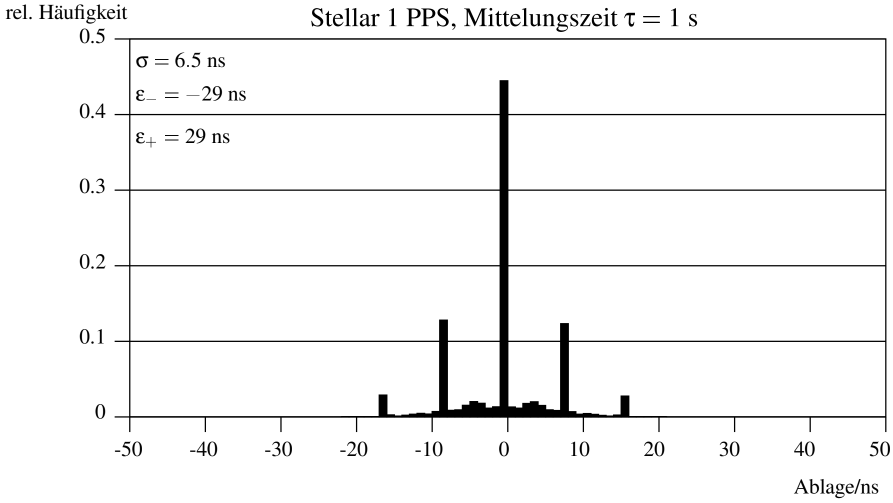

Abbildung A.17: Verteilungsfunktion Stellar 1 PPS für Mittelungszeit $\tau = 1$ s

Stellar 1 PPS, Mittelungszeit $\tau = 1$ s		
$\sigma = 6.5$ ns, $\varepsilon_- = -29$ ns, $\varepsilon_+ = 29$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -50	0	0 %
-50 … -45	0	0 %
-45 … -40	0	0 %
-40 … -35	0	0 %
-35 … -30	0	0 %
-30 … -25	300	$5.4 \cdot 10^{-5}$
-25 … -20	3868	$6.9 \cdot 10^{-4}$
-20 … -15	183383	3.3 %
-15 … -10	87243	1.6 %
-10 … -5	944827	16.9 %
-5 … 0	2842052	50.8 %
0 … 5	436147	7.8 %
5 … 10	850515	15.2 %
10 … 15	77447	1.4 %
15 … 20	162241	2.9 %
20 … 25	2436	$4.4 \cdot 10^{-4}$
25 … 30	152	$2.7 \cdot 10^{-5}$
30 … 35	0	0 %
35 … 40	0	0 %
40 … 45	0	0 %
45 … 50	0	0 %
> 50	0	0 %

Tabelle A.7: Verteilungsfunktion Stellar 1 PPS für Mittelungszeit $\tau = 1$ s

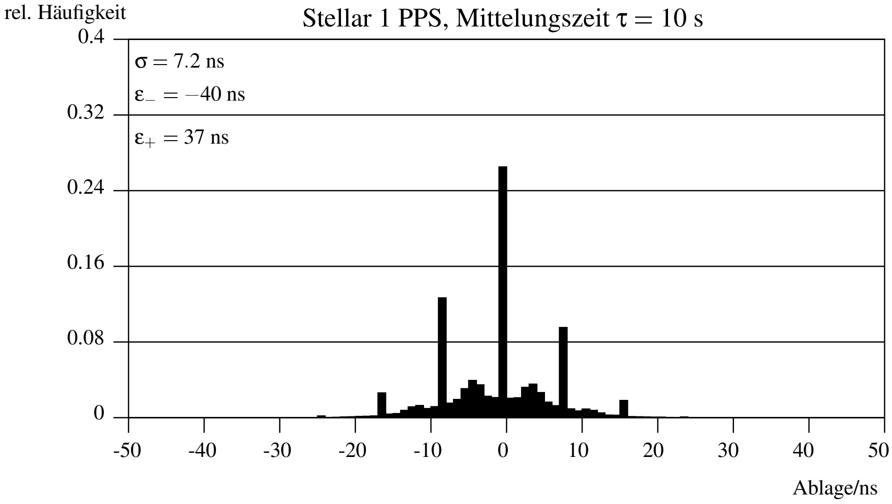

Abbildung A.18: Verteilungsfunktion Stellar 1 PPS für Mittelungszeit $\tau = 10$ s

Stellar 1 PPS, Mittelungszeit $\tau = 10$ s		
$\sigma = 7.2$ ns, $\varepsilon_- = -40$ ns, $\varepsilon_+ = 37$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -50	0	0 %
-50 … -45	0	0 %
-45 … -40	2	$3.6 \cdot 10^{-7}$
-40 … -35	20	$3.6 \cdot 10^{-6}$
-35 … -30	241	$4.3 \cdot 10^{-5}$
-30 … -25	2481	$4.4 \cdot 10^{-4}$
-25 … -20	24261	$4.3 \cdot 10^{-3}$
-20 … -15	194000	3.5 %
-15 … -10	257235	4.6 %
-10 … -5	1138429	20.4 %
-5 … 0	2142295	38.3 %
0 … 5	758696	13.6 %
5 … 10	788426	14.1 %
10 … 15	152003	2.7 %
15 … 20	121609	2.2 %
20 … 25	9843	$1.8 \cdot 10^{-3}$
25 … 30	850	$1.5 \cdot 10^{-4}$
30 … 35	65	$1.2 \cdot 10^{-5}$
35 … 40	6	$1.1 \cdot 10^{-6}$
40 … 45	0	0 %
45 … 50	0	0 %
> 50	0	0 %

Tabelle A.8: Verteilungsfunktion Stellar 1 PPS für Mittelungszeit $\tau = 10$ s

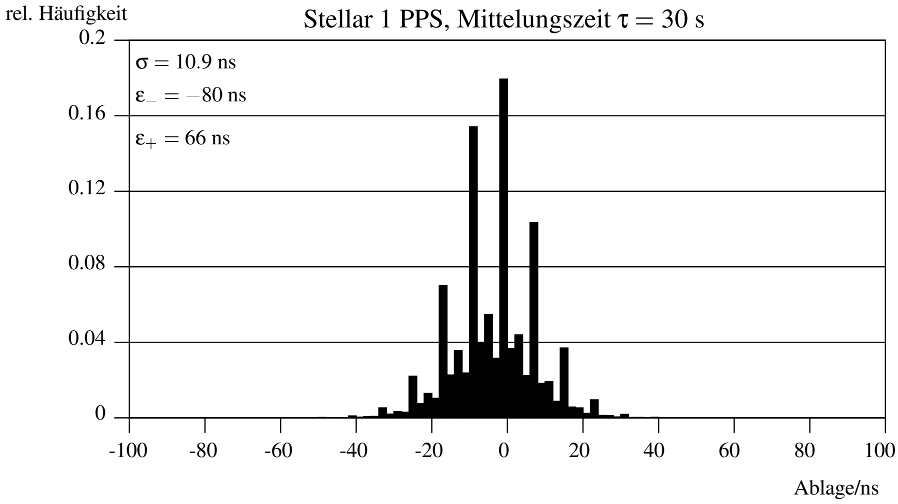

Abbildung A.19: Verteilungsfunktion Stellar 1 PPS für Mittelungszeit $\tau = 30$ s

Stellar 1 PPS, Mittelungszeit $\tau = 30$ s		
$\sigma = 10.9$ ns, $\varepsilon_- = -80$ ns, $\varepsilon_+ = 66$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -100	0	0 %
-100 … -90	0	0 %
-90 … -80	4	$7.2 \cdot 10^{-7}$
-80 … -70	26	$4.7 \cdot 10^{-6}$
-70 … -60	130	$2.3 \cdot 10^{-5}$
-60 … -50	522	$9.3 \cdot 10^{-5}$
-50 … -40	8498	$1.5 \cdot 10^{-3}$
-40 … -30	50556	$9.0 \cdot 10^{-3}$
-30 … -20	272950	4.9 %
-20 … -10	908752	16.3 %
-10 … 0	2569011	46.0 %
0 … 10	1256370	22.5 %
10 … 20	424592	7.6 %
20 … 30	83493	1.5 %
30 … 40	14639	$2.6 \cdot 10^{-3}$
40 … 50	757	$1.4 \cdot 10^{-4}$
50 … 60	121	$2.2 \cdot 10^{-5}$
60 … 70	13	$2.3 \cdot 10^{-6}$
70 … 80	0	0 %
80 … 90	0	0 %
90 … 100	0	0 %
> 100	0	0 %

Tabelle A.9: Verteilungsfunktion Stellar 1 PPS für Mittelungszeit $\tau = 30$ s

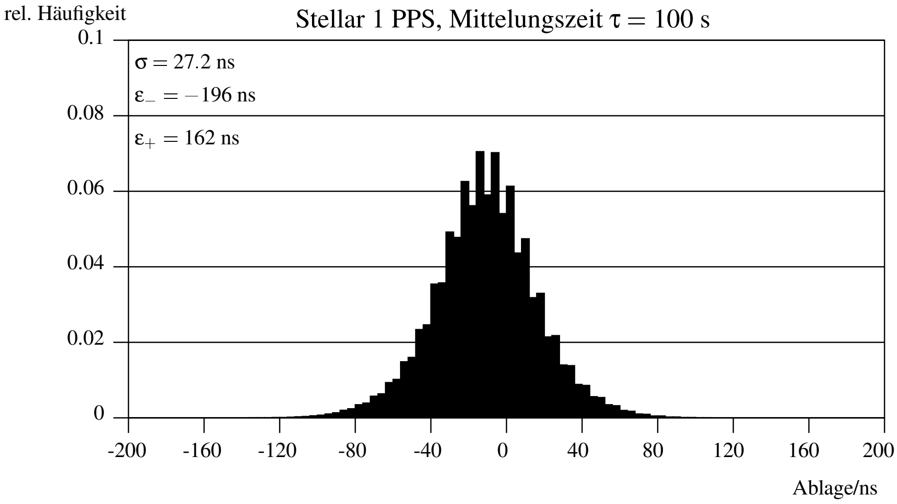

Abbildung A.20: Verteilungsfunktion Stellar 1 PPS für Mittelungszeit $\tau = 100$ s

Stellar 1 PPS, Mittelungszeit $\tau = 100$ s		
$\sigma = 27.2$ ns, $\varepsilon_- = -196$ ns, $\varepsilon_+ = 162$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -200	0	0 %
-200 … -180	36	$6.4 \cdot 10^{-6}$
-180 … -160	62	$1.1 \cdot 10^{-5}$
-160 … -140	531	$9.5 \cdot 10^{-5}$
-140 … -120	1723	$3.1 \cdot 10^{-4}$
-120 … -100	8837	$1.6 \cdot 10^{-3}$
-100 … -80	42718	$7.6 \cdot 10^{-3}$
-80 … -60	161730	2.9 %
-60 … -40	498457	8.9 %
-40 … -20	1290162	23.1 %
-20 … 0	1733623	31.0 %
0 … 20	1214922	21.7 %
20 … 40	446981	8.0 %
40 … 60	147185	2.6 %
60 … 80	34973	$6.3 \cdot 10^{-3}$
80 … 100	6909	$1.2 \cdot 10^{-3}$
100 … 120	1202	$2.2 \cdot 10^{-4}$
120 … 140	288	$5.2 \cdot 10^{-5}$
140 … 160	25	$4.5 \cdot 10^{-6}$
160 … 180	0	0 %
180 … 200	0	0 %
> 200	0	0 %

Tabelle A.10: Verteilungsfunktion Stellar 1 PPS für Mittelungszeit $\tau = 100$ s

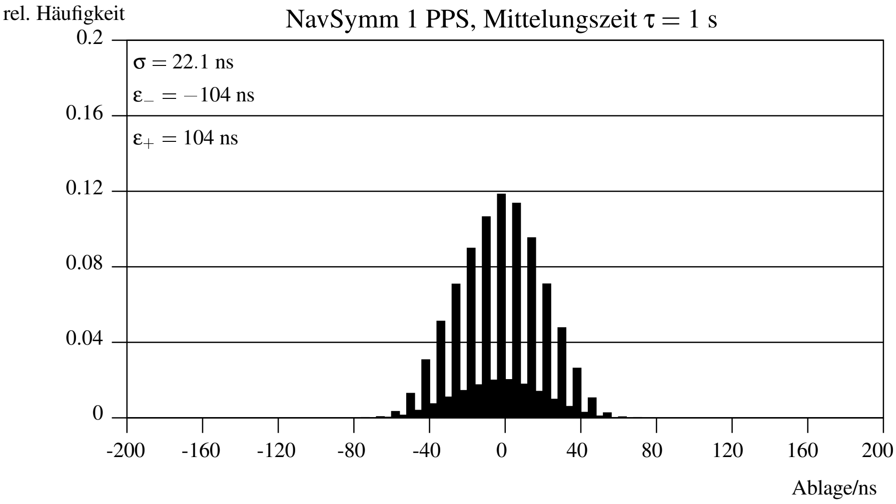

Abbildung A.21: Verteilungsfunktion NavSymm 1 PPS für Mittelungszeit $\tau = 1$ s

NavSymm 1 PPS, Mittelungszeit $\tau = 1$ s		
$\sigma = 22.1$ ns, $\varepsilon_- = -104$ ns, $\varepsilon_+ = 104$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -200	0	0 %
-200 … -180	0	0 %
-180 … -160	0	0 %
-160 … -140	0	0 %
-140 … -120	0	0 %
-120 … -100	2	$3.6 \cdot 10^{-7}$
-100 … -80	562	$1.0 \cdot 10^{-4}$
-80 … -60	6173	$1.1 \cdot 10^{-3}$
-60 … -40	292404	5.2 %
-40 … -20	865158	15.5 %
-20 … 0	1967673	35.2 %
0 … 20	1457849	26.1 %
20 … 40	898044	16.1 %
40 … 60	95895	1.7 %
60 … 80	3805	$6.8 \cdot 10^{-4}$
80 … 100	183	$3.3 \cdot 10^{-5}$
100 … 120	3	$5.4 \cdot 10^{-7}$
120 … 140	0	0 %
140 … 160	0	0 %
160 … 180	0	0 %
180 … 200	0	0 %
> 200	0	0 %

Tabelle A.11: Verteilungsfunktion NavSymm 1 PPS für Mittelungszeit $\tau = 1$ s

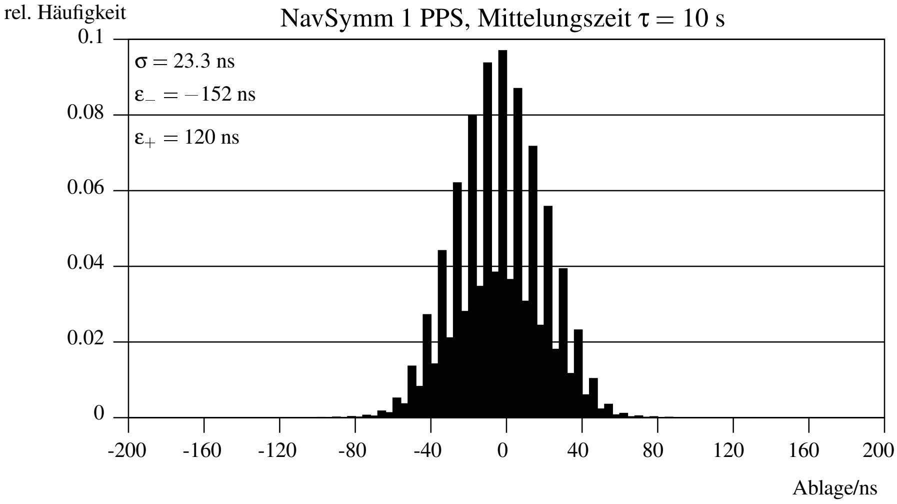

Abbildung A.22: Verteilungsfunktion NavSymm 1 PPS für Mittelungszeit $\tau = 10$ s

NavSymm 1 PPS, Mittelungszeit $\tau = 10$ s		
$\sigma = 23.3$ ns, $\varepsilon_- = -152$ ns, $\varepsilon_+ = 120$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -200	0	0 %
-200 … -180	0	0 %
-180 … -160	0	0 %
-160 … -140	4	$7.2 \cdot 10^{-7}$
-140 … -120	13	$2.3 \cdot 10^{-6}$
-120 … -100	351	$6.3 \cdot 10^{-5}$
-100 … -80	3400	$6.1 \cdot 10^{-4}$
-80 … -60	23963	$4.3 \cdot 10^{-3}$
-60 … -40	323688	5.8 %
-40 … -20	947621	17.0 %
-20 … 0	1920233	34.4 %
0 … 20	1399215	25.0 %
20 … 40	827476	14.8 %
40 … 60	127980	2.3 %
60 … 80	12184	$2.2 \cdot 10^{-3}$
80 … 100	1224	$2.2 \cdot 10^{-4}$
100 … 120	115	$2.1 \cdot 10^{-5}$
120 … 140	0	0 %
140 … 160	0	0 %
160 … 180	0	0 %
180 … 200	0	0 %
> 200	0	0 %

Tabelle A.12: Verteilungsfunktion NavSymm 1 PPS für Mittelungszeit $\tau = 10$ s

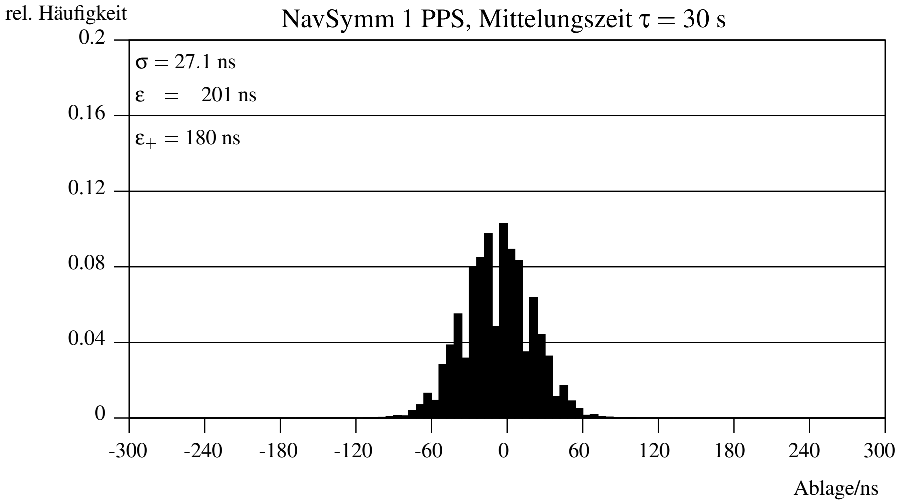

Abbildung A.23: Verteilungsfunktion NavSymm 1 PPS für Mittelungszeit $\tau = 30$ s

NavSymm 1 PPS, Mittelungszeit $\tau = 30$ s		
$\sigma = 27.1$ ns, $\varepsilon_- = -201$ ns, $\varepsilon_+ = 180$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -300	0	0 %
-300 … -270	0	0 %
-270 … -240	0	0 %
-240 … -210	0	0 %
-210 … -180	19	$3.4 \cdot 10^{-6}$
-180 … -150	33	$5.9 \cdot 10^{-6}$
-150 … -120	512	$9.2 \cdot 10^{-5}$
-120 … -90	7865	$1.4 \cdot 10^{-3}$
-90 … -60	148100	2.7 %
-60 … -30	910479	16.3 %
-30 … 0	2308900	41.3 %
0 … 30	1761629	31.5 %
30 … 60	421267	7.5 %
60 … 90	26049	$4.7 \cdot 10^{-3}$
90 … 120	2075	$3.7 \cdot 10^{-4}$
120 … 150	192	$3.4 \cdot 10^{-5}$
150 … 180	19	$3.4 \cdot 10^{-6}$
180 … 210	0	0 %
210 … 240	0	0 %
240 … 270	0	0 %
270 … 300	0	0 %
> 300	0	0 %

Tabelle A.13: Verteilungsfunktion NavSymm 1 PPS für Mittelungszeit $\tau = 30$ s

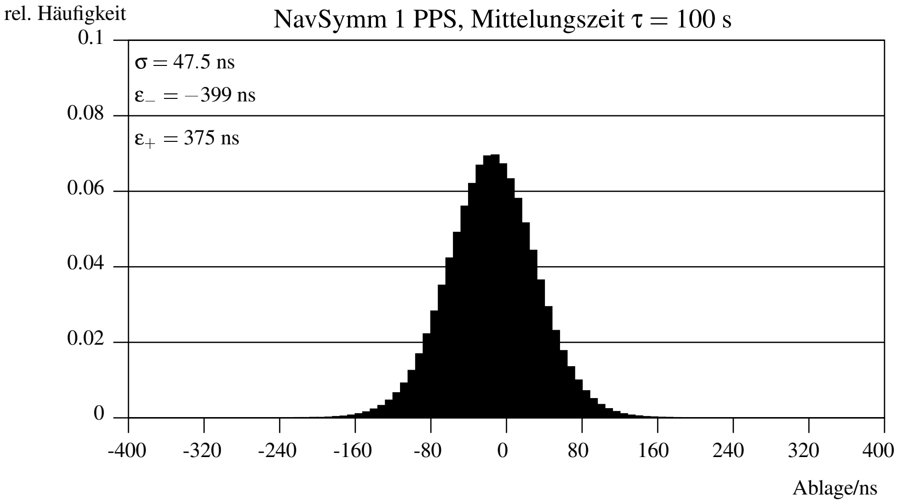

Abbildung A.24: Verteilungsfunktion NavSymm 1 PPS für Mittelungszeit $\tau = 100$ s

NavSymm 1 PPS, Mittelungszeit $\tau = 100$ s		
$\sigma = 47.5$ ns, $\varepsilon_- = -399$ ns, $\varepsilon_+ = 375$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -400	2	$3.6 \cdot 10^{-7}$
-400 … -360	3	$5.4 \cdot 10^{-7}$
-360 … -320	40	$7.2 \cdot 10^{-6}$
-320 … -280	85	$1.5 \cdot 10^{-5}$
-280 … -240	307	$5.5 \cdot 10^{-5}$
-240 … -200	1505	$2.7 \cdot 10^{-4}$
-200 … -160	10439	$1.9 \cdot 10^{-3}$
-160 … -120	72484	1.3 %
-120 … -80	377419	6.8 %
-80 … -40	1178188	21.1 %
-40 … 0	1872740	33.5 %
0 … 40	1418298	25.4 %
40 … 80	524536	9.4 %
80 … 120	111031	2.0 %
120 … 160	16238	$2.9 \cdot 10^{-3}$
160 … 200	2242	$4.0 \cdot 10^{-4}$
200 … 240	325	$5.8 \cdot 10^{-5}$
240 … 280	59	$1.1 \cdot 10^{-5}$
280 … 320	59	$1.1 \cdot 10^{-5}$
320 … 360	18	$3.2 \cdot 10^{-6}$
360 … 400	1	$1.8 \cdot 10^{-7}$
> 400	0	0 %

Tabelle A.14: Verteilungsfunktion NavSymm 1 PPS für Mittelungszeit $\tau = 100$ s

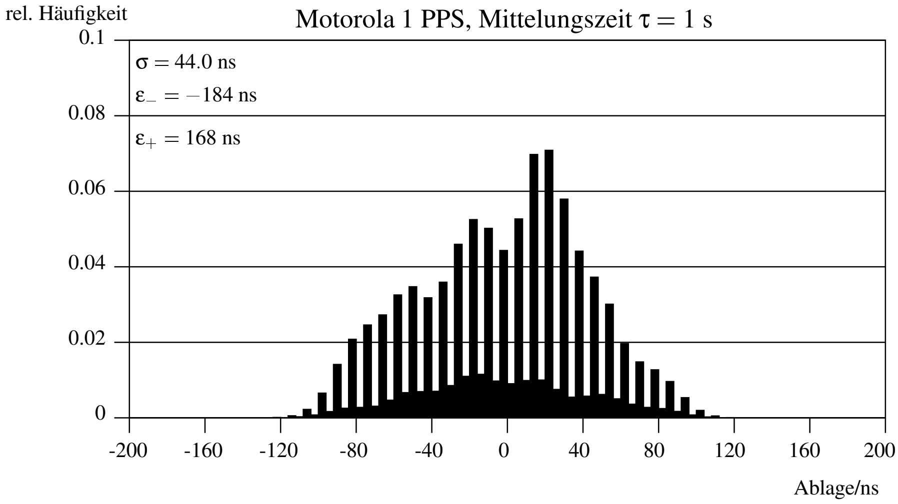

Abbildung A.25: Verteilungsfunktion Motorola 1 PPS für Mittelungszeit $\tau = 1$ s

Motorola 1 PPS, Mittelungszeit $\tau = 1$ s		
$\sigma = 44.0$ ns, $\varepsilon_- = -184$ ns, $\varepsilon_+ = 168$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -200	0	0 %
-200 ... -180	1	$1.8 \cdot 10^{-7}$
-180 ... -160	3	$5.4 \cdot 10^{-7}$
-160 ... -140	14	$2.5 \cdot 10^{-6}$
-140 ... -120	828	$1.5 \cdot 10^{-4}$
-120 ... -100	21797	$3.9 \cdot 10^{-3}$
-100 ... -80	255884	4.6 %
-80 ... -60	349139	6.2 %
-60 ... -40	630244	11.3 %
-40 ... -20	606592	10.9 %
-20 ... 0	941008	16.8 %
0 ... 20	846757	15.1 %
20 ... 40	1039615	18.6 %
40 ... 60	471450	8.4 %
60 ... 80	299747	5.4 %
80 ... 100	111386	2.0 %
100 ... 120	16002	$2.9 \cdot 10^{-3}$
120 ... 140	114	$2.0 \cdot 10^{-5}$
140 ... 160	9	$1.6 \cdot 10^{-6}$
160 ... 180	1	$1.8 \cdot 10^{-7}$
180 ... 200	0	0 %
> 200	0	0 %

Tabelle A.15: Verteilungsfunktion Motorola 1 PPS für Mittelungszeit $\tau = 1$ s

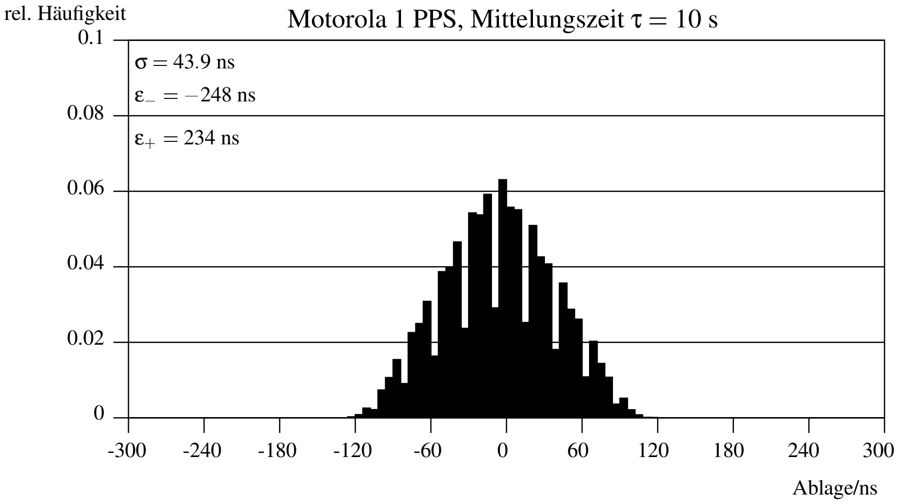

Abbildung A.26: Verteilungsfunktion Motorola 1 PPS für Mittelungszeit $\tau = 10$ s

Motorola 1 PPS, Mittelungszeit $\tau = 10$ s		
$\sigma = 43.9$ ns, $\varepsilon_- = -248$ ns, $\varepsilon_+ = 234$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -300	0	0 %
-300 … -270	0	0 %
-270 … -240	4	$7.2 \cdot 10^{-7}$
-240 … -210	5	$8.9 \cdot 10^{-7}$
-210 … -180	13	$2.3 \cdot 10^{-6}$
-180 … -150	38	$6.8 \cdot 10^{-6}$
-150 … -120	2011	$3.6 \cdot 10^{-4}$
-120 … -90	133014	2.4 %
-90 … -60	575678	10.3 %
-60 … -30	924701	16.5 %
-30 … 0	1450569	25.9 %
0 … 30	1284704	23.0 %
30 … 60	836240	15.0 %
60 … 90	335844	6.0 %
90 … 120	47386	$8.5 \cdot 10^{-3}$
120 … 150	199	$3.6 \cdot 10^{-5}$
150 … 180	23	$4.1 \cdot 10^{-6}$
180 … 210	8	$1.4 \cdot 10^{-6}$
210 … 240	5	$8.9 \cdot 10^{-7}$
240 … 270	0	0 %
270 … 300	0	0 %
> 300	0	0 %

Tabelle A.16: Verteilungsfunktion Motorola 1 PPS für Mittelungszeit $\tau = 10$ s

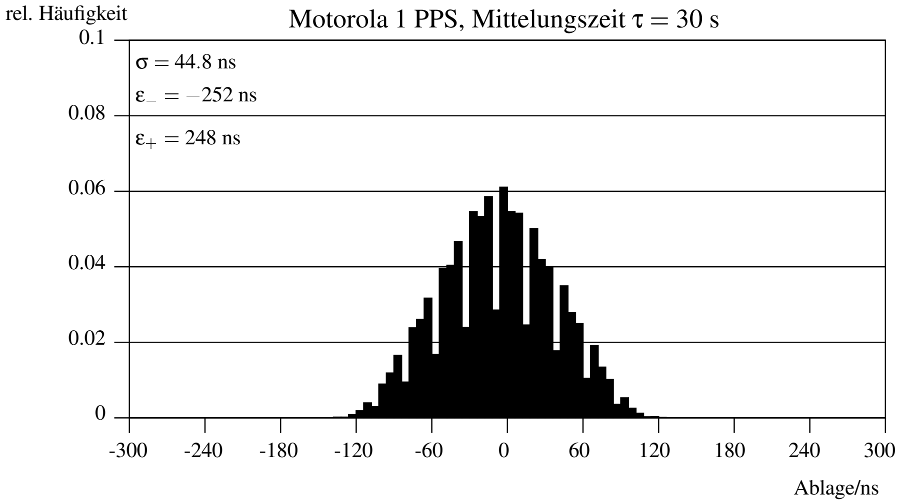

Abbildung A.27: Verteilungsfunktion Motorola 1 PPS für Mittelungszeit $\tau = 30$ s

Motorola 1 PPS, Mittelungszeit $\tau = 30$ s		
$\sigma = 44.8$ ns, $\varepsilon_- = -252$ ns, $\varepsilon_+ = 248$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -300	0	0 %
-300 … -270	0	0 %
-270 … -240	5	$8.9 \cdot 10^{-7}$
-240 … -210	13	$2.3 \cdot 10^{-6}$
-210 … -180	26	$4.7 \cdot 10^{-6}$
-180 … -150	144	$2.6 \cdot 10^{-5}$
-150 … -120	7371	$1.3 \cdot 10^{-3}$
-120 … -90	165629	3.0 %
-90 … -60	602245	10.8 %
-60 … -30	935448	16.7 %
-30 … 0	1432073	25.6 %
0 … 30	1261238	22.6 %
30 … 60	813926	14.6 %
60 … 90	317238	5.7 %
90 … 120	54145	$9.7 \cdot 10^{-3}$
120 … 150	860	$1.5 \cdot 10^{-4}$
150 … 180	41	$7.3 \cdot 10^{-6}$
180 … 210	7	$1.3 \cdot 10^{-6}$
210 … 240	4	$7.2 \cdot 10^{-7}$
240 … 270	1	$1.8 \cdot 10^{-7}$
270 … 300	0	0 %
> 300	0	0 %

Tabelle A.17: Verteilungsfunktion Motorola 1 PPS für Mittelungszeit $\tau = 30$ s

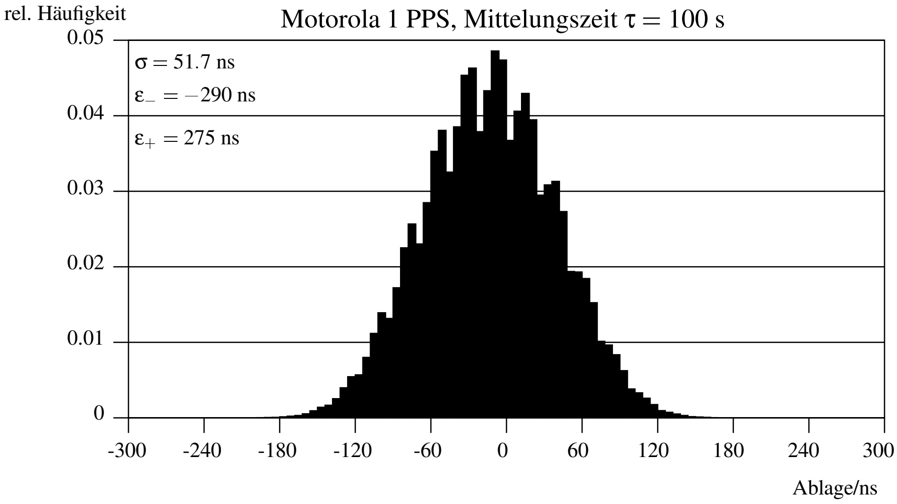

Abbildung A.28: Verteilungsfunktion Motorola 1 PPS für Mittelungszeit $\tau = 100$ s

Motorola 1 PPS, Mittelungszeit $\tau = 100$ s		
$\sigma = 51.7$ ns, $\varepsilon_- = -290$ ns, $\varepsilon_+ = 275$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -300	0	0 %
-300 ... -270	7	$1.3 \cdot 10^{-6}$
-270 ... -240	20	$3.6 \cdot 10^{-6}$
-240 ... -210	70	$1.3 \cdot 10^{-5}$
-210 ... -180	1005	$1.8 \cdot 10^{-4}$
-180 ... -150	12324	$2.2 \cdot 10^{-3}$
-150 ... -120	84414	1.5 %
-120 ... -90	290773	5.2 %
-90 ... -60	653737	11.7 %
-60 ... -30	1061169	19.0 %
-30 ... 0	1249536	22.4 %
0 ... 30	1058419	18.9 %
30 ... 60	716907	12.8 %
60 ... 90	345936	6.2 %
90 ... 120	99450	1.8 %
120 ... 150	15120	$2.7 \cdot 10^{-3}$
150 ... 180	1383	$2.5 \cdot 10^{-4}$
180 ... 210	65	$1.2 \cdot 10^{-5}$
210 ... 240	5	$8.9 \cdot 10^{-7}$
240 ... 270	4	$7.2 \cdot 10^{-7}$
270 ... 300	0	0 %
> 300	0	0 %

Tabelle A.18: Verteilungsfunktion Motorola 1 PPS für Mittelungszeit $\tau = 100$ s

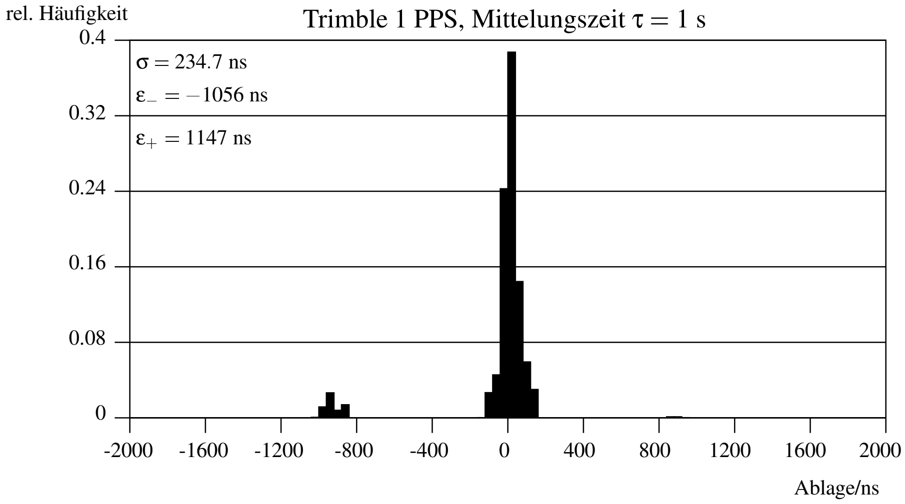

Abbildung A.29: Verteilungsfunktion Trimble 1 PPS für Mittelungszeit $\tau = 1$ s

Trimble 1 PPS, Mittelungszeit $\tau = 1$ s		
$\sigma = 234.7$ ns, $\varepsilon_- = -1056$ ns, $\varepsilon_+ = 1147$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -2000	0	0 %
-2000 ... -1800	0	0 %
-1800 ... -1600	0	0 %
-1600 ... -1400	0	0 %
-1400 ... -1200	0	0 %
-1200 ... -1000	3044	$5.4 \cdot 10^{-4}$
-1000 ... -800	337192	6.0 %
-800 ... -600	0	0 %
-600 ... -400	0	0 %
-400 ... -200	0	0 %
-200 ... 0	1763212	31.5 %
0 ... 200	3473538	62.1 %
200 ... 400	0	0 %
400 ... 600	0	0 %
600 ... 800	0	0 %
800 ... 1000	13482	$2.4 \cdot 10^{-3}$
1000 ... 1200	143	$2.6 \cdot 10^{-5}$
1200 ... 1400	0	0 %
1400 ... 1600	0	0 %
1600 ... 1800	0	0 %
1800 ... 2000	0	0 %
> 2000	0	0 %

Tabelle A.19: Verteilungsfunktion Trimble 1 PPS für Mittelungszeit $\tau = 1$ s

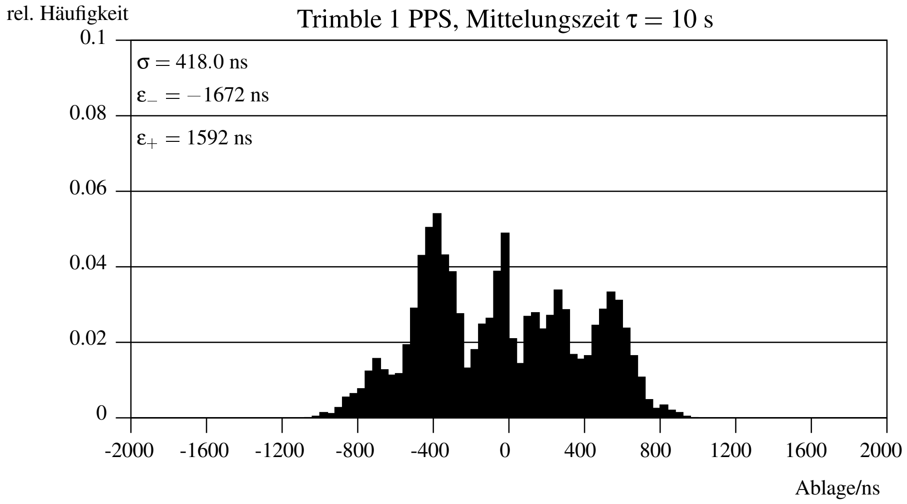

Abbildung A.30: Verteilungsfunktion Trimble 1 PPS für Mittelungszeit $\tau = 10$ s

Trimble 1 PPS, Mittelungszeit $\tau = 10$ s		
$\sigma = 418.0$ ns, $\varepsilon_{-} = -1672$ ns, $\varepsilon_{+} = 1592$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -2000	0	0 %
-2000 … -1800	0	0 %
-1800 … -1600	4	$7.2 \cdot 10^{-7}$
-1600 … -1400	17	$3.0 \cdot 10^{-6}$
-1400 … -1200	156	$2.8 \cdot 10^{-5}$
-1200 … -1000	3035	$5.4 \cdot 10^{-4}$
-1000 … -800	95927	1.7 %
-800 … -600	334121	6.0 %
-600 … -400	858500	15.4 %
-400 … -200	987332	17.7 %
-200 … 0	877462	15.7 %
0 … 200	635046	11.4 %
200 … 400	681602	12.2 %
400 … 600	750218	13.4 %
600 … 800	325868	5.8 %
800 … 1000	40825	$7.3 \cdot 10^{-3}$
1000 … 1200	270	$4.8 \cdot 10^{-5}$
1200 … 1400	75	$1.3 \cdot 10^{-5}$
1400 … 1600	4	$7.2 \cdot 10^{-7}$
1600 … 1800	0	0 %
1800 … 2000	0	0 %
> 2000	0	0 %

Tabelle A.20: Verteilungsfunktion Trimble 1 PPS für Mittelungszeit $\tau = 10$ s

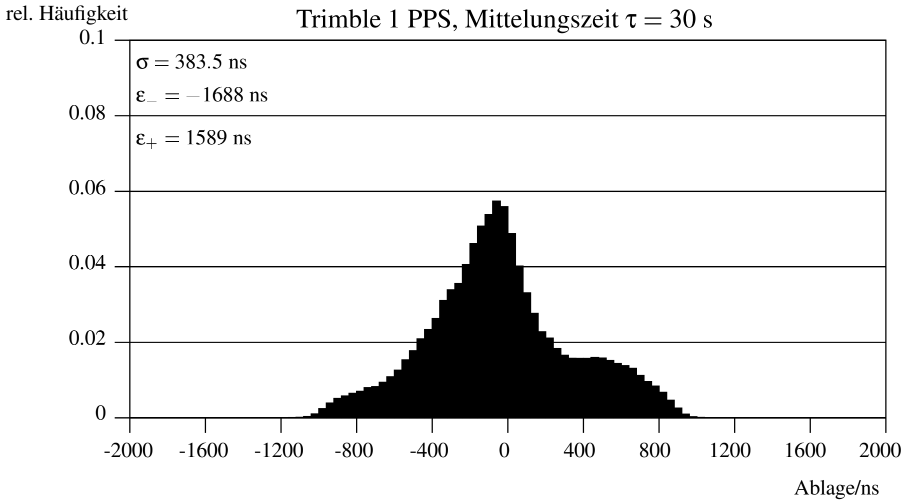

Abbildung A.31: Verteilungsfunktion Trimble 1 PPS für Mittelungszeit $\tau = 30$ s

Trimble 1 PPS, Mittelungszeit $\tau = 30$ s		
$\sigma = 383.5$ ns, $\varepsilon_- = -1688$ ns, $\varepsilon_+ = 1589$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -2000	0	0 %
-2000 … -1800	0	0 %
-1800 … -1600	2	$3.6 \cdot 10^{-7}$
-1600 … -1400	47	$8.4 \cdot 10^{-6}$
-1400 … -1200	282	$5.0 \cdot 10^{-5}$
-1200 … -1000	9225	$1.7 \cdot 10^{-3}$
-1000 … -800	133132	2.4 %
-800 … -600	243338	4.4 %
-600 … -400	503393	9.0 %
-400 … -200	936695	16.8 %
-200 … 0	1476718	26.4 %
0 … 200	965022	17.3 %
200 … 400	489874	8.8 %
400 … 600	431468	7.7 %
600 … 800	313915	5.6 %
800 … 1000	86125	1.5 %
1000 … 1200	1075	$1.9 \cdot 10^{-4}$
1200 … 1400	109	$1.9 \cdot 10^{-5}$
1400 … 1600	14	$2.5 \cdot 10^{-6}$
1600 … 1800	0	0 %
1800 … 2000	0	0 %
> 2000	0	0 %

Tabelle A.21: Verteilungsfunktion Trimble 1 PPS für Mittelungszeit $\tau = 30$ s

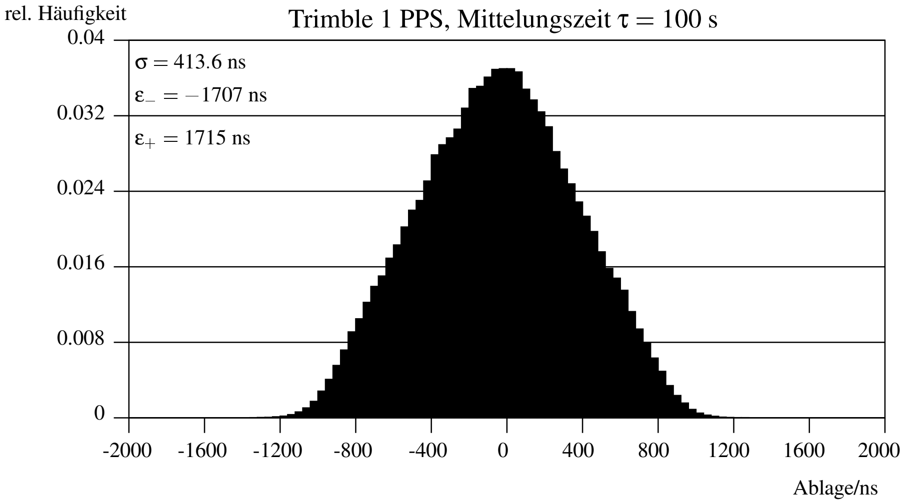

Abbildung A.32: Verteilungsfunktion Trimble 1 PPS für Mittelungszeit $\tau = 100$ s

Trimble 1 PPS, Mittelungszeit $\tau = 100$ s		
$\sigma = 413.6$ ns, $\varepsilon_- = -1707$ ns, $\varepsilon_+ = 1715$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -2000	0	0 %
-2000 ... -1800	0	0 %
-1800 ... -1600	6	$1.1 \cdot 10^{-6}$
-1600 ... -1400	92	$1.6 \cdot 10^{-5}$
-1400 ... -1200	1337	$2.4 \cdot 10^{-4}$
-1200 ... -1000	22276	$4.0 \cdot 10^{-3}$
-1000 ... -800	160996	2.9 %
-800 ... -600	383557	6.9 %
-600 ... -400	607567	10.9 %
-400 ... -200	837797	15.0 %
-200 ... 0	1006425	18.0 %
0 ... 200	975779	17.5 %
200 ... 400	743901	13.3 %
400 ... 600	499459	8.9 %
600 ... 800	270777	4.8 %
800 ... 1000	73788	1.3 %
1000 ... 1200	6189	$1.1 \cdot 10^{-3}$
1200 ... 1400	381	$6.8 \cdot 10^{-5}$
1400 ... 1600	35	$6.3 \cdot 10^{-6}$
1600 ... 1800	2	$3.6 \cdot 10^{-7}$
1800 ... 2000	0	0 %
> 2000	0	0 %

Tabelle A.22: Verteilungsfunktion Trimble 1 PPS für Mittelungszeit $\tau = 100$ s

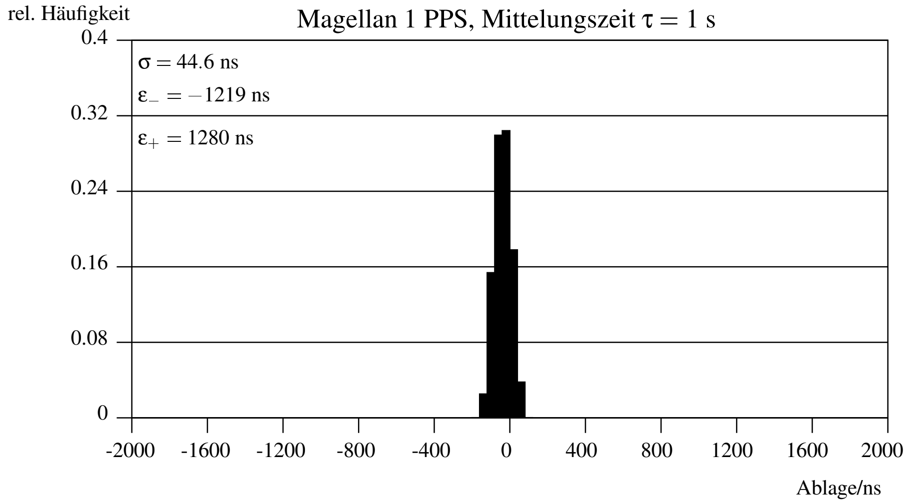

Abbildung A.33: Verteilungsfunktion Magellan 1 PPS für Mittelungszeit $\tau = 1$ s

Magellan 1 PPS, Mittelungszeit $\tau = 1$ s		
$\sigma = 44.6$ ns, $\varepsilon_- = -1219$ ns, $\varepsilon_+ = 1280$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -2000	0	0 %
-2000 … -1800	0	0 %
-1800 … -1600	0	0 %
-1600 … -1400	0	0 %
-1400 … -1200	1	$2.0 \cdot 10^{-7}$
-1200 … -1000	7	$1.4 \cdot 10^{-6}$
-1000 … -800	25	$5.0 \cdot 10^{-6}$
-800 … -600	51	$1.0 \cdot 10^{-5}$
-600 … -400	197	$3.9 \cdot 10^{-5}$
-400 … -200	1094	$2.2 \cdot 10^{-4}$
-200 … 0	3945148	78.3 %
0 … 200	1090318	21.6 %
200 … 400	542	$1.1 \cdot 10^{-4}$
400 … 600	128	$2.5 \cdot 10^{-5}$
600 … 800	39	$7.7 \cdot 10^{-6}$
800 … 1000	11	$2.2 \cdot 10^{-6}$
1000 … 1200	1	$2.0 \cdot 10^{-7}$
1200 … 1400	2	$4.0 \cdot 10^{-7}$
1400 … 1600	0	0 %
1600 … 1800	0	0 %
1800 … 2000	0	0 %
> 2000	0	0 %

Tabelle A.23: Verteilungsfunktion Magellan 1 PPS für Mittelungszeit $\tau = 1$ s

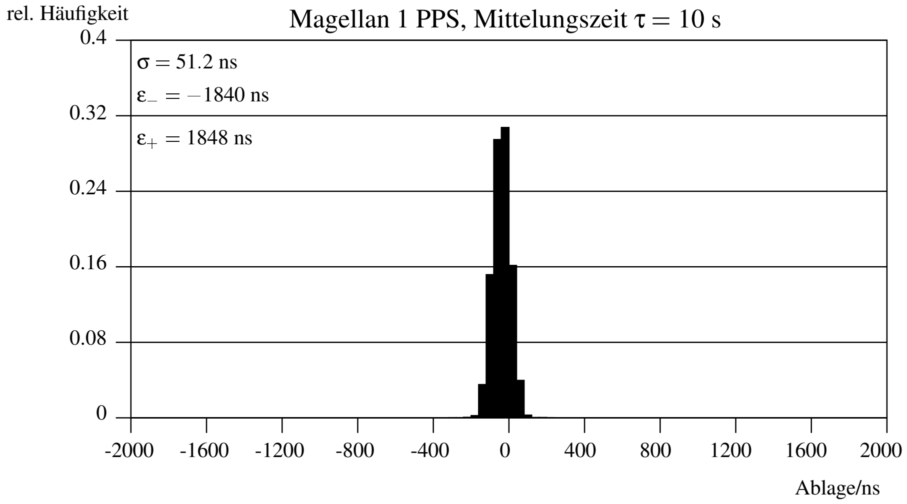

Abbildung A.34: Verteilungsfunktion Magellan 1 PPS für Mittelungszeit $\tau = 10$ s

Magellan 1 PPS, Mittelungszeit $\tau = 10$ s		
$\sigma = 51.2$ ns, $\varepsilon_- = -1840$ ns, $\varepsilon_+ = 1848$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -2000	0	0 %
-2000 … -1800	2	$4.0 \cdot 10^{-7}$
-1800 … -1600	6	$1.2 \cdot 10^{-6}$
-1600 … -1400	15	$3.0 \cdot 10^{-6}$
-1400 … -1200	31	$6.2 \cdot 10^{-6}$
-1200 … -1000	95	$1.9 \cdot 10^{-5}$
-1000 … -800	221	$4.4 \cdot 10^{-5}$
-800 … -600	594	$1.2 \cdot 10^{-4}$
-600 … -400	1752	$3.5 \cdot 10^{-4}$
-400 … -200	7056	$1.4 \cdot 10^{-3}$
-200 … 0	3980922	79.2 %
0 … 200	1029891	20.5 %
200 … 400	3726	$7.4 \cdot 10^{-4}$
400 … 600	1089	$2.2 \cdot 10^{-4}$
600 … 800	349	$6.9 \cdot 10^{-5}$
800 … 1000	150	$3.0 \cdot 10^{-5}$
1000 … 1200	64	$1.3 \cdot 10^{-5}$
1200 … 1400	17	$3.4 \cdot 10^{-6}$
1400 … 1600	7	$1.4 \cdot 10^{-6}$
1600 … 1800	5	$9.9 \cdot 10^{-7}$
1800 … 2000	1	$2.0 \cdot 10^{-7}$
> 2000	0	0 %

Tabelle A.24: Verteilungsfunktion Magellan 1 PPS für Mittelungszeit $\tau = 10$ s

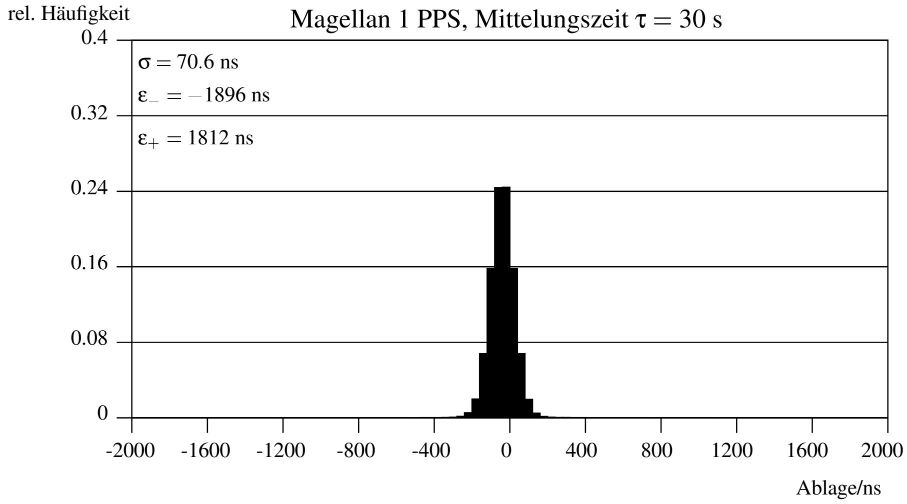

Abbildung A.35: Verteilungsfunktion Magellan 1 PPS für Mittelungszeit $\tau = 30$ s

Magellan 1 PPS, Mittelungszeit $\tau = 30$ s		
$\sigma = 70.6$ ns, $\varepsilon_- = -1896$ ns, $\varepsilon_+ = 1812$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -2000	0	0 %
-2000 … -1800	5	$10.0 \cdot 10^{-7}$
-1800 … -1600	9	$1.8 \cdot 10^{-6}$
-1600 … -1400	23	$4.6 \cdot 10^{-6}$
-1400 … -1200	75	$1.5 \cdot 10^{-5}$
-1200 … -1000	166	$3.3 \cdot 10^{-5}$
-1000 … -800	521	$1.0 \cdot 10^{-4}$
-800 … -600	1243	$2.5 \cdot 10^{-4}$
-600 … -400	4658	$9.3 \cdot 10^{-4}$
-400 … -200	43750	$8.7 \cdot 10^{-3}$
-200 … 0	3676531	73.5 %
0 … 200	1261218	25.2 %
200 … 400	10876	$2.2 \cdot 10^{-3}$
400 … 600	2518	$5.0 \cdot 10^{-4}$
600 … 800	904	$1.8 \cdot 10^{-4}$
800 … 1000	316	$6.3 \cdot 10^{-5}$
1000 … 1200	116	$2.3 \cdot 10^{-5}$
1200 … 1400	52	$1.0 \cdot 10^{-5}$
1400 … 1600	13	$2.6 \cdot 10^{-6}$
1600 … 1800	2	$4.0 \cdot 10^{-7}$
1800 … 2000	0	0 %
> 2000	0	0 %

Tabelle A.25: Verteilungsfunktion Magellan 1 PPS für Mittelungszeit $\tau = 30$ s

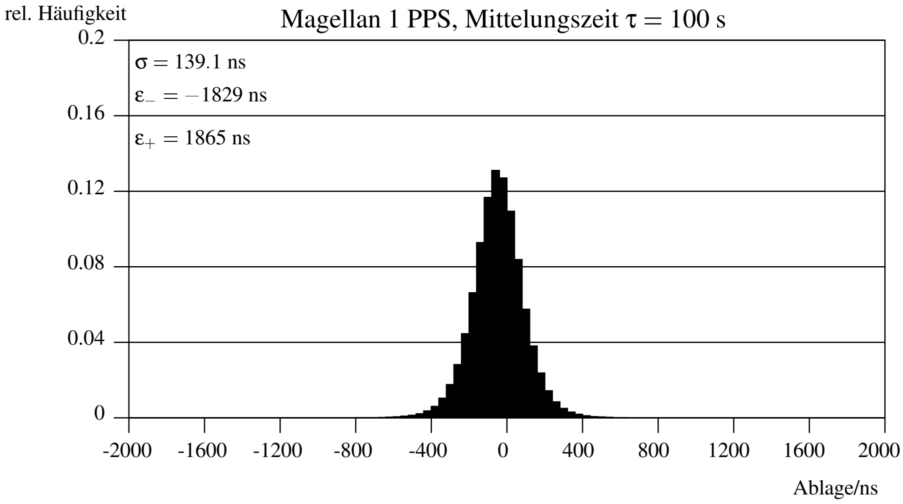

Abbildung A.36: Verteilungsfunktion Magellan 1 PPS für Mittelungszeit $\tau = 100$ s

Magellan 1 PPS, Mittelungszeit $\tau = 100$ s		
$\sigma = 139.1$ ns, $\varepsilon_- = -1829$ ns, $\varepsilon_+ = 1865$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -2000	0	0 %
-2000 … -1800	9	$1.8 \cdot 10^{-6}$
-1800 … -1600	11	$2.2 \cdot 10^{-6}$
-1600 … -1400	15	$3.0 \cdot 10^{-6}$
-1400 … -1200	118	$2.4 \cdot 10^{-5}$
-1200 … -1000	337	$6.8 \cdot 10^{-5}$
-1000 … -800	1044	$2.1 \cdot 10^{-4}$
-800 … -600	4820	$9.8 \cdot 10^{-4}$
-600 … -400	43249	$8.8 \cdot 10^{-3}$
-400 … -200	528795	10.7 %
-200 … 0	2639398	53.4 %
0 … 200	1545876	31.3 %
200 … 400	162586	3.3 %
400 … 600	13492	$2.7 \cdot 10^{-3}$
600 … 800	1661	$3.4 \cdot 10^{-4}$
800 … 1000	371	$7.5 \cdot 10^{-5}$
1000 … 1200	86	$1.7 \cdot 10^{-5}$
1200 … 1400	34	$6.9 \cdot 10^{-6}$
1400 … 1600	7	$1.4 \cdot 10^{-6}$
1600 … 1800	47	$9.5 \cdot 10^{-6}$
1800 … 2000	9	$1.8 \cdot 10^{-6}$
> 2000	0	0 %

Tabelle A.26: Verteilungsfunktion Magellan 1 PPS für Mittelungszeit $\tau = 100$ s

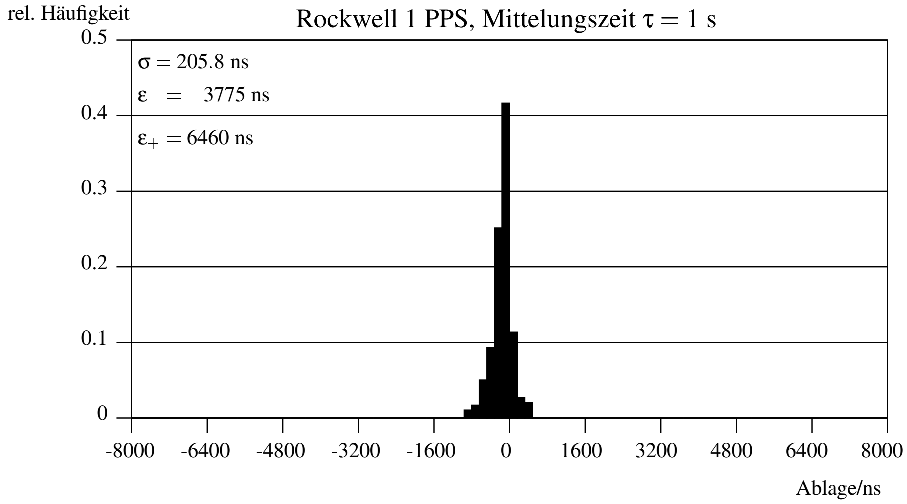

Abbildung A.37: Verteilungsfunktion Rockwell 1 PPS für Mittelungszeit $\tau = 1$ s

Rockwell 1 PPS, Mittelungszeit $\tau = 1$ s		
$\sigma = 205.8$ ns, $\varepsilon_- = -3775$ ns, $\varepsilon_+ = 6460$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -8000	0	0 %
-8000 … -7200	0	0 %
-7200 … -6400	0	0 %
-6400 … -5600	0	0 %
-5600 … -4800	0	0 %
-4800 … -4000	0	0 %
-4000 … -3200	1	$1.8 \cdot 10^{-7}$
-3200 … -2400	4	$7.2 \cdot 10^{-7}$
-2400 … -1600	66	$1.2 \cdot 10^{-5}$
-1600 … -800	58483	1.0 %
-800 … 0	4632096	82.9 %
0 … 800	899541	16.1 %
800 … 1600	0	0 %
1600 … 2400	17	$3.0 \cdot 10^{-6}$
2400 … 3200	0	0 %
3200 … 4000	1	$1.8 \cdot 10^{-7}$
4000 … 4800	0	0 %
4800 … 5600	0	0 %
5600 … 6400	17	$3.0 \cdot 10^{-6}$
6400 … 7200	0	0 %
7200 … 8000	0	0 %
> 8000	0	0 %

Tabelle A.27: Verteilungsfunktion Rockwell 1 PPS für Mittelungszeit $\tau = 1$ s

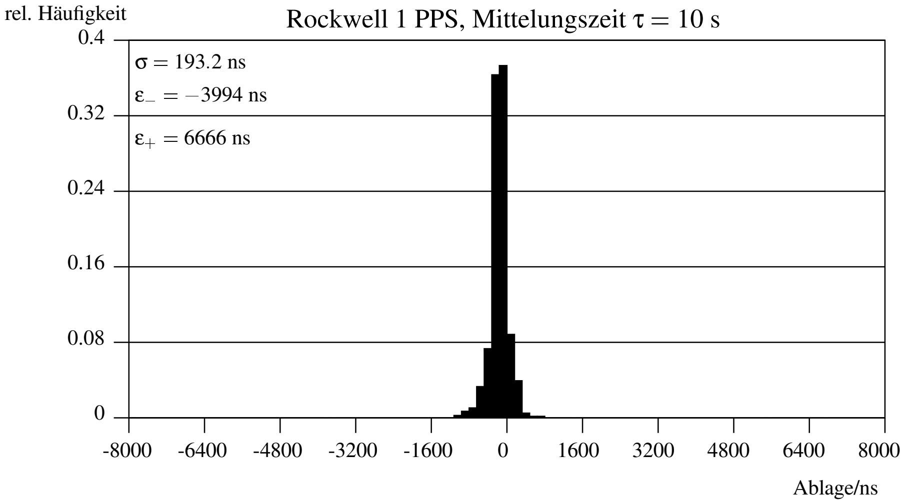

Abbildung A.38: Verteilungsfunktion Rockwell 1 PPS für Mittelungszeit $\tau = 10$ s

Rockwell 1 PPS, Mittelungszeit $\tau = 10$ s		
$\sigma = 193.2$ ns, $\varepsilon_- = -3994$ ns, $\varepsilon_+ = 6666$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -8000	0	0 %
-8000 … -7200	0	0 %
-7200 … -6400	0	0 %
-6400 … -5600	0	0 %
-5600 … -4800	0	0 %
-4800 … -4000	5	$8.9 \cdot 10^{-7}$
-4000 … -3200	5	$8.9 \cdot 10^{-7}$
-3200 … -2400	4	$7.2 \cdot 10^{-7}$
-2400 … -1600	738	$1.3 \cdot 10^{-4}$
-1600 … -800	53489	$9.6 \cdot 10^{-3}$
-800 … 0	4774121	85.4 %
0 … 800	761306	13.6 %
800 … 1600	62	$1.1 \cdot 10^{-5}$
1600 … 2400	152	$2.7 \cdot 10^{-5}$
2400 … 3200	12	$2.1 \cdot 10^{-6}$
3200 … 4000	5	$8.9 \cdot 10^{-7}$
4000 … 4800	5	$8.9 \cdot 10^{-7}$
4800 … 5600	12	$2.1 \cdot 10^{-6}$
5600 … 6400	148	$2.6 \cdot 10^{-5}$
6400 … 7200	10	$1.8 \cdot 10^{-6}$
7200 … 8000	0	0 %
> 8000	0	0 %

Tabelle A.28: Verteilungsfunktion Rockwell 1 PPS für Mittelungszeit $\tau = 10$ s

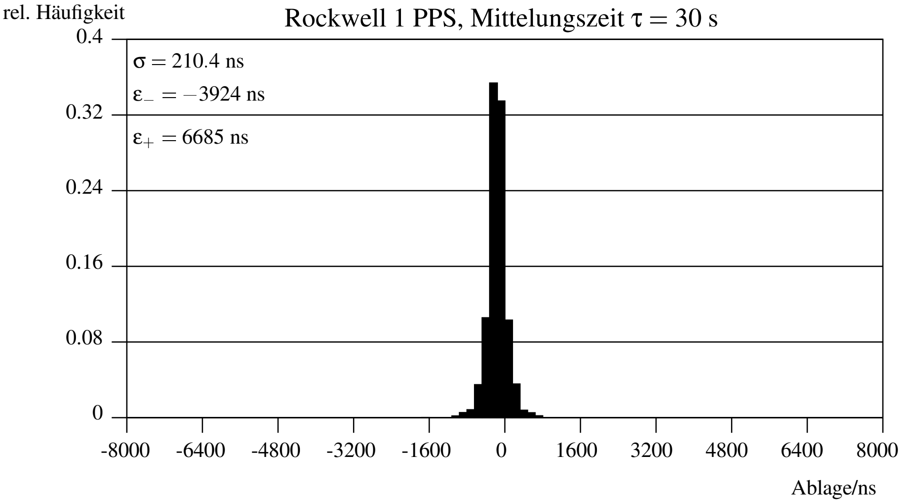

Abbildung A.39: Verteilungsfunktion Rockwell 1 PPS für Mittelungszeit $\tau = 30$ s

Rockwell 1 PPS, Mittelungszeit $\tau = 30$ s		
$\sigma = 210.4$ ns, $\varepsilon_- = -3924$ ns, $\varepsilon_+ = 6685$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -8000	0	0 %
-8000 ... -7200	0	0 %
-7200 ... -6400	0	0 %
-6400 ... -5600	0	0 %
-5600 ... -4800	0	0 %
-4800 ... -4000	26	$4.7 \cdot 10^{-6}$
-4000 ... -3200	4	$7.2 \cdot 10^{-7}$
-3200 ... -2400	10	$1.8 \cdot 10^{-6}$
-2400 ... -1600	2161	$3.9 \cdot 10^{-4}$
-1600 ... -800	41673	$7.5 \cdot 10^{-3}$
-800 ... 0	4682682	83.8 %
0 ... 800	862205	15.4 %
800 ... 1600	249	$4.5 \cdot 10^{-5}$
1600 ... 2400	435	$7.8 \cdot 10^{-5}$
2400 ... 3200	61	$1.1 \cdot 10^{-5}$
3200 ... 4000	15	$2.7 \cdot 10^{-6}$
4000 ... 4800	15	$2.7 \cdot 10^{-6}$
4800 ... 5600	44	$7.9 \cdot 10^{-6}$
5600 ... 6400	452	$8.1 \cdot 10^{-5}$
6400 ... 7200	14	$2.5 \cdot 10^{-6}$
7200 ... 8000	0	0 %
> 8000	0	0 %

Tabelle A.29: Verteilungsfunktion Rockwell 1 PPS für Mittelungszeit $\tau = 30$ s

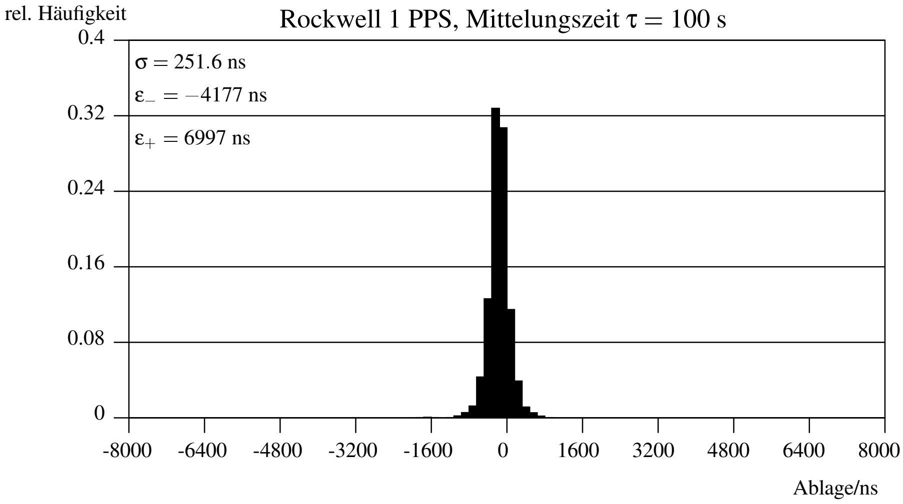

Abbildung A.40: Verteilungsfunktion Rockwell 1 PPS für Mittelungszeit $\tau = 100$ s

Rockwell 1 PPS, Mittelungszeit $\tau = 100$ s		
$\sigma = 251.6$ ns, $\varepsilon_- = -4177$ ns, $\varepsilon_+ = 6997$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -8000	0	0 %
-8000 … -7200	0	0 %
-7200 … -6400	0	0 %
-6400 … -5600	0	0 %
-5600 … -4800	0	0 %
-4800 … -4000	35	$6.3 \cdot 10^{-6}$
-4000 … -3200	65	$1.2 \cdot 10^{-5}$
-3200 … -2400	33	$5.9 \cdot 10^{-6}$
-2400 … -1600	7002	$1.3 \cdot 10^{-3}$
-1600 … -800	46639	$8.3 \cdot 10^{-3}$
-800 … 0	4567616	81.7 %
0 … 800	963900	17.2 %
800 … 1600	1211	$2.2 \cdot 10^{-4}$
1600 … 2400	1538	$2.8 \cdot 10^{-4}$
2400 … 3200	137	$2.5 \cdot 10^{-5}$
3200 … 4000	50	$8.9 \cdot 10^{-6}$
4000 … 4800	50	$8.9 \cdot 10^{-6}$
4800 … 5600	48	$8.6 \cdot 10^{-6}$
5600 … 6400	1628	$2.9 \cdot 10^{-4}$
6400 … 7200	24	$4.3 \cdot 10^{-6}$
7200 … 8000	0	0 %
> 8000	0	0 %

Tabelle A.30: Verteilungsfunktion Rockwell 1 PPS für Mittelungszeit $\tau = 100$ s

A.5 Ablagen der 10 MHz-Pulse für verschiedene Mittelungszeiten

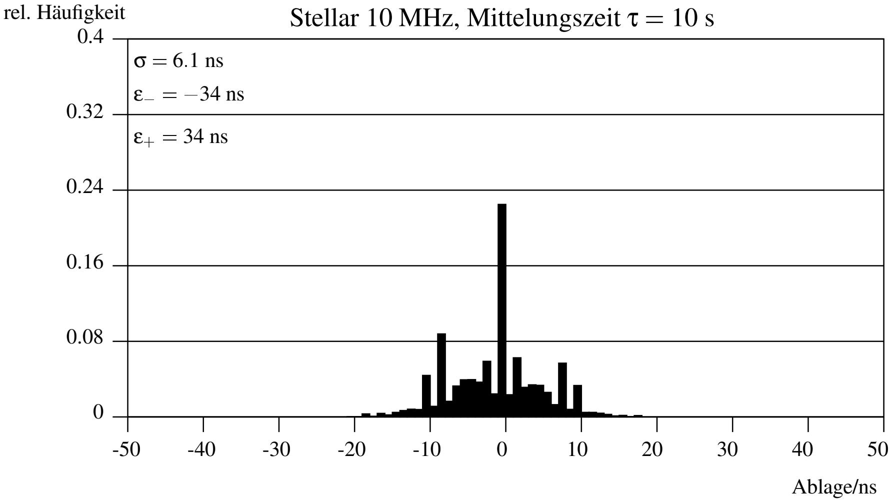

Abbildung A.41: Verteilungsfunktion Stellar 10 MHz für Mittelungszeit $\tau = 10$ s

Stellar 10 MHz, Mittelungszeit $\tau = 10$ s		
$\sigma = 6.1$ ns, $\varepsilon_{-} = -34$ ns, $\varepsilon_{+} = 34$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -50	0	0 %
-50 … -45	0	0 %
-45 … -40	0	0 %
-40 … -35	0	0 %
-35 … -30	23	$4.1 \cdot 10^{-6}$
-30 … -25	289	$5.2 \cdot 10^{-5}$
-25 … -20	3045	$5.4 \cdot 10^{-4}$
-20 … -15	58426	1.0 %
-15 … -10	402151	7.2 %
-10 … -5	1050600	18.8 %
-5 … 0	2151568	38.5 %
0 … 5	1035369	18.5 %
5 … 10	769392	13.8 %
10 … 15	97878	1.8 %
15 … 20	21006	$3.8 \cdot 10^{-3}$
20 … 25	654	$1.2 \cdot 10^{-4}$
25 … 30	58	$1.0 \cdot 10^{-5}$
30 … 35	3	$5.4 \cdot 10^{-7}$
35 … 40	0	0 %
40 … 45	0	0 %
45 … 50	0	0 %
> 50	0	0 %

Tabelle A.31: Verteilungsfunktion Stellar 10 MHz für Mittelungszeit $\tau = 10$ s

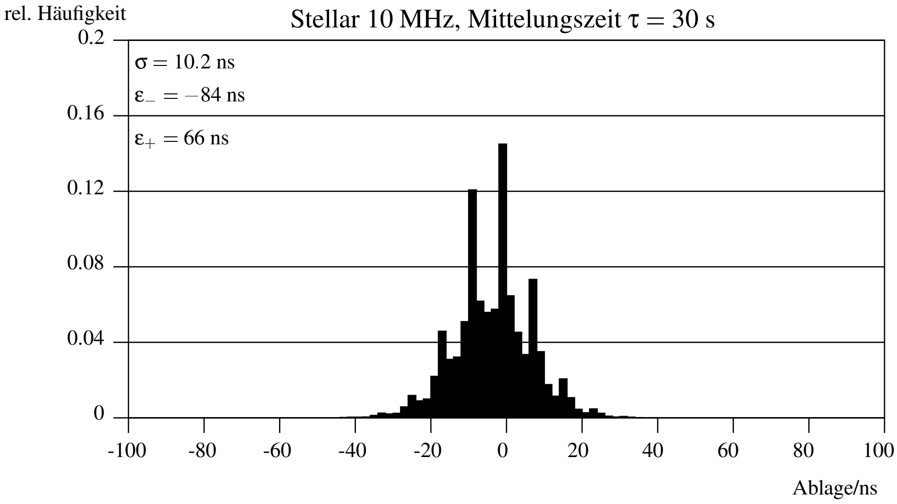

Abbildung A.42: Verteilungsfunktion Stellar 10 MHz für Mittelungszeit $\tau = 30$ s

Stellar 10 MHz, Mittelungszeit $\tau = 30$ s		
$\sigma = 10.2$ ns, $\varepsilon_- = -84$ ns, $\varepsilon_+ = 66$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -100	0	0 %
-100 … -90	0	0 %
-90 … -80	4	$7.2 \cdot 10^{-7}$
-80 … -70	23	$4.1 \cdot 10^{-6}$
-70 … -60	101	$1.8 \cdot 10^{-5}$
-60 … -50	512	$9.2 \cdot 10^{-5}$
-50 … -40	4261	$7.6 \cdot 10^{-4}$
-40 … -30	38078	$6.8 \cdot 10^{-3}$
-30 … -20	219339	3.9 %
-20 … -10	1017955	18.2 %
-10 … 0	2465029	44.1 %
0 … 10	1408886	25.2 %
10 … 20	363728	6.5 %
20 … 30	63552	1.1 %
30 … 40	8178	$1.5 \cdot 10^{-3}$
40 … 50	703	$1.3 \cdot 10^{-4}$
50 … 60	80	$1.4 \cdot 10^{-5}$
60 … 70	5	$8.9 \cdot 10^{-7}$
70 … 80	0	0 %
80 … 90	0	0 %
90 … 100	0	0 %
> 100	0	0 %

Tabelle A.32: Verteilungsfunktion Stellar 10 MHz für Mittelungszeit $\tau = 30$ s

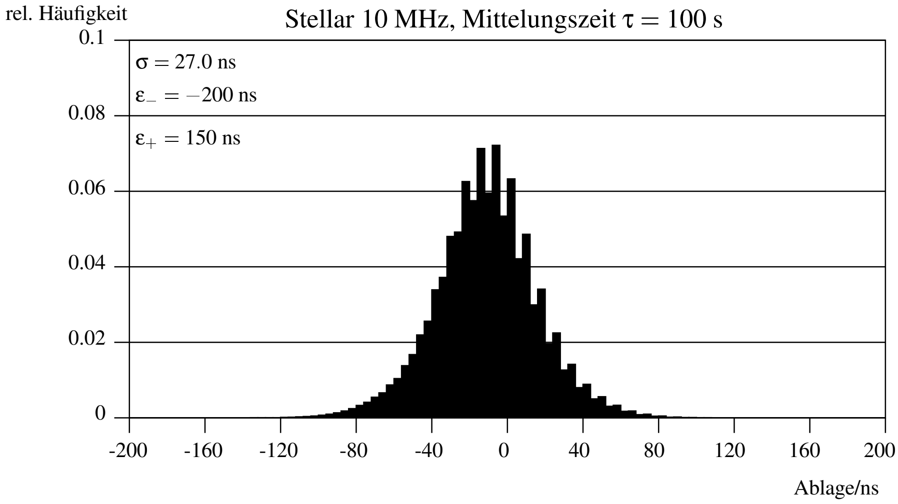

Abbildung A.43: Verteilungsfunktion Stellar 10 MHz für Mittelungszeit $\tau = 100$ s

Stellar 10 MHz, Mittelungszeit $\tau = 100$ s		
$\sigma = 27.0$ ns, $\varepsilon_- = -200$ ns, $\varepsilon_+ = 150$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -200	2	$3.6 \cdot 10^{-7}$
-200 … -180	31	$5.5 \cdot 10^{-6}$
-180 … -160	70	$1.3 \cdot 10^{-5}$
-160 … -140	503	$9.0 \cdot 10^{-5}$
-140 … -120	1645	$2.9 \cdot 10^{-4}$
-120 … -100	8302	$1.5 \cdot 10^{-3}$
-100 … -80	41767	$7.5 \cdot 10^{-3}$
-80 … -60	158211	2.8 %
-60 … -40	495702	8.9 %
-40 … -20	1291951	23.1 %
-20 … 0	1755757	31.4 %
0 … 20	1219205	21.8 %
20 … 40	430833	7.7 %
40 … 60	145352	2.6 %
60 … 80	33013	$5.9 \cdot 10^{-3}$
80 … 100	6606	$1.2 \cdot 10^{-3}$
100 … 120	1107	$2.0 \cdot 10^{-4}$
120 … 140	290	$5.2 \cdot 10^{-5}$
140 … 160	17	$3.0 \cdot 10^{-6}$
160 … 180	0	0 %
180 … 200	0	0 %
> 200	0	0 %

Tabelle A.33: Verteilungsfunktion Stellar 10 MHz für Mittelungszeit $\tau = 100$ s

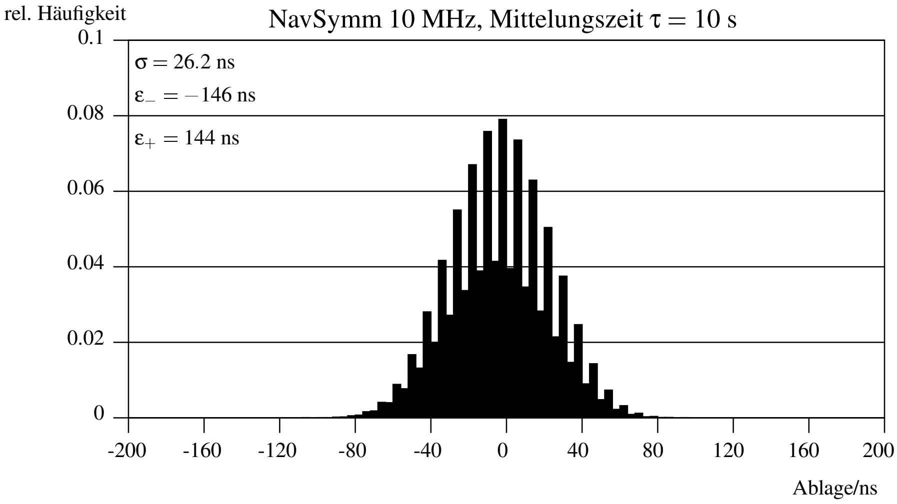

Abbildung A.44: Verteilungsfunktion NavSymm 10 MHz für Mittelungszeit $\tau = 10$ s

NavSymm 10 MHz, Mittelungszeit $\tau = 10$ s		
$\sigma = 26.2$ ns, $\varepsilon_- = -146$ ns, $\varepsilon_+ = 144$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -200	0	0 %
-200 … -180	0	0 %
-180 … -160	0	0 %
-160 … -140	4	$7.2 \cdot 10^{-7}$
-140 … -120	117	$2.1 \cdot 10^{-5}$
-120 … -100	738	$1.3 \cdot 10^{-4}$
-100 … -80	6636	$1.2 \cdot 10^{-3}$
-80 … -60	68684	1.2 %
-60 … -40	416605	7.5 %
-40 … -20	992260	17.8 %
-20 … 0	1688802	30.2 %
0 … 20	1335391	23.9 %
20 … 40	831180	14.9 %
40 … 60	211001	3.8 %
60 … 80	33747	$6.0 \cdot 10^{-3}$
80 … 100	1935	$3.5 \cdot 10^{-4}$
100 … 120	347	$6.2 \cdot 10^{-5}$
120 … 140	18	$3.2 \cdot 10^{-6}$
140 … 160	2	$3.6 \cdot 10^{-7}$
160 … 180	0	0 %
180 … 200	0	0 %
> 200	0	0 %

Tabelle A.34: Verteilungsfunktion NavSymm 10 MHz für Mittelungszeit $\tau = 10$ s

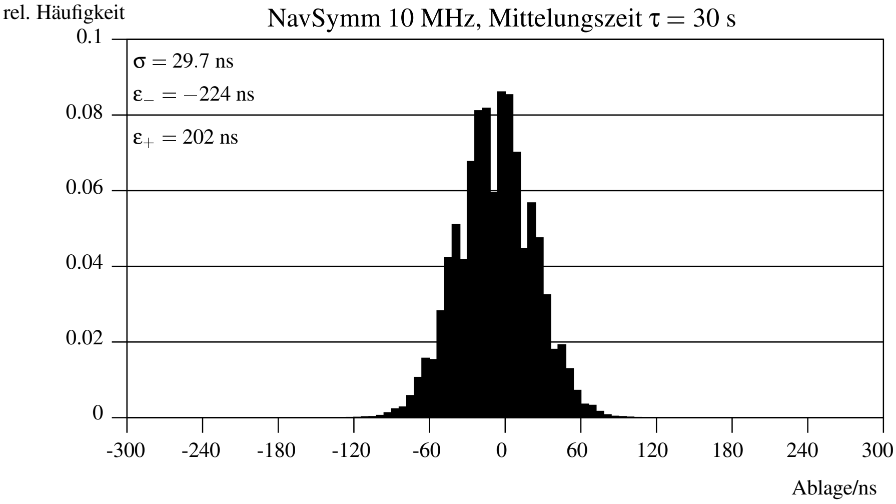

Abbildung A.45: Verteilungsfunktion NavSymm 10 MHz für Mittelungszeit $\tau = 30$ s

NavSymm 10 MHz, Mittelungszeit $\tau = 30$ s		
$\sigma = 29.7$ ns, $\varepsilon_- = -224$ ns, $\varepsilon_+ = 202$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -300	0	0 %
-300 … -270	0	0 %
-270 … -240	0	0 %
-240 … -210	5	$8.9 \cdot 10^{-7}$
-210 … -180	13	$2.3 \cdot 10^{-6}$
-180 … -150	74	$1.3 \cdot 10^{-5}$
-150 … -120	916	$1.6 \cdot 10^{-4}$
-120 … -90	14535	$2.6 \cdot 10^{-3}$
-90 … -60	208810	3.7 %
-60 … -30	999108	17.9 %
-30 … 0	2101924	37.6 %
0 … 30	1701267	30.4 %
30 … 60	502893	9.0 %
60 … 90	53901	$9.6 \cdot 10^{-3}$
90 … 120	3362	$6.0 \cdot 10^{-4}$
120 … 150	286	$5.1 \cdot 10^{-5}$
150 … 180	43	$7.7 \cdot 10^{-6}$
180 … 210	2	$3.6 \cdot 10^{-7}$
210 … 240	0	0 %
240 … 270	0	0 %
270 … 300	0	0 %
> 300	0	0 %

Tabelle A.35: Verteilungsfunktion NavSymm 10 MHz für Mittelungszeit $\tau = 30$ s

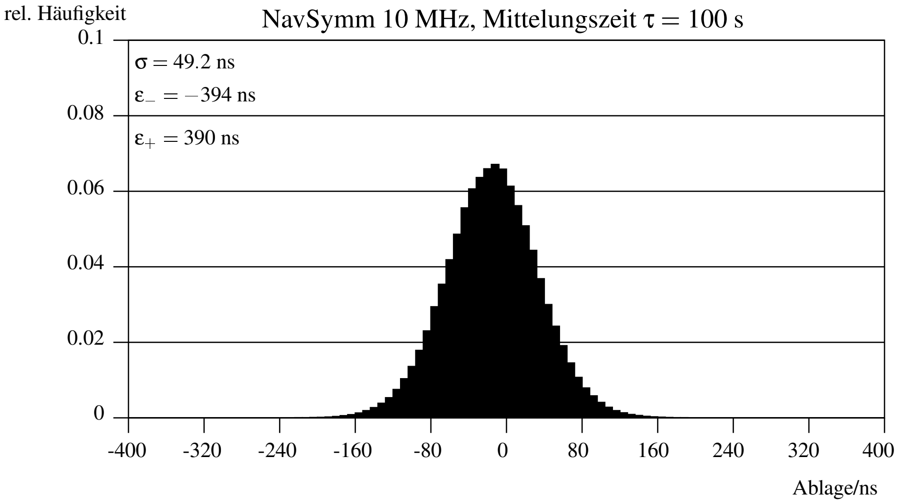

Abbildung A.46: Verteilungsfunktion NavSymm 10 MHz für Mittelungszeit $\tau = 100$ s

NavSymm 10 MHz, Mittelungszeit $\tau = 100$ s		
$\sigma = 49.2$ ns, $\varepsilon_- = -394$ ns, $\varepsilon_+ = 390$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -400	3	$5.4 \cdot 10^{-7}$
-400 … -360	9	$1.6 \cdot 10^{-6}$
-360 … -320	31	$5.5 \cdot 10^{-6}$
-320 … -280	93	$1.7 \cdot 10^{-5}$
-280 … -240	387	$6.9 \cdot 10^{-5}$
-240 … -200	1826	$3.3 \cdot 10^{-4}$
-200 … -160	12727	$2.3 \cdot 10^{-3}$
-160 … -120	85024	1.5 %
-120 … -80	404968	7.2 %
-80 … -40	1178894	21.1 %
-40 … 0	1805926	32.3 %
0 … 40	1394612	25.0 %
40 … 80	551204	9.9 %
80 … 120	126864	2.3 %
120 … 160	20181	$3.6 \cdot 10^{-3}$
160 … 200	2728	$4.9 \cdot 10^{-4}$
200 … 240	394	$7.1 \cdot 10^{-5}$
240 … 280	70	$1.3 \cdot 10^{-5}$
280 … 320	53	$9.5 \cdot 10^{-6}$
320 … 360	18	$3.2 \cdot 10^{-6}$
360 … 400	7	$1.3 \cdot 10^{-6}$
> 400	0	0 %

Tabelle A.36: Verteilungsfunktion NavSymm 10 MHz für Mittelungszeit $\tau = 100$ s

A.6 Sonstige Verteilungen

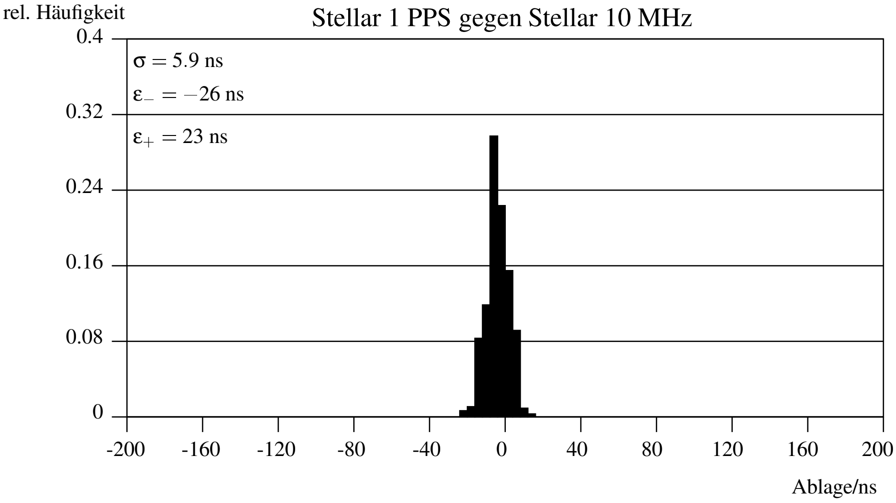

Abbildung A.47: Verteilungsfunktion Stellar 1 PPS gg. Stellar 10 MHz

Stellar 1 PPS gegen Stellar 10 MHz		
$\sigma = 5.9$ ns, $\varepsilon_- = -26$ ns, $\varepsilon_+ = 23$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -200	0	0 %
-200 … -180	0	0 %
-180 … -160	0	0 %
-160 … -140	0	0 %
-140 … -120	0	0 %
-120 … -100	0	0 %
-100 … -80	0	0 %
-80 … -60	0	0 %
-60 … -40	0	0 %
-40 … -20	37684	$6.7 \cdot 10^{-3}$
-20 … 0	4103496	73.4 %
0 … 20	1449452	25.9 %
20 … 40	0	0 %
40 … 60	0	0 %
60 … 80	0	0 %
80 … 100	0	0 %
100 … 120	0	0 %
120 … 140	0	0 %
140 … 160	0	0 %
160 … 180	0	0 %
180 … 200	0	0 %
> 200	0	0 %

Tabelle A.37: Verteilungsfunktion Stellar 1 PPS gg. Stellar 10 MHz

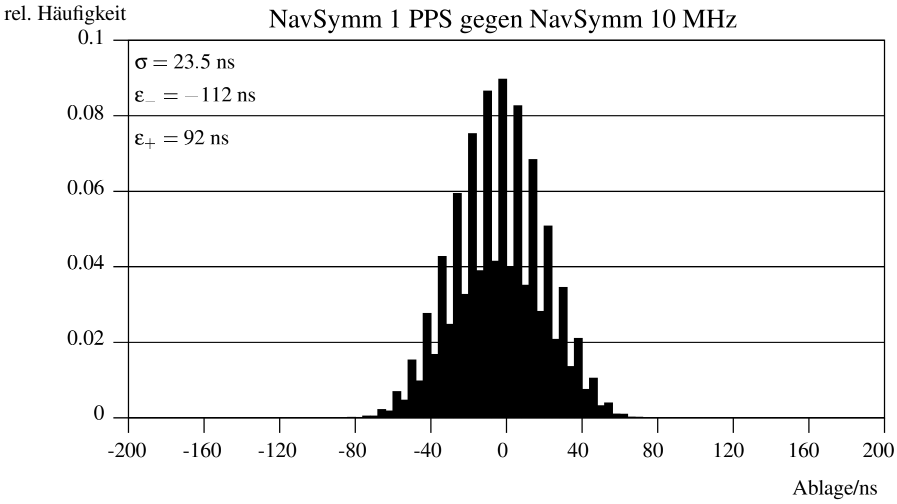

Abbildung A.48: Verteilungsfunktion NavSymm 1 PPS gg. NavSymm 10 MHz

NavSymm 1 PPS gegen NavSymm 10 MHz		
$\sigma = 23.5$ ns, $\varepsilon_- = -112$ ns, $\varepsilon_+ = 92$ ns		
Intervall	abs. Häufigkeit	rel.Häufigkeit
< -200	0	0 %
-200 … -180	0	0 %
-180 … -160	0	0 %
-160 … -140	0	0 %
-140 … -120	0	0 %
-120 … -100	41	$7.3 \cdot 10^{-6}$
-100 … -80	979	$1.8 \cdot 10^{-4}$
-80 … -60	27792	$5.0 \cdot 10^{-3}$
-60 … -40	359185	6.4 %
-40 … -20	985498	17.6 %
-20 … 0	1854338	33.2 %
0 … 20	1421156	25.4 %
20 … 40	785502	14.1 %
40 … 60	145777	2.6 %
60 … 80	7506	$1.3 \cdot 10^{-3}$
80 … 100	13	$2.3 \cdot 10^{-6}$
100 … 120	0	0 %
120 … 140	0	0 %
140 … 160	0	0 %
160 … 180	0	0 %
180 … 200	0	0 %
> 200	0	0 %

Tabelle A.38: Verteilungsfunktion NavSymm 1 PPS gg. NavSymm 10 MHz

B Listings

B.1 Hauptmodule

B.1.1 Hauptmodul

Modul GPSEVAL.CPP

```
//: GPSEVAL.CPP - Software zum Ansprechen der Meßhardware
//. für die GPS Evaluationsmessungen
//.
//. Autor: Dieter Höchtl (E881/9125460)
//. Geschichte:
//. 02.08.1995 Erstellung
//. 13.10.1995 Überarbeitung: Triggerlänge auf 8 Sekunden
//.            ausdehnen
//. 26.10.1995 Beseitigung grober Fehler

//. Bestandteile:
//.
//. GPSEVAL .CPP - Hauptmodul, enthält Schleife für
//.                Tastatureingaben und Pollingbetrieb
//. SYS     .CPP - Bedient alle Systemteile (LA, IO, COM)
//. LA      .CPP - Routinen zur Bearbeitung der Rohdaten
//.                von den LAs
//. IO      .CPP - Interruptroutine für ACK-Flanke der
//.                par. Schnitt. (TRIGGER)
//. COM     .CPP - Routinen zur Verarbeitung der GPS-
//.                Messages über RS232
//. TIMER   .CPP - Timerroutinen für Systemtimer
//.                (Auflösung 0,83 us)
//. SERIALBO.CPP - Low-Level Routinen für RS232
//.                (Pufferung, Zeitstempel)
//. OUT     .CPP - Routine zur einfachen Ausgabe auf
//.                Bildschirm
//. HP16500. CPP - Enthält Routinen zur Kommunikation
//.                mit den beiden LAs
//. IEEE488. LIB - Bibliothek für Kommunikation über
//.                den HPIB-Bus

// Includes

#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include "sys.h"
#include "out.h"
#include "global.h"
#include "timer.h"

// Instanz des Systemobjekts

Sys syst;

//- main    Hauptschleife

int main(void) {
  int done = 0;
  int code = 0;
  int i;

  // --- Bildschirmausgabe
  out.Clear();
  out.Box(0,0,79,24,0x17);
  out.CenterText(0,"GPS-Evaluation Software Version vom "
                 "09.11.1995 (a)",0x1f);
  out.PosText(1,1,"Pulsliste von Sekunde",0x14);
  out.PosText(1,2,"Quelle    Ablage (ns) Sekunde(s) "
                "1(ms) 2(ms)   Spezifische Daten",0x12);
  for (i=0; i<8; i++) {
    out.PosText(1,3+i,gpsName(i),0x1f);
  }
  out.PosText(1,12,"Fehlpulsliste von Sekunde",0x14);
  out.PosText(1,13,"Quelle    Ablage (ns) Sekunde(s) "
            "1(ms) 2(ms)   Spezifische Daten",0x12);
  out.PosText(1,20,"                    PulsTail        "
            "PulsSent            PulsHead     "
            "    ",0x17);
  out.PosText(1,21,"Sekunde:            FehlTail        "
            "FehlSent            FehlHead       "
            "     ",0x17);
  out.PosText(1,22,"La     :              PC-Zeit:      "
            "      ",0x17);
  out.PosText(1,23,"Trigger:              Lost: 0   "
            "PC-Uhr:     ",0x17);

  // --- Initialisierung des ganzen Systems
  if (syst.Initialize() != 0) done = 1;

  // --- Bedienung des Systems im Pollingbetrieb
  while (!done) {
    // Polling (Synchroner Aufruf aller Systemkomponenten)
    if (syst.Synchron() != 0) done = 1;

    // Tastaturcheck ?
    if (kbhit()) {
      switch (getch()) {
      // Abbruchcode: 4715 (zur Vermeidung
      // eines unabsichtlichen Abbruchs)
      case '4': code = 1; break;
      case '7': if (code == 1) code = 2; break;
      case '1': if (code == 2) code = 3; break;
      case '5':
        if (code == 3) {
          done = 1;
          break;
        }
      }
    }
  }

  // --- Wurde System durch fatalen Fehler abbgebrochen
  // Fehler anzeigen
```

```
  if (phase==PhFatal) {
    out.PosTextB(78,1,12,gerrstr(gerr),0xCF);
    sprintf(buf,"Triggerlänge: %11.6f",TriggerLength);
    out.PosTextB(78,1,13,buf,0xCF);
  }
  return 0;
}
```

B.1.2 Globale Variablen

Header GLOBAL.H

```
#ifndef __GLOBAL_H    /* prevent multiple includes */
#define __GLOBAL_H

#include "la.h"
#include "out.h"

// --- mögliche Systemzustände (Variable phase)

enum {
  PhFatal=-1,
  PhInit,
  PhTrigger1,
  PhTrigger2,
  PhFirstLa,
  PhNormal
};

// --- Fatale Fehler (Variable gerr)

enum {
  Gerr_NoError=0,
  Gerr_LaInit,
  Gerr_LaStart,
  Gerr_BocaInit,
  Gerr_LptInit,
  Gerr_TriggerLength,
  Gerr_TooMuchTriggerLost,
  Gerr_Trigger12TimeOut,
  Gerr_NoLaData,
  Gerr_Inconsistency,
  Gerr_LaStateFalse,
  Gerr_NormSilent,
  Gerr_SaveFileError,
  Gerr_ErrFileError,
  Gerr_SaveFifoOverflow,
  Gerr_ErrFifoOverflow,
  Gerr_LaFifoOverflow
};


// --- Fehlerwert für ein LA-Datenpaket

// Problem mit den La-Daten (siehe SysLa)
#define SysLaDataProblem       0x01
// Ein Triggerinterrupt ist nicht gekommen: wurde simuliert
#define SysTriggerLost         0x02
// Ein fehlerhafter Trigger ist aufgetreten
#define SysTriggerErr          0x04
// Inkonsistente Timerwerte ausgelesen
#define SysTimerFail           0x08
// Kein Trigger war in den LA-Daten zu erkennen
#define SysLaNoTrigger         0x10
// Irreguläres Verhalten des Triggers in den LA-Daten
#define SysLaIrregularTrigger  0x20
// Fehlerhafter Level des Triggers in den LA-Daten
#define SysLaWrongTriggerState 0x30
// Falsche Anzahl an Ref100-Pulsen
#define SysLaWrongNumberRef100 0x40
// Es konnte kein Triggerende gefunden werden
#define SysNoTriggerEnd        0x50

// --- Maskierung der einzelnen LA-Kanäle

// (Flanke bei gesetztem Bit berücksichtigen)

#define MaskRising        0x1FFF   // Erste 13 Kanäle
#define MaskFalling       0x0000

// --- Flags für Daten in GPS-FIFO

// Receiver zeigt gültige Daten an
#define GpsDataValid          0x02
// Receiver zeigt Receiver-Fault an
#define GpsReceiverFault      0x04
// Recevier zeigt Antenna-Fault an
#define GpsAntennaFault       0x08
// Erste Nachricht fehlerhaft (Prüfsumme)
#define GpsCorruptedMessage1  0x10
// Zweite Nachricht fehlerhaft (Prüfsumme)
#define GpsCorruptedMessage2  0x20

// --- Flags für Pulsdaten eines Kanals

// enthält Datensatz überhaupt Daten ?
#define SaveFilled             0x01
// Eine der GPS-Nachrichten zeigt gültige Daten an
#define SaveDataValid          0x02
// Eine der GPS-Nachrichten zeigt Receiver-Fault an
#define SaveReceiverFault      0x04
// Eine der GPS-Nachrichten zeigt Antenna-Fault an
#define SaveAntennaFault       0x08
// Erste Nachricht fehlerhaft (Prüfsumme)
#define SaveCorruptedMessage1  0x10
// Zweite Nachricht fehlerhaft (Prüfsumme)
#define SaveCorruptedMessage2  0x20
// Zuviele Flanken auf einem Kanal (>20 in einem LA-Paket)
#define SaveJumpyChannel       0x80

// --- Ungültige Werte für verschiedene Datentypen
//     festlegen

#define InvalidLong  0x80000000L
#define InvalidWord   -32768
#define InvalidDouble -9999999999.0


// --- Triggervariablen

// Anzahl maximal zulässiger Triggerinterrupt-Verluste
#define MaxTriggerLost 11

// Toleranz innerhalb derer ein Triggerinterrupt
// noch als korrekt angesehen wird (zur Berücksichtigung
// von Verzögerungen bei der Interruptbehandlung)
#define refEpsilonTrigger  0.002

// Filterungsfaktor für Kalman-Filterung zur Ermittlung
// des PC-Uhr-Frequenzfehlers)
#define kPC .95

// Zeitstempel des vorletzten Triggerinterrupts
extern unsigned long tsTrigger1;

// Zeitstempel des letzten Triggerinterrupts
extern unsigned long tsTrigger2;

// Anzahl der zuletzt hintereinander verlorenen
// Triggerinterrupts
extern int nTriggerLost;

// Anzahl der seit Meßbeginn aufgetretenen
// fehlerhaften Triggerinterrupts (außerhalb des
// Toleranzfensters refEpsilonTrigger
extern int nTriggerErr;

// Die Dauer zwischen den letzten zwei Triggerinterrupts
// (Der PC-Uhrfrequenzfehler wurde hierbei
// schon korrigiert)
extern double TriggerLength;
```

```
// --- Buffer und Variablen für die Kommunikation
//     des Meß-PC mit dem Server-PC

// Buffer und Länge der Daten zum Absenden
// von Nachrichten zum Server-PC
extern char serv[256];
extern int servLen;
//
extern unsigned long ack;

extern int newAck;   // Flag, das anzeigt, daß eine neue
                     // Bestätigungsnachricht für
                     // Pulsdaten einer Meßsekunden,
                     // die an den Server-PC übermittelt
                     // wurde, eingetroffen ist
// Meßsekunde der Meßdaten, die zuletzt bestätigt wurde
extern unsigned long ack;
// wie newAck, nur für für Fehlpulse
extern int newAckErr;
// Nummer des Fehlpulses, der zuletzt bestätigt wurde
extern unsigned long ackErr;

// --- LA-FIFO-Variablen
// Interner Gültigkeitswert der Daten im LA-FIFO
#define LaValid     2
#define LaInValid   1

// maximale Größe des LA-FIFOS (hier 10 Pakete)
#define LaFifoLen      10

// FIFO-Zeiger und Buffer
extern unsigned long fifoHeadLa;
extern unsigned long fifoTailLa;
extern LaData laData[LaFifoLen];

// Interner Gültigkeitswert der Daten im LA-FIFO
#define LaValid     2
#define LaInValid   1


// --- Systemzustand

// globaler Zustand (Werte: PhNormal, PhInit, ...)
extern int phase;
// Fehlerzustand (Gerr_NoError: kein Fehler)
extern int gerr;
// Zusatzinfo für Fehlerzustand
extern int gerrsub;


// --- Frequenzfehler von LA-Uhren und PC-Uhr
//  (relative Frequenzabweichung)

extern double LA1Correction;
extern double LA2Correction;
extern double PCCorrection;

// --- Daten über das letzte LA-Paket

// Zeitstempel des Eintreffens dieses Pakets
extern unsigned long tsLA;

// Triggerlevel des Triggers dieses Pakets (0 oder 1)
extern int stLA;


// --- Temporäre Buffer für diverse Zwecke

extern char buf[256];
extern char str[256];


// --- Namen der einzelnen GPS-Empfänger (0=Stellar, ...)

char *gpsName(int num);


// --- Bezeichnungen der Fehlerzustände

char *gerrstr(int err);

#endif
```

Modul GLOBAL.CPP

```
//: GLOBAL.CPP
//.
//. Globale Variablen, Erklärung siehe GLOBAL.H

#include "global.h"

int phase;
int gerr;
int gerrsub;

unsigned long tsTrigger1;
unsigned long tsTrigger2;
int nTriggerLost = 0;
int nTriggerErr = 0;
unsigned long tsLA;
int stLA;

double TriggerLength;
double LA1Correction;
double LA2Correction;
double PCCorrection;

unsigned long fifoHeadLa = 0;
unsigned long fifoTailLa = 0;
LaData laData[LaFifoLen];

int newAck;
unsigned long ack;
int newAckErr;
unsigned long ackErr;

char buf[256];
char str[256];
char serv[256];
int  servLen;

char *GpsName[]={
  "Stellar",
  "NavSymm",
  "Motorola",
  "Trimble",
  "Magellan",
  "Rockwell",
  "Stellar10" ,
  "NavSymm10"};

char *gerrstring[] = {
  "Kein Fehler",
  "Fehler bei Initialisierung des Logikanalysators",
  "Fehler beim Start des Logikanalysators",
  "Fehler bei Initialisierung der BOCA-Karte und der GPS-Empfänger",
  "Fehler bei Initialisierung der parallelen Schnittstelle",
  "Triggerlänge falsch",
  "Zuviele Trigger verloren",
  "Timeout bei Warten auf 1. oder 2. Trigger",
  "Keine Logikanalysatordaten nach 1. Trigger",
  "Inkosistenz: Höchstwahrscheinlich Softwarefehler",
  "La-Daten: kommen mit falschem Triggerlevel",
  "Normierungsempfänger gibt keinen Puls aus",
  "Fehler bei Ansprechen von Pulslistendatei",
  "Fehler bei Ansprechen von Fehlpulslistendatei",
  "Überlauf des Pulsfifos",
  "Überlauf des Fehlpulsfifos",
  "Überlauf des LA-Fifos"
};

char *gerrstr(int err) {
  return gerrstring[err];
}

char *gpsName(int num) {
```

```
  return GpsName[num];
}
```

B.2 Systemmodule

B.2.1 Systemsteuerung und Kombination von LA- und GPS-Daten

Header SYS.H

```
#ifndef __SYS_H    /* prevent multiple includes */
#define __SYS_H

#include "timer.h"
#include "la.h"
#include "io.h"
#include "com.h"
#include "save.h"


class Sys {
  int init;
  La  logicAnalyser;
  Io   inOut;
  Save saveData;
  long timeout;
  unsigned long t1,t2;
  int ofifo;
  unsigned long oSecond;
public:
  Com  comGPS;
  Sys();
  ~Sys();
  int Initialize();
  int Synchron();
  int CheckTimeOut();
  int Pulse(LaData *h1, LaData *h2, int LaPort,
            int ComPort,int SavePort);
};

#endif
```

Modul SYS.CPP

```
//: SYS.CPP - Software zur Bedienung des gesamten Systems
//.

#include <stdio.h>
#include <dos.h>
#include "global.h"
#include "la.h"
#include "sys.h"

// Systemzustände
//  Phase
//  PhInit    Initialisieren des Logikanalysators
//            Starten des Logikanalysators
//            Initialisieren der seriellen
//              Schnittstelle zur Kommunikation
//              mit den GPS-Empfängern
//            Initialisieren des Triggerinterrupts
// PhTrigger1 Warten auf 1. Triggerinterrupt
// PhTrigger2 Warten auf 2. Triggerinterrupt
// PhFirstLa  Warten auf die kurz nach dem
//              2.Triggerinterrupt übermittelten
//              LA-Daten
// PhNormal   Normaler Auswertungsmodus
//              (erwünschter Dauerzustand)
// PhFatal    Fataler Fehler aufgetreten

//- Konstruktor Sys
//.
//. Starten mit der Phase Init

Sys::Sys() {
  phase = PhInit;
  oSecond = 0;
}

//- Destruktor Sys

Sys::~Sys() {
}

//- int Sys:Initialize()
//.
//. Initialisiere die Systemkomponenten

int Sys::Initialize() {
  // Logikanalysator initialisieren
  out.PosTextB(70,1,24,"Logikanalysator initialisieren",
               0x1C);
  if ((logicAnalyser.Initialize()) != 0) {
    phase =PhFatal; gerr = Gerr_LaInit;
    return -1;
  }
  // Logikanalysatoren starten
  out.PosTextB(70,1,24,"Logikanalysator starten",0x1C);
  if ((logicAnalyser.Start()) != 0) {
    phase =PhFatal; gerr = Gerr_LaInit;
    return -1;
  }
  // 8fach RS232  und GPS-Empfänger vorbereiten
  out.PosTextB(70,1,24,"RS232 und GPS-Emfpänger "
                       "initialisieren",0x1C);
  if ((comGPS.Initialize(0x100,5)) != 0) {
    phase =PhFatal; gerr = Gerr_BocaInit;
    return -1;
  }
  // Triggerinterrupt aktivieren
  out.PosTextB(70,1,24,"Triggerinterrupt initialisieren",
               0x1C);
  if ((inOut.Initialize(0x378,7)) != 0) {
    phase =PhFatal; gerr = Gerr_LptInit;
    return -1;
  }
  // Zeitstempel für Timeout für 1.Trigger bestimmen
  // (Sollte der 1.Trigger nicht innerhalb 8 s kommen,
  // ist etwas faul)
  out.PosTextB(70,1,24,"Warten auf 1.Trigger",0x1C);
  tsTrigger1 = tim.getTime();
  phase = PhTrigger1;

  init = 1;
  return 0;
}

//- int Sys:Synchron
//. In dieser Methode werden alle anfallenden Arbeiten
//. (Anzeige von Daten, Übernahme LA-Daten, Übernahme
//. GPS-Nachrichten, Verrechnung der Daten und
//. Kombination von GPS-Daten und LA-Daten)
//. durchgeführt

int Sys::Synchron() {
  unsigned long t;

  // --- Zeitliche Steuerung
  // Gemeinsam mit der Triggerinterruptroutine steuert
  // diese Methode die zeitlichen Abläufe:
  // 1. Anhängen von leeren Datenpaketen an den LA-FIFO
  //    im 4~s Rhythmus
  // 2. zeitgerechtes Abholen der verechneten Daten von
  //    den Logikanalysatoren. Wurden kein korrektes
  //    Datenpaket innerhalb gewisser Fristen empfangen,
  //    bleibt das Datenpaket leer.
  // 3. Prüfe, ob Trigger verloren ging (Bedingung:
  //    mehr als 8~s seit letztem Trigger vergangen)
```

```
if (CheckTimeOut() != 0)  return -1;

// --- GPS-Nachrichten analysieren und in GPS-FIFO
//     ablegen
comGPS.Synchron();


// --- Falls genug Daten im LA-FIFO vorliegen,
// LA-Daten mit GPS-Nachrichten kombinieren und
// zur Zwischenspeicherung an save-Objekt übergeben
// Genug Daten liegen vor, falls FIFO zumindest so
// aussieht (fifoTailLa >=fifoHeadLa+3):
// 0: zu bearbeitendes LA-Paket  (h1)  <-- fifoHeadLa
// 1: nachfolgendes LA-Paket     (h2)
// 2: Bereich neuer LA-Daten
// 3: ungültig                         <-- fifoTailLa
if (fifoTailLa >=fifoHeadLa+3) {
  // Interrupts sperren
  disable();
  LaData *h1 = &laData[(fifoHeadLa) % LaFifoLen];
  LaData *h2 = &laData[(fifoHeadLa+1) % LaFifoLen];
  // LA-Datenpaket umfaßt Daten für Meßsekunde
  // h1->second ... h1->second+3:
  h1->second = (fifoHeadLa)*4;
  enable();
  //
  if (h2->valid == LaInValid) h2 = 0;
  // Fehler sofern LA-Daten ungültig
  if (h1->valid == LaInValid) {
    h1->state |= SysLaDataProblem;
    LAfailed++;
  }
  // Erzeuge vier neue Pulsdateneinträge im save-Objekt
  saveData.GeneralConditions(h1->second,h1->state);
  saveData.GeneralConditions(h1->second+1,h1->state);
  saveData.GeneralConditions(h1->second+2,h1->state);
  saveData.GeneralConditions(h1->second+3,h1->state);
  // Kombiniere GPS- und LA-Daten und fülle
  // Pulseinträge aus
  if (!(h1->state & SysLaDataProblem)) {
    // Stellar
    Pulse(h1,h2,6,0,0);
    // NavSymm
    Pulse(h1,h2,8,1,1);
    // Motorola
    Pulse(h1,h2,7,2,2);
    // Trimble
    Pulse(h1,h2,10,3,3);
    // Magellan
    Pulse(h1,h2,9,4,4);
    // Rockwell
    Pulse(h1,h2,12,5,5);
    // Stellar 10 MHz
    Pulse(h1,h2,3,7,6);
    // NavSymm 10 MHz
    Pulse(h1,h2,4,7,7);
  }
  // LA-Datenpaket aus FIFO entfernen
  h1->valid = LaInValid;
  h1->state = 0;
  fifoHeadLa++;
}

// --- Polling-Aufruf für das save-Objekt. Hier wird,
// falls die Ausgabebuffer der RS232-Schnittstelle leer
// sind, versucht die Pulsdaten an den Server-PC zu
// übermitteln
saveData.Synchron();

// --- Anzeigen duchführen
// Momentane Phase in der letzten Zeile anzeigen
switch (phase) {
  case PhInit:
    out.PosTextB(40,1,24,"Initialisieren",0x1C);
    break;
  case PhTrigger1:
    out.PosTextB(40,1,24,"Warten auf 1. Trigger",0x1C);
    break;
  case PhTrigger2:
    out.PosTextB(40,1,24,"Warten auf 2. Trigger",0x1C);
    break;
  case PhFirstLa:
    out.PosTextB(40,1,24,"Warten auf 1. LA-Daten",0x1C);
    break;
  case PhNormal:
    out.PosTextB(40,1,24,"Normaler Betrieb",0x1A);
    break;
  case PhFatal:
    out.PosTextB(40,1,24,"Fataler Fehler",0x1f);
    break;
}

// Zeitstempel des letzten Triggers anzeigen
if (phase != PhTrigger1) {
  if (phase == PhTrigger2) {
    t = tsTrigger1;
  } else {
    t = tsTrigger2;
  }
  sprintf(buf,"%11.6f",tim.transTime(t));
  out.PosText(10,23,buf,0x17);
}
// Anzahl der hintereinander verlorenen Trigger
// anzeigen
sprintf(buf,"%1d",nTriggerLost);
out.PosText(29,23,buf,(nTriggerLost == 0) ? 0x17 :
        0x1C);
// PC-Uhrfrequenzfehler und Zeitstempel für den Empfang
// des letzten LA-Datenpakets anzeigen
if ((phase == PhFirstLa) || (phase == PhNormal)) {
  sprintf(buf,"%+.3E",PCCorrection-1);
  out.PosText(40,23,buf, 0x17);
  if (tim.diffTime(t,tsLA)>=0) {
    sprintf(buf,"%11.6f",tim.transTime(tim.diffTime(t,
            tsLA)));
    out.PosText(10,22,buf, 0x17);
  }
}

sprintf(buf,"%10lu",fifoTailLa*4);
out.PosText(10,21,buf, 0x17);
unsigned long t2 = tim.getTime();
// Momentanen Stand der PC-Uhr anzeigen
sprintf(buf,"%11.6f %11.6f",tim.transTime(t2),
        tim.transTime(tim.diffTime(t,t2)));
out.PosText(32,22,buf, 0x17);

if (phase == PhFatal)  return -1;

return 0;
}

//- int Sys::Pulse(LaData *h1, LaData *h2, int LaPort,
//                 int ComPort,int SavePort)
//
// Verrechnung von Pulsen und GPS-Nachrichten eines
// Kanals
// h1:       betrachtetes LA-Paket
// h2:       LA-Paket nach dem betrachteten
// LaPort:   zu bearbeitender Kanal am LA
// ComPort:  Kanalnummer auf 8fach RS232-Schnittstellen-
//           karte für zu verarbeitende GPS-Nachrichten
// SavePort: Kanalnummer für die Speicherung

int Sys::Pulse(LaData *h1,LaData *h2,int LaPort,
               int ComPort,int SavePort) {
  int jumpy;
  int nopuls;
  int trans;
  int i;
  unsigned long t0,t1,t2,tg;
  GpsData *h;

  // Liegen überhaupt Transitionen des Kanals
  // >LaPort< vor ?
  nopuls=0;
  trans = h1->trans[LaPort];
  if (trans == 0) {
```

```
    // Keine Transitionen -> Pseudopulse einführen um
    // GPS-Messages nicht zu verlieren
    nopuls = 1;
    trans = 4;
  }
  if (trans > 0) {
    // Test auf Jumpy Channel (Zuviele (>20)Transitionen
    // für 4~s-LA-Datenpaket
    jumpy = 0;
    if (trans>MaxTransitions) {
      trans = MaxTransitions;
      jumpy = SaveJumpyChannel;
    }
    // Gehe alle Transitionen des Kanals >LaPort< durch
    for (i=0; i<trans; i++)  {
      double offset = -999999999.0;
      unsigned long second = InvalidLong;
      int delay1 = InvalidWord;
      int delay2 = InvalidWord;
      long free = InvalidLong;
      long free2 = InvalidLong;
      long free3 = InvalidLong;
      long free4 = InvalidLong;
      unsigned long second2 = InvalidLong;
      // Jeder Puls eines Pakets
      // erhält ein gesetztes Jumpy-Flag, falls
      // dieses für das Paket gesetzt ist
      unsigned flags = jumpy;
      if (!nopuls) {
        // Berechne Zeitstempel nach PC-Uhr
        // für betrachtete Transition
        t1 = (unsigned long) ((h1->data[LaPort][i]-
            h1->refTrigger)/1000000000.0*PCCorrection
            /PCTick+(double) h1->tsTrigger);

        if (i==trans-1) {
          t2 = t1+0x40000000;
          if (h2 != 0) {
            if (h2->trans[LaPort]>0) {
              t2 = (unsigned long) ((h2->data[LaPort][0]-
                 h2->refTrigger)/1000000000.0*PCCorrection
                 /PCTick+(double) h2->tsTrigger);
            }
          } else {
            // Triggerende als Pulsgrenze
            t2 = (unsigned long) ((4.0*PCCorrection/PCTick)
                 + (double) h1->tsTrigger);
          }
        } else {
          t2 = (unsigned long) ((h1->data[LaPort][i+1]-
              h1->refTrigger)/1000000000.0*PCCorrection
              /PCTick+ (double) h1->tsTrigger);
        }
        offset = h1->data[LaPort][i];
      } else {
        switch(i) {
          case 0:
            t1 = h1->tsTrigger;
            t2 = (unsigned long) ((1.0*PCCorrection/PCTick)
                 + (double) h1->tsTrigger);
            break;
          case 1:
            t1 = (unsigned long) ((1.0*PCCorrection/PCTick)
                 + (double) h1->tsTrigger);
            t2 = (unsigned long) ((2.0*PCCorrection/PCTick)
                 + (double) h1->tsTrigger);
            break;
          case 2:
            t1 = (unsigned long) ((2.0*PCCorrection/PCTick)
                 + (double) h1->tsTrigger);
            t2 = (unsigned long) ((3.0*PCCorrection/PCTick)
                 + (double) h1->tsTrigger);
            break;
          case 3:
            t1 = (unsigned long) ((3.0*PCCorrection/PCTick)
                 + (double) h1->tsTrigger);
            t2 = (unsigned long) ((4.0*PCCorrection/PCTick)
                 + (double) h1->tsTrigger);
        }
      }
      sprintf(buf,"%11.6f",tim.transTime(h1->tsTrigger));
      out.PosText(1,0,buf, 0x17);
      // Hole alle passenden Gps-Messages
      while (1) {
        // Noch eine Message vorhanden: Anschauen
        if ((h = comGPS.LookGpsData(ComPort)) != 0) {
          tg = h->tsTimestamp;
          // Zeitstempel größer als der von Transition
          // 10ms Vorlauf
          if (tim.diffTime(t1,tg)>=-12500) {
            // Zeitstempel aber kleiner als der der
            // nächsten Transition
            if (tim.diffTime(t2,tg)<-12500)  {
              // Message tatsächlich entnehmen
              h = comGPS.GetGpsData(ComPort);
              // Enthält Time-Informationen
              if (second == InvalidLong) {
                second = h->second;
                 flags |= ((h->flags) & 0xFF00);
              }
              if (second2 == InvalidLong) {
                second2 = h->second2;
              }
              if (free2 == InvalidLong) {
                free2 = h->free2;
              }
              if (free3 == InvalidLong) {
                free3 = h->free3;
              }
              if (free4 == InvalidLong) {
                free4 = h->free4;
              }
              if (free == InvalidLong) {
                free = h->free;
              }
              if (h->flags & GpsDataValid)
                flags |= (SaveDataValid);
              if (h->flags & GpsReceiverFault)
                flags |= (SaveReceiverFault);
              if (h->flags & GpsAntennaFault)
                flags |= (SaveAntennaFault);
              if ((delay1 == InvalidWord) &&
                  (h->flags & GpsMessage1)) {
                if (h->flags & GpsCorruptedMessage1)
                  flags |= SaveCorruptedMessage1;
                delay1 = 1000*(tim.diffTime(t1,tg)*PCTick);
              }
              if ((delay2 == InvalidWord) &&
                   (h->flags & GpsMessage2)) {
                if (h->flags & GpsCorruptedMessage2)
                  flags |= SaveCorruptedMessage2;
                delay2 = 1000*(tim.diffTime(t1,tg)*PCTick);
              }
            } else break;
          } else {
            // Vernichte GPS-Message (weil zu alt)
            h = comGPS.GetGpsData(ComPort);
          }
        } else  break;
      }
      // Pulsdaten eines Pakets in vier Teile
      // aufteilen und zur Speicherung an
      // save-Objekt übergeben
      if (nopuls) {
          if ((delay1 != InvalidWord) ||
              (delay2 != InvalidWord)) {
            saveData.NewPulse(SavePort,h1->second+i,
               InvalidLong, second, delay1, delay2,flags,
               free,second2,free2,free3,free4);
          }
      } else {
        if (offset>3000000000.0+h1->refTrigger) {
          saveData.NewPulse(SavePort,h1->second+3,
             (long) (offset-3000000000.0), second,
             delay1, delay2,flags,free,second2,free2,
             free3,free4);
        } else if (offset>2000000000.0+h1->refTrigger) {
          saveData.NewPulse(SavePort,h1->second+2,
```

```
              (long) (offset-2000000000.0), second,
              delay1, delay2,flags,free,second2,free2,
              free3,free4);
         } else if (offset>1000000000.0+h1->refTrigger) {
            saveData.NewPulse(SavePort,h1->second+1,
              (long) (offset-1000000000.0), second,
              delay1, delay2,flags,free,second2,free2,
              free3,free4);
         } else {
            saveData.NewPulse(SavePort,h1->second,
              (long) (offset), second,
              delay1, delay2,flags,free,second2,free2,
              free3,free4);
         }
       }
    } // nächste Transition
  }
  return 0;
}


//- Sys::CheckTimeOut
//.
//. Diese Methode bewerkstelligt folgendes:
//.
//. 1. Anhängen eines leeren LA-Datenpakets in das
//.    LA-FIFO 4~s nach Auftreten des Triggerinterrupts
//. 2. Erzeugen eines fiktiven Triggerinterrupts durch
//.    Extrapolation der Triggerzeitstempel, falls
//.    Triggerinterrupt nicht erzeugt wird.

int Sys::CheckTimeOut() {
  unsigned long t;
  double refWait;
  // Interrupts sperren
  disable();
  // Zeitstempel holen
  t = tim.getTimed();
  // Je nach Systemzustand anders vorgehen:
  switch (phase) {
  case PhNormal:
  case PhFirstLa:
    // Zeit seit letztem Trigger berechnen
    // (Sekundenwert innerhalb der Triggerperiode)
    // und PC-Uhrenfehlerkorrektur anbringen
    refWait = tim.diffTime(tsTrigger2,t)*PCTick
       /PCCorrection;
    // Konsistenz der Timerwerte testen
    if (refWait<0) nTimerFail++;
    // Kein Trigger gekommen ? (Dauer > 8 Sekunden)
    if (refWait>=8.0+refEpsilonTrigger) {
      // Trigger simulieren
      while (refWait>=8.0+refEpsilonTrigger) {
        nTriggerLost++;
        // Zeitstempel extrapolieren
        tsTrigger1 = tsTrigger2;
        tsTrigger2 += (unsigned long) ((8.0/PCTick)
            *PCCorrection);
        refWait = tim.diffTime(tsTrigger2,t)*PCTick
            /PCCorrection;
        // Neues LA-Datenpakte anhängen, genauso wie
        // im Falle eines Triggerinterrupts
        // (siehe IO.CPP)
        fifoTailLa &= 0xFFFFFFFE;
        LaData *h = &laData[fifoTailLa % LaFifoLen];
        h->tsTrigger = tsTrigger1;
        h = &laData[(fifoTailLa+1) % LaFifoLen];
        h->tsTrigger = tsTrigger1+tim.diffTime(
                          tsTrigger1,tsTrigger2)/2;
        fifoTailLa +=2;
      }
      // Wurde zuoft hintereinander der Trigger
      // verloren ?
      // -> Fataler Abbruch
      if (nTriggerLost>MaxTriggerLost) {
        phase = PhFatal;
        gerr = Gerr_TooMuchTriggerLost;
        enable();
        return -1;
      }
    }
    // Wurde die zweite der Hälfte der Triggerperiode
    // erreicht -> leeres LA-Datenpaket anhängen
    if (refWait>=4.0)
      fifoTailLa |= 1;
    LaData *h = &laData[fifoTailLa];
    // Fehlverhalten des Triggers
    // in zugehörigem LA-Paket vermerken
    // Trigger verloren
    if (nTriggerLost>0) h->state |= SysTriggerLost;
    // Trigger fehlerhafte
    if (nTriggerErr>0)  h->state |= SysTriggerErr;
    // Trigger verloren
    if (nTimerFail>0)  h->state |= SysTimerFail;
    break;
  case PhTrigger1:
  case PhTrigger2:
    // Triggertimeout für 1. und 2. Trigger
    // Toleranz geringer, da noch kein
    // PC-Uhrfrequenzfehler vorliegt
    if (tim.diffTime(tsTrigger1,t)*PCTick>8.01) {
      phase = PhFatal;
      gerr = Gerr_Trigger12TimeOut;
      enable();
      return -1;
    }
    break;
  }
  // Merke gegenwärtige Fifoposition (wegen Interrupt)
  unsigned long ta = fifoTailLa;
  // Interrupt wieder zulassen
  enable();

  // Prüfe, ob Daten von den Logikanalysatoren
  // vorliegen
  switch(phase) {
  case PhNormal:
  case PhFirstLa:
    // zwischen Sekunde 4.1 und 7.0 einer
    // Triggerperiode Daten vom LA mit Triggerlevel 0
    // akzeptieren
    if (refWait>=7.0) {
    } else if (refWait>=4.1) {
      logicAnalyser.CalculateOnData(0,ta);
    // zwischen Sekunde 0.1 und 3.0 einer
    // Triggerperiode Daten vom LA mit Triggerlevel 1
    // akzeptieren
    } else if (refWait>=3.0) {
    } else if (refWait>=0.1) {
      logicAnalyser.CalculateOnData(1,ta);
    }
    // Im Sekundenrhythmus Anzeigen der Pulsdaten
    // durchführen
    unsigned long s = ((unsigned long) refWait)+
       ((ta & 0xFFFFFFFE)*4);
    if (oSecond<=s) {
      if (s>=16) saveData.Show(s-16);
      oSecond = s+1;
    }
    break;
  case PhTrigger1:
  case PhTrigger2:
    logicAnalyser.CalculateOnData(0,0);
    break;
  }
  return 0;
}
```

B.2.2 Vorverarbeitung der LA-Daten

Header LA.H

```
#ifndef __LA_H    /* prevent multiple includes */
#define __LA_H
```

```
// Anzahl maximal zu berücksichtigender Kanäle
#define Channels      16
// Anzahl maximal auftretender Transitionen (Flanken)
// pro Kanal
#define MaxTransitions 20

// Objekt zur Beschreibung eines Datenpakets

class LaData {
public:
  unsigned valid;    // Gültigkeit der LA-Daten
                     // (LaValid oder LaInValid)
  unsigned state;
  // Zeitstempel der Transitionen der einzelnen Kanäle
  // bezüglich Rb-Zeit
  double data[Channels][MaxTransitions];
  // Anzahl der in dem LA-Paket für diesen Kanal
  // aufgetretenen Transitionen
  int trans[Channels];
  // Zeitstempel der die Aufzeichnung des LA-Pakets
  // auslösenden Triggerflanke
  unsigned long tsTrigger;
  double refTrigger;
  unsigned long second;
  LaData();
  ~LaData();
};

//- Klasse La, zur Bedienung der LAs und der Verrechnung
//. der Daten

class La {
  int init;
  unsigned long lastread;
public:
  struct HP_Info_Data *info;
  La();
  ~La();
  int Initialize();
  int Start();
  int CalculateOnData(int state, unsigned long lfifo);
};

#endif
```

Modul LA.CPP

```
//: LA.CPP
//. Modul zur Verrechnung der Daten des Logikanalysators
//.
//.

#include <stdio.h>
#include <dos.h>
#include "timer.h"
#include "la.h"
#include "hp16500.h"
#include "global.h"


//- Convenience-Funktion, zur Umrechnung eines als
//  8 Byte Integer (signed) gegebenen Zeitstempels
//  (mit Pikosekunden-Auflösung) auf Double (in
//  (Nanosekunden)

double time(char far * tm) {
  double t;
  if ((((unsigned char) tm[0])& 0x80) == 0) {
    t = (unsigned char) tm[0];
    t = (t*256) + (unsigned char) tm[1];
    t = (t*256) + (unsigned char) tm[2];
    t = (t*256) + (unsigned char) tm[3];
    t = (t*256) + (unsigned char) tm[4];
    t = (t*256) + (unsigned char) tm[5];
    t = (t*256) + (unsigned char) tm[6];
    t = (t*256) + (unsigned char) tm[7];
  } else {
    t =  (unsigned char) ~tm[0];
    t = (t*256) + (unsigned char) ~tm[1];
    t = (t*256) + (unsigned char) ~tm[2];
    t = (t*256) + (unsigned char) ~tm[3];
    t = (t*256) + (unsigned char) ~tm[4];
    t = (t*256) + (unsigned char) ~tm[5];
    t = (t*256) + (unsigned char) ~tm[6];
    t = (t*256) + (unsigned char) ~tm[7];
    t = -t-1;      }
  return (t/1000);
}


//- Initalisierungsstruktur für die Logikanalysatoren

static struct HP_Initstrc InitStructure = {
  HP_LOAD_CONFIG | HP_HARDDISK | HP_VIEW_MENUS, // flags
  21,                   // controller address
  7,                    // HP address
  40000,                // buffer length
  "/GPS",               // directory
  "GPSALL.___", // configuration file name
  3,                    // logic analyser 1 module number
  4,                    // logic analyser 2 module number
  1,                    // pattern generator module number
};


//- Konstruktor LaData
//.
//. Objekt zur Beschreibung eines LA-Datenpaktes

LaData::LaData() {
  valid = LaInValid;
  state = 0;
}

//- Destruktor LaData

LaData::~LaData() {
}


//- Konstruktor La
//.
//. Klasse zur Bearbeitung der LA-Daten
//. Es wird bei der Initialisierung ein Zeitstempel
//. lastread geholt. Die Bedeutung diese Zeitstempels
//. ist der Methode CalculateOnData zu entnehmen.

La::La() {
  init = 0;
  // ersten Zeitstempel generieren
  lastread = tim.getTime();
}

//- Initialize
//.
//. Initialisierung der Logikanalysatoren

int La::Initialize() {
  init = 1;
  int i = HP_Init ((struct HP_Initstrc far *) &InitStructure) ;
  return i;
}

//- Start
//.
//. Starten der Logikanalysatoren

int La::Start() {
  return HP_Start();
}

//- Destruktor La
//.

void La::~La() {
```

```
  if (init != 0) {
    HP_End();
  }
}

//- La::CalculateOnData
//.
//.

int La::CalculateOnData(int state,unsigned long lfifo) {
  int gotdata;
  // Zeitstempel holen
  unsigned long t = tim.getTime();
  // GetData darf nur mit gewissem zeitlichen Abstand
  // aufgerufen werden
  // Hier werden ca. 320 ms als Mindestwartezeit benutzt
  if (tim.diffTime(lastread,t)<400000L) return 0;
  gotdata = HP_GetData(&info);
  lastread = tim.getTime();
  // Liegen neue LA-Daten vor ?
  if (gotdata == 0) {
    if (info) {
      // Zeitstempel für dieses LA-Paket speichern
      tsLA = lastread;
      // Interrupt sperren,
      // damit Triggerinterrupt nicht reinschlägt
      disable();
      // Falls noch auf 1. und 2. Trigger
      // gewartet wird La-Daten ignorieren
      if ((phase == PhTrigger1) ||
        (phase == PhTrigger2)) {
        enable();
        return 0;
      }
      if (phase == PhFirstLa) {
        phase = PhNormal;
      }
      // Zugehörige Position im LA-Fifo für
      // das neue Paket bestimmen
      LaData   *h = &laData[lfifo % LaFifoLen];
      // Vorerst Paket als ungültig markieren
      h->valid = LaInValid;
      h->state = 0;
      enable();
      // Felder zur Beschreibung der aufgetretenen
      // Übergänge. Tritt auf (auch nur) einem der
      // Kanäle eine Flanke auf, so wird ein Zeitstempel-
      // wert (8 ns Granularität) und der Pegel aller Kanäle
      // augzeichnet.
      // Für jedes solches Ereignis wird ein
      // Eintrag im Feld ad (Pegelwerte) und
      // im Feld td (Zeitstempel) erzeugt
      struct acqu_data far *ad = info->acq_data;
      struct time_data far *td = info->tim_data;
      // Anzahl der aufgezeichneten Einträge
      int dlen = info->datlen_pod1;
      // Index der auslösenden Triggerflanke
      int ind  = info->trgpnt_pod1;

      int indTriggerStart,indTriggerEnd;
      char stTrigger;
      int i;

      // Anzahl der Transitionen für alle Kanäle
      // auf Null initialisieren
      for (i=0; i<Channels; i++)
        h->trans[i] = 0;

      // Genug Daten vorhanden ?
      if (ind<4 || ind>dlen-4) {
        return 0;
      }
      // Die Berechnung des Indexes des
      // Triggers in der La-Struktur:
      // Der übergebene Index ind ist leider nicht
      // exakt. Daher werden vier Einträge
      // um den Trigger betrachtet und der
      // korrekte Index des Triggers bestimmt
      // Der Trigger wird auf Kanal 2 mitgesamplet
      char s1 = ((ad[ind-2].pod1_0_7 & 0x4)>>2);
      char s2 = ((ad[ind-1].pod1_0_7 & 0x4)>>2);
      char s3 = ((ad[ind].pod1_0_7 & 0x4)>>2);
      char s4 = ((ad[ind+1].pod1_0_7 & 0x4)>>2);

      // Kein Triggerübergang konnte festgestellt !
      if (s1==s4) {
        h->state |= SysLaNoTrigger;
        return 0;
      }
      // Der Trigger verhält sich irregulär
      // (Mehr als eine Flanke)
      if ((s1!=s2) && (s3 !=s4)) {
        h->state |= SysLaIrregularTrigger;
        return 0;
      }
      if (s1==s2)
        if (s2==s3)
          indTriggerStart = ind+1;
        else
         indTriggerStart = ind;
      else
        indTriggerStart = ind-1;

      // Neuer Pegel der Triggerleitung nach dem Trigger
      stTrigger = s4;
      stLA = stTrigger;

      // Der Pegel des Triggers ist nicht der erwartete !
      if (state != stTrigger) {
        h->state |= SysLaWrongTriggerState;
        return 0;
      }

      // Zeitstempel des Triggers
      double tslTriggerStart = time(td[indTriggerStart].timebytes);
      double tslNullRef;
      double tslFirstRef = 0;
      double tslSecondRef = 0;

      // Suche ersten und zweiten Ref100-Puls
      int number = 0;
      unsigned m1,m2,m;
      for (i=indTriggerStart+1; i< dlen; i++) {
        m1 = (((unsigned char) ad[i-1].pod1_8_15) << 8)
          + ((unsigned char) ad[i-1].pod1_0_7);
        m2 = (((unsigned char) ad[i].pod1_8_15) << 8)
          + ((unsigned char) ad[i].pod1_0_7);
        // Steigende Flanke verwenden
        m = ((m1^m2) & m2);
        if (m & 1) {
          number++;
          if (number == 1) {
            tslFirstRef = time(td[i].timebytes);
          } else if (number == 2) {
            tslSecondRef = time(td[i].timebytes);
            break;
          }
        }
      }
      // Weniger als 2 REF100-Pulse !
      if (number<2) {
        h->state |= SysLaWrongNumberRef100;
        return 0;
      }
      tslNullRef = tslFirstRef-(tslSecondRef-tslFirstRef);

      indTriggerEnd = 0;
      int Ref100 = -1;
      double tslDiffRef = tslSecondRef-tslFirstRef;
      double tslLastRef = tslNullRef;
      double help;
      int j;

      // Von der startenden Triggerflanke bis zur
      // beendenden Triggerflanken, Einträge
      // durchlaufen und Zeitstempel durch
      // lineares Inter- und Extrapolieren auf
```

```
      // die durch REF100-Pulse (Rb-Zeit) definierte
      // Zeitbasis umrechnen
      for (i=indTriggerStart+1; i<dlen; i++) {
        m1 = (((unsigned char) ad[i-1].pod1_8_15) << 8)
           + ((unsigned char) ad[i-1].pod1_0_7);
        m2 = (((unsigned char) ad[i].pod1_8_15) << 8)
          + ((unsigned char) ad[i].pod1_0_7);
        help = time(td[i].timebytes);
        // Richtige Flanke verwenden
        m = ((m1^m2) & m2 & MaskRising) |
          ((m1^m2) & m1 & MaskFalling);
        if (m & 1) {
          tslDiffRef = help - tslLastRef;
          tslLastRef = help;
          Ref100++;
        }
        for (j=1; j<Channels; j++) {
          if ((m & (1 << j)) != 0) {
            if (h->trans[j]<MaxTransitions) {
               h->data[j][h->trans[j]++] =
                 (Ref100*10000000.0)+
                   (help-tslLastRef)*10000000/tslDiffRef;
            } else {
              h->trans[j]++;
            }
          }
        }
        if (((m2 & 0x4)>>2) != stTrigger) {
          indTriggerEnd = i;
          break;
        }
      }

      // Nicht genau 400 Pulse vorhanden (0..399) !
      if (Ref100 != 399) {
        h->state |= SysLaWrongNumberRef100;
        return 0;
      }

      // Keine beendende Triggerflanke vorhanden !
      if (indTriggerEnd == 0) {
        h->state |= SysNoTriggerEnd;
        return 0;
      } else {
        double tslTriggerEnd = time(
                 td[indTriggerEnd].timebytes);
        h->refTrigger = (tslTriggerStart-tslFirstRef)
          *10000000.0/(tslSecondRef-tslFirstRef);
        if (stTrigger == 1) {
          LA1Correction = (tslTriggerEnd-
          tslTriggerStart)/4000000000.0;
        } else {
          LA2Correction = (tslTriggerEnd-
          tslTriggerStart)/4000000000.0;
        }
      }
      disable();
      // La-Daten als gültig markieren
      h->valid = LaValid;
      h->state = 0;
      enable();
    }
  }
  return 0;
}
```

B.2.3 Analyse der GPS-Nachrichten

Header COM.H

```
#ifndef __COM_H    /* prevent multiple includes */
#define __COM_H

#include "serialbo.h"

// Buffer für Parserzeichenstrom
#define LenGPSFifo 70
#define LenParserBuffer 256

//- Objekt zur Beschreibung einer GPS-Nachricht

class GpsData {
public:
  unsigned long second;       // Sekundenwert
  unsigned long second2;      // Zweiter Sekundenwert
                              // (nur beim Stellar
                              // verwendet)
  long          free;         // freie verwendbare
                              // Variablen
  long          free2;
  long          free3;
  long          free4;
  unsigned      flags;       // GPS-Flags
                             // (Siehe GLOBAL.H)
  unsigned long tsTimestamp; // PC-Zeitstempel für den
                             // empfang des ersten
                             // Zeichens
  GpsData();
  ~GpsData();
};

enum ParserType {
  parInactive=0,
  parStellar=1,
  parNavSymm=2,
  parMotorola=3,
  parTrimble=4,
  parMagellan=5,
  parRockwell=6,
  parServer=7};

class Parser {
  int type;
  int phase;
  int index;
  int free;
  unsigned cksm,fcksm;
  int a;
  int Sign;
  char checksum;
  char checksum2;
  char fcheck;
  char fcheck2;
  int fifoTail;
  int fifoHead;
  unsigned long data;
  unsigned long data2,data3,data4;
  unsigned long r;
  GpsData fifo[LenGPSFifo];
  char buffer[LenParserBuffer];
public:
  Parser();
  ~Parser();
  int Parse(char c, unsigned long t);
  int Initialize(int Type);
  GpsData *GetGpsData();
  GpsData *LookGpsData();
  int GetChar(char Character, char c, int nextphase);
  int GetWord(char *string, char c, int nextphase);
  int GetNumber(int len, char end, char c,
int nextphase);
  int GetSigned(int len, char end, char c,
int nextphase);
  int GetHex(int len,char c,int nextphase);
  int GetBuffer(int len,char c,int nextphase);
  int GetToChar(int len, char end, char c,
int nextphase);
  double RockwellEFP(char * adr);
  unsigned long DateToSecond(unsigned char hour,
     unsigned char minute,
     unsigned char second,
     unsigned char day,
     unsigned char month,
     unsigned year);
  long CharToLong(char *adr);
```

```
};

class Com {
  int init;
  boca BB1008;
  Parser pars[8];
  int p2;
  char c2;
public:
  port  *com[8];
  Com();
  ~Com();
  int Synchron();
  char Checksum(char * pre,int len);
  int MessageMotorola(const char *adr, int Len);
  int MessageMagellanA(const char *adr, int Len);
  int MessageMagellanB(const char *adr, int Len);
  int InitReceiver(int number);
  int Initialize(int Address, int Irq);
  int LongToChar(long t,char *adr);
  GpsData *GetGpsData(int Port);
  GpsData *LookGpsData(int Port);
};

#endif
```

Modul COM.CPP

```
//: COM.CPP
//.
//. Modul zur Kommunikation mit
//. den GPS-Empfängern über die 8-fach serielle
//. Schnittstellenkarte von BOCA

#include <stdio.h>
#include <string.h>
#include <math.h>
#include "com.h"
#include "global.h"

int posi = 0;
int g=0;
unsigned long r=0;


//- Konstruktor Parser
//.
//. Ein Objekt, dieser Klasse analysiert einen
//. Zeichenstrom nach einem der Protokolle der
//. GPS-Empfänger und erzeugt Beschreibungsstrukturen
//. GpsData, die in den GPS-FIFO fifo des jeweiligen
//. Parsers geschoben werden.

Parser::Parser() {
  type = parInactive;
  fifoTail = 0;
  fifoHead = 0;
  index= 0;
  checksum = 0;
  fcheck = 0;
  phase = 0;
  data = 0;
  data4 = 0;
}

//- Destruktor Parser

Parser::~Parser() {
}

//- Parser::Initializer
//.
//. Mit dieser Methode wird das zu verwendende Protokoll
//. ausgewählt

int Parser::Initialize(int Type) {
  type = Type;
  r = 0;
  phase = 0;
  return 0;
}

//- Parser:Parse
// Herzstück des Parsers
// Messages der GPS-Empfänger analysieren

int Parser::Parse(char c, unsigned long t) {
  GpsData *h;
  unsigned long hReference;
  int hSign;

  h = &fifo[fifoTail];
  switch (type) {
  // --- Nachrichten des NavSymm (GPS und SFM)
  case parNavSymm:
    checksum ^= c;
    switch(phase) {
    case -1:
    case 0:
      if (index == 0) {
        h->tsTimestamp = t;
        h->second = InvalidLong;
        h->free = InvalidLong;
        h->flags = 0;
       }
       if (GetWord("$PTELP,",c,1) != 0) {
         phase = -1;
       }
       break;
    case 1:
      if (c=='G') {
        phase = 2;
      } else if (c=='S') {
        phase = 15;
      }
      break;
    case 2:
      if (GetWord("PS,",c,3) != 0) {
        phase = -1;
      }
      break;
    case 3:
      h->flags |= GpsMessage1;
      GetToChar(10,',',c,4);
      break;
    case 4:
      GetToChar(10,',',c,5);
      break;
    case 5:
      GetToChar(10,',',c,6);
      break;
    case 6:
      GetToChar(10,',',c,7);
      break;
    case 7:
      GetToChar(10,',',c,8);
      break;
    case 8:
      GetToChar(10,',',c,9);
      break;
    case 9:
      GetToChar(10,',',c,10);
      break;
    case 10:
      GetToChar(10,',',c,11);
      break;
    case 11:
      GetToChar(10,',',c,12);
      break;
    case 12:
      GetNumber(7,',',c,13);
      if (phase == 13) data2 = data;
      break;
    case 13:
      if (GetNumber(7,char(13),c,14) != 0) {
        phase = -1;
```

```
      }
      if (phase == 14)  h->second =
        DateToSecond(data2 / 10000, (data2 / 100) % 100,
                     data2 % 100, data /10000,
                     (data /100) % 100,(data % 100)+1900);
        break;
    case 14:
      if (c==char(10)) {
        phase = 0;
        fifoTail++;
        fifoTail %= LenParserFifo;
      } else
        phase = -1;
      break;
    case 15:
      if (GetWord("FM,",c,16) != 0) {
        phase = -1;
      }
      break;
    case 16:
      h->flags |= GpsMessage2;
      GetToChar(10,',',c,17);
      break;
    case 17:
      GetToChar(10,',',c,18);
      break;
    case 18:
      GetToChar(10,',',c,19);
      break;
    case 19:
      GetToChar(10,',',c,20);
      break;
    case 20:
      GetToChar(10,',',c,21);
      break;
    case 21:
      GetToChar(10,',',c,22);
      break;
    case 22:
      if (GetNumber(7,',',c,23) != 0) {
        phase = -1;
      }
      if (phase == 23) {
        if (data != 0) {
          h->flags |= GpsDataValid;
        }
        h->free  = data & 0xFFFF;
      }
      break;
    case 23:
      GetToChar(10,',',c,24);
      break;
    case 24:
      GetToChar(10,',',c,25);
      break;
    case 25:
      GetToChar(10,',',c,26);
      break;
    case 26:
      GetToChar(10,',',c,27);
      break;
    case 27:
      GetToChar(10,char(13),c,14);
      break;
    }
    break;
    // --- Nachrichten des Stellar (STATUS und TAF)
  case parStellar:
    checksum ^= c;
    switch(phase) {
    case -1:
    case  0:
      if (GetChar('$',c,1) != 0) {
        phase = -1;
      } else {
        checksum = 0;
        h->tsTimestamp = t;
        h->second = InvalidLong;
        h->free = InvalidLong;
        h->second2 = InvalidLong;
        h->free2 = InvalidLong;
        h->free3 = InvalidLong;
        h->free4 = InvalidLong;
        h->flags = 0;
      }
      break;
    case 1:
      if (c=='S') {
        phase = 2;
        // Message Erkannt:
        h->flags |= GpsMessage1;
      } else if (c=='T') {
        phase = 20;
        h->flags |= GpsMessage2;
      } else {
        phase = -1;
      }
      break;
    case 2:
      if (GetWord("TATUS,",c,3) != 0) {
        phase = -2;
      }
      break;
    case 3:
      if (GetNumber(6,',',c,4) != 0) {
        phase = -2;
      }
      if (phase ==4) {
        h->second = data*604800L;
      }
      break;
    case 4:
      if (GetNumber(11,',',c,5) != 0) {
        phase = -2;
      }
      if (phase ==5) {
        h->second += data /100;
        h->flags |= (data % 100)<<8;
      }
      break;
    case 5:
      if (c=='G') {
        phase = 6;
      } else if (c=='U') {
        phase = 7;
      } else {
        phase = -2;
      }
      break;
    case 6:
      if (GetWord("PS,DATA ",c,8) != 0) {
        phase =-2;
      }
      break;
    case 7:
      if (GetWord("TC,DATA ",c,8) != 0) {
        phase =-2;
      }
      break;
    case 8:
      if (c=='V') {
        h->flags |= GpsDataValid;
        phase = 9;
      } else if (c=='I') {
        phase = 10;
      } else {
        phase = -2;
      }
      break;
    case 9:
      if (GetWord("ALID,TRACKING ",c,11) != 0) {
        phase =-2;
      }
      break;
    case 10:
      if (GetWord("NVALID,TRACKING ",c,11) != 0) {
        phase =-2;
      }
```

```
    break;
  case 11:
    if (GetNumber(3,' ',c,12) != 0) {
      phase = -2;
    };
    break;
  case 12:
    if (GetWord("SVs,GPS DB ",c,13) != 0) {
      phase =-2;
    }
    break;
  case 13:
    if (c=='I') {
      phase = 14;
    } else if (c=='C') {
      phase = 15;
    } else {
      phase = -2;
    }
    break;
  case 14:
    if (GetWord("NCOMPLETE,FAULT ",c,16) != 0) {
      phase =-2;
    }
    break;
  case 15:
    if (GetWord("OMPLETE,FAULT ",c,16) != 0) {
      phase =-2;
    }
    break;
  case 16:
    if (GetHex(4,c,17) != 0) {
      phase = -2;
    }
    if (phase == 17) {
      if (data&0xA000) {
        h->flags |= GpsAntennaFault;
      }
      if (data&0x0B00) {
        h->flags |= GpsReceiverFault;
      }
      fcheck = checksum;
    }
    break;
  case 17:
    if (GetChar('*',c,18) != 0) {
      phase = -2;
    }
    break;
  case 18:
    if (GetHex(2,c,19) != 0) {
      phase = -2;
    }
    if (phase == 19) {
      if (data !=fcheck) {
        phase = -2;
      }
    }
    break;
  case 19:
    if (GetWord("\015\012",c,-3) != 0) {
      phase = -2;
    }
    break;
    // TAF
  case 20:
    if (GetWord("AF,",c,21) != 0) {
      phase = -4;
    }
    break;
  case 21:
    if (GetNumber(6,',',c,22) != 0) {
      phase = -4;
    }
    if (phase ==22) {
      h->second2 = data*604800L;
    }
    break;
  case 22:
    if (GetNumber(11,',',c,23) != 0) {
      phase = -4;
    }
    if (phase ==23) {
      h->second2 += data /100;
    }
    break;
  case 23:
    if (GetSigned(12,',',c,24) != 0) {
      phase = -4;
    }
    if (phase ==24) {
      h->free =  (long) data;
    }
    break;
  case 24:
    if (GetSigned(12,',',c,25) != 0) {
      phase = -4;
    }
    if (phase ==25) {
      h->free3 = (long) (data);
    }
    break;
  case 25:
    if (GetSigned(12,',',c,26) != 0) phase = -4;
    if (phase ==26) {
      h->free2 =  (long) data;
    }
    break;
  case 26:
    if (GetSigned(12,',',c,27) != 0) phase = -4;
    if (phase ==27) {
      h->free4 = (long) data;
    }
    break;
  case 27:
    if (GetNumber(5,',',c,28) != 0) phase = -4;
    break;
  case 28:
    if (GetNumber(5,'*',c,29) != 0) phase = -4;
    if (phase == 29) fcheck = checksum ^ '*';
    break;
  case 29:
    if (GetHex(2,c,30) != 0) {
      phase = -4;
    }
    if (phase == 30) {
      if (data !=fcheck) {
        phase = -4;
      }
    }
    break;
  case 30:
    if (GetWord("\015\012",c,-3) != 0) {
      phase = -4;
    }
    break;
  }
  // Erkannte Message ist fehlerhaft
  if (phase ==-2) {
    phase = -1;
    h->flags |= GpsCorruptedMessage1;
    fifoTail++;
    fifoTail %= LenParserFifo;
  }
  if (phase ==-4) {
    phase = -1;
    h->flags |= GpsCorruptedMessage2;
    fifoTail++;
    fifoTail %= LenParserFifo;
  }
  if (phase ==-3) {
    phase = 0;
    fifoTail++;
    fifoTail %= LenParserFifo;
  }
  break;
  // Nachrichten des Motorola (Position-Status-Data)
case parMotorola:
```

```
checksum ^= c;
switch(phase) {
case -1:
case 0:
  if (c=='@') {
    phase = 1;
    h->tsTimestamp = t;
    h->second = InvalidLong;
    h->free = InvalidLong;
    h->flags = 0;
  } else {
    phase = -1;
  }
  break;
case 1:
  if (c=='@') {
    phase = 2;
    checksum = 0;
  } else {
    phase = -1;
  }
  break;
case 2:
  if (c=='B') phase = 3; else phase = -1;
                         break;
case 3:
  if (c=='a') phase = 4; else
    if (c=='n') phase = 6; else phase = -1;
                           break;
case 4:
  GetBuffer(62,c,5);
  if (phase == 5) {
    h->flags |= GpsMessage1;
    h->second = DateToSecond(
                 (unsigned char) buffer[4],
                 (unsigned char) buffer[5],
                 (unsigned char) buffer[6],
                 (unsigned char) buffer[1],
                 (unsigned char) buffer[0],
                 ((unsigned char) buffer[2])*256+
                 (unsigned char) buffer[3]);
    // Sekundenbruchteile >=0.5 -> Sekunde um 1 erhöhen
    if (((unsigned char) buffer[7])>=30) {
      h->second++;
    }

    fcheck=checksum;
    h->free = ((unsigned long) buffer[64])+
      (((unsigned long) buffer[35])<<8);
  }
  break;
case 5:
  if (GetWord("\015\012",c,-3) != 0) {
    phase = -2;
  }
  if (phase == -3) {
    if (fcheck != 0) phase= -2;
  }
  break;
case 6:
  GetBuffer(53,c,7);
  if (phase == 7) {
    h->flags |= GpsMessage2;
    if (buffer[17] == 0) {
      h->flags |= GpsDataValid;
    }
        fcheck=checksum;
  }
  break;
case 7:
  if (GetWord("\015\012",c,-3) != 0) {
    phase = -4;
  }
  if (phase == -3) {
    if (fcheck != 0) phase= -4;
  }
  break;
}
if (phase == -2) {
    phase = -1;
    h->flags |= GpsCorruptedMessage1;
    fifoTail++;
    fifoTail %= LenParserFifo;
  }
  if (phase == -4) {
    phase = -1;
    h->flags |= GpsCorruptedMessage2;
    fifoTail++;
    fifoTail %= LenParserFifo;
    }
  if (phase == -3) {
    phase = 0;
    fifoTail++;
    fifoTail %= LenParserFifo;
  }
  break;
  // Nachrichten des Trimble (
case parTrimble:
  checksum ^= c;
  switch(phase) {
  case -1:
  case 0:
    if (c=='>') {
      checksum = '>';
      phase = 1;
      h->tsTimestamp = t;
      h->second = InvalidLong;
      h->free = InvalidLong;
      h->flags = 0;
    } else {
      phase = -1;
    }
    break;
  case 1:
    if (c=='R') phase = 2; else phase = -1;
      break;
  case 2:
    if (c=='T') phase = 3; else
      if (c=='S') phase = 9; else phase = -1;
                             break;
  case 3:
    if (c=='M') phase = 4; else phase = -1;
                           break;
  case 4:
    GetBuffer(28,c,5);
    if (phase == 5) {
          h->second = DateToSecond(
                 (buffer[0]-'0')*10+(buffer[1]-'0'),
                 (buffer[2]-'0')*10+(buffer[3]-'0'),
                 (buffer[4]-'0')*10+(buffer[5]-'0'),
                 (buffer[9]-'0')*10+(buffer[10]-'0'),
                 (buffer[11]-'0')*10+(buffer[12]-'0'),
                 (buffer[13]-'0')*1000
                    +(buffer[14]-'0')*100+
                 (buffer[15]-'0')*10+(buffer[16]-'0'));
          if ((buffer[6]-'0')>=5) {
            h->second++;
          }
          h->flags |= GpsMessage1;
          h->free = (long unsigned)
            ((buffer[17]-'0')*10+(buffer[18]-'0'));
          h->free +=
            ((long unsigned) ((buffer[20]-'0')*100
                              +(buffer[21]-'0')))*1000;
          h->free +=
            ((long unsigned) (buffer[22]-'0'))*10000L;
    }
        break;
  case 5:
    if (c==';') phase = 6; else phase = -2;
                           break;
  case 6:
    if (c=='*') phase = 7; else phase = -2;
                           fcheck = checksum;
                           break;
  case 7:
    if (GetHex(2,c,8) != 0) {
      phase = -2;
```

```
    }
    if (phase == 8) {
      if (fcheck != data) {
        phase = -2;
      }
    }
    break;
  case 8:
    if (c=='<') phase = -3; else phase = -2;
                            break;
  case 9:
    if (c=='T') phase =10; else phase = -1;
                           break;
  case 10:
    GetBuffer(10,c,11);
    if (phase == 11) {
      h->flags |= GpsMessage2;
      if ((buffer[0] == '0') && (buffer[1]=='0')) {<
        h->flags |= GpsDataValid;
      }

    }
    break;
  case 11:
    if (c==';')
      phase = 12;
    else
      phase = -4;
    break;
  case 12:
    if (c=='*')
      phase = 13;
    else
      phase = -4;
    fcheck = checksum;
    break;
  case 13:
    if (GetHex(2,c,14) != 0) {
      phase = -4;
    }
    if (phase == 8) {
      if (fcheck != data) {
            phase = -4;
      }
    }
    break;
  case 14:
    if (c=='<') phase = -3; else phase = -4;
                            break;
  }
  if (phase == -2) {
    phase = -1;
    h->flags |= GpsCorruptedMessage1;
      fifoTail++;
      fifoTail %= LenParserFifo;
  }
  if (phase == -4) {
    phase = -1;
    h->flags |= GpsCorruptedMessage2;
    fifoTail++;
    fifoTail %= LenParserFifo;
  }
  if (phase == -3) {
    phase = 0;
    fifoTail++;
    fifoTail %= LenParserFifo;
  }
  break;
  // Nachricht des Magellan (K3)
case parMagellan:
  checksum ^= c;
  switch(phase) {
  case -1:
  case  0:
    if (GetWord("$$K\003",c,1) != 0) {
      phase = -1;
    }
    if (phase == 1) {
      checksum = 'K' ^ 3;
      h->tsTimestamp = t;
      h->second = InvalidLong;
      h->free = InvalidLong;
      h->flags = GpsMessage1;
    }
    break;
  case 1:
    GetBuffer(17,c,2);
    if (phase == 2) {
      h->second = DateToSecond(buffer[0],buffer[1],
                  buffer[2],buffer[3],
                  buffer[4],
                  ((unsigned char)buffer[5])*256
                  +(unsigned char) buffer[6]);
      if (buffer[7] == 1) {
        h->flags |= GpsDataValid;
      }
      unsigned a;
      a = buffer[16];
      h->flags |= (a<<8);
      fcheck = checksum;
    }
    break;
  case 2:
    phase = (c != fcheck) ? -2 : 3;
    break;
  case 3:
    if (GetChar(char(10),c,-3) != 0) {
      phase = -2;
    }
    break;
  }
  if (phase == -2) {
    phase = -1;
    h->flags |= GpsCorruptedMessage1;
    fifoTail++;
    fifoTail %= LenParserFifo;
  }
  if (phase == -4) {
    phase = -1;
    h->flags |= GpsCorruptedMessage2;
    fifoTail++;
    fifoTail %= LenParserFifo;
  }
  if (phase == -3) {
    phase = 0;
    fifoTail++;
    fifoTail %= LenParserFifo;
  }
  break;
  // Nachricht vom Server-PC (mit doppelter Prüfsumme)
  // entweder Bestätigung für Pulsdaten (ACK)
  // oder Bestätigung für Fehlpulsdaten (ACL)
case parServer:
  checksum ^= c;
  checksum2 += c;
  switch(phase) {
  case -1:
  case  0:
    if (c=='$') {
      checksum = 0;
      checksum2 = 0;
      phase = 1;
    } else {
      phase = -1;
    }
    break;
  case 1:
    if (GetWord("AC",c,2) != 0) {
      phase = -1;
    }
    break;
  case 2:
    if (c=='K') {
      phase = 3;
      hSign = 0;
    } else if (c =='L') {
      phase = 3;
      hSign = 1;
```

```
      } else
        phase =  -1;
      break;
    case 3:
      if (GetBuffer(4,c,4) != 0) {
        phase = -1;
      }
      if (phase == 4) {
        hReference = * ((long unsigned *) buffer);
        fcheck = checksum;
        fcheck2 = checksum2;
      }
      break;
    case 4:
      if (c==fcheck)
        phase = 5;
      else
        phase =-1;
      break;
    case 5:
      if (c==fcheck2)
        phase = 0;
      else
        phase =-1;
      if (phase == 0) {
        if (hSign) {
          if (newAckErr == 0) {
            newAckErr = 1;
            ackErr = hReference;
          }
        } else {
          if (newAck == 0) {
            newAck = 1;
            ack = hReference;
          }
        }
      }
      break;
    }
    break;
    // Nachricht vom Rockwell
  case parRockwell:
    if (a) {
      cksm += ((unsigned char) c) *256;
    } else {
      cksm += (unsigned char) c;
    }
    a = !a;
    switch(phase) {
    case -1:
    case 0:
      if (((unsigned char) c )==0xFF) {
        phase = 1;
        cksm = 0xFF;
        a = 1;
        h->tsTimestamp = t;
        h->second = InvalidLong;
        h->free = InvalidLong;
        h->flags = 0;
      }
      else
        phase = -1;
      break;
    case 1:
      if (((unsigned char) c)==0x81)
        phase = 2;
      else {
        phase = -1;
      }
      break;
    case 2: data = (unsigned  char) c; phase = 3; break;
    case 3:
      data += ((unsigned char) c)*256;
      if (data == 103) {
        phase = 4;
      } else {
        phase = -1;
      }
      break;
    case 4:
      GetBuffer(4,c,5);
      if (phase == 5) {
        h->flags |= GpsMessage1;
        fcksm = cksm;
      }
      break;
    case 5:
      GetBuffer(2,c,6);
      if (phase == 6) {
        cksm = (unsigned char) buffer[0] +
          ((unsigned char) buffer[1]*256);
        cksm = -cksm;
        if (fcksm ==  cksm) {
          cksm = 0;
          a=0;
          phase = 6;
        } else {
          phase = -2;
        }
      }
      break;
    case 6:
      GetBuffer(116,c,7);
      if (phase ==7) {
        fcksm = cksm;
        double hd;
        hd =RockwellEFP(buffer);
        h->second = (*((unsigned *) (buffer+6)))*604800L+(unsigned long) hd;
        h->free = fmod(hd,1) * 1000000000.0;
        if (*(int*) (buffer+8)) {
          h->flags |= GpsDataValid;
        }
      }
      break;
    case 7:
      GetBuffer(2,c,8);
      if (phase == 8) {
        cksm = (unsigned char) buffer[0] +
          ((unsigned char) buffer[1]*256);
        cksm = -cksm;
        if (fcksm ==  cksm) {
          cksm = 0;
          a=0;
          phase = -3;
        } else {
          phase = -2;
        }
      }
      break;
    }
    if (phase == -2) {
      phase = -1;
      h->flags |= GpsCorruptedMessage1;
      fifoTail++;
      fifoTail %= LenParserFifo;
    }
    if (phase == -3) {
      phase = 0;
      fifoTail++;
      fifoTail %= LenParserFifo;
    }
    break;
  }
  return 0;
}

int Parser::GetWord(char *string, char c,
                    int nextphase) {
  if (string[index++] == c) {
    if (string[index] == '\0') {
      index = 0;
      phase= nextphase;
    }
    return 0;
  } else {
    index = 0;
    return -1;
  }
```

```
}

int Parser::GetChar(char Character, char c,
                    int nextphase) {
  if (Character == c) {
    phase= nextphase;
    return 0;
  } else {
    return -1;
  }
}

int Parser::GetSigned(int len, char end, char c,
                      int nextphase) {
  if (index == 0) {
    data = 0;
    if (c=='-') {
      Sign = 1;
      index++;
      return 0;
    } else Sign = 0;
  }
  if ((c>='0') && (c<='9')) {
    data = data*10 +int(c-'0');
    index++;
    if (index>=len) {
      index = 0;
      if (Sign) data = -data;
      return -1;
    }
    return 0;
  } else {
    if (c==end) {
      index = 0;
      if (Sign) data = -data;
      phase = nextphase;
      return 0;
    } else {
      index = 0;
      return -1;
    }
  }
}


int Parser::GetNumber(int len, char end, char c,
                      int nextphase) {
  if (index == 0) data = 0;
  if ((c>='0') && (c<='9')) {
    data = data*10 +int(c-'0');
    index++;
    if (index>=len) {
      index = 0;
      return -1;
    }
    return 0;
  } else {
    if (c==end) {
      index = 0;
      phase = nextphase;
      return 0;
    } else {
      index = 0;
      return -1;
    }
  }
}


int Parser::GetToChar(int len, char end, char c,
                      int nextphase) {
  if (c==end) {
    data = index;
    index = 0;
    phase = nextphase;
    return 0;
  } else {
    buffer[index++] = c;
    if (index >=len) {
      data = index;
      index = 0;
      phase = nextphase;
      return 0;
    }
  }
  return 0;
}

int Parser::GetHex(int len, char c, int nextphase) {
  if (index == 0) data = 0;
  index++;
  if ((c>='0') && (c<='9')) {
    data = data*16+int(c-'0');
  } else if ((c>='A') && (c<='F')) {
    data = data*16+int(c-'A')+10;
  } else if ((c>='a') && (c<='f')) {
    data = data*16+int(c-'a')+10;
  } else {
    index = 0;
    return -1;
  }
  if (index>=len) {
    index = 0;
    phase = nextphase;
  }
  return 0;
}

int Parser::GetBuffer(int len, char c, int nextphase) {
  buffer[index] = c;
  index++;
  if (index>=len) {
    index = 0;
    phase = nextphase;
  }
  return 0;
}

long Parser::CharToLong(char *adr) {
  long t;
  if (adr[0]<0) {
    t = (unsigned char) (adr[0] &0x7f);
  } else {
    t = (unsigned char) (adr[0]);
  }
  t = t*256 + (unsigned char) adr[1];
  t = t*256 + (unsigned char) adr[2];
  t = t*256 + (unsigned char) adr[3];
  if (adr[0]<0) t = t | 0x8000000L;
  return t;
}

double Parser::RockwellEFP(char * adr) {
  double t,e;
  t = ((*((unsigned char *) adr+5)) &0x7F)*
    pow(2.0,-8.0);
  t += (*((unsigned char *) adr+4)) *pow(2.0,-16.0);
  t += (*((unsigned char *) adr+3)) *pow(2.0,-24.0);
  t += (*((unsigned char *) adr+2)) *pow(2.0,-32.0);
  t += (*((unsigned char *) adr+1)) *pow(2.0,-40.0);
  t += 0.5;
  e = (*((unsigned char *) adr))-128;
  t  =(double) (t*pow(2.0,e));
  if ((*(unsigned char *) (adr+5)) & 0x80) t = -t;
  return t;
}

//- Parser:DateToSecond
//.
//. Umwandlung eines kompletten Datums (mit Uhrzeit)
//. in einen GPS-Sekundenwert (Sekunden seit
//. 6.Jänner 1980)

unsigned long Parser::DateToSecond(unsigned char hour,
      unsigned char minute, unsigned char second,
      unsigned char day, unsigned char month,
      unsigned year) {
  unsigned dayOfMonth[12] =
```

```
    {0,31,61,92,122,153,184,214,245,275,306,337};
  if (month<3) {
    year--;
    month+=12;
  }
  if (year<1980) return 0;
  if ((month<3) || (month>14)) return 0;
  unsigned schalttage = (year-1980) / 4;
  unsigned days = schalttage+ dayOfMonth[month-3]+
    (year-1980)*365+day-1+29+31-5;
  return (((unsigned long) days)*86400+
        ((unsigned long) hour)*3600+
        ((unsigned long) minute)*60+
        (unsigned long) second);
}

//- Parser::GetGpsData
//.
//. Siehe Com::GetGpsData
//. Da die FIFO-Variablen (fifo, fifoHead und fifoTail)
//. im Parser sind, mußten eigene
//. Methoden zur Beschaffung der ältesten
//. GPS-Nachricht geschrieben werden

GpsData *Parser::GetGpsData() {
  GpsData *h;
  if (fifoTail == fifoHead)  return 0;

  h = &fifo[fifoHead++];
  fifoHead %= LenParserFifo;
  return h;
}

//- Parser::LookGpsData
//.
//. Siehe Com::LookGpsData
//. Da die FIFO-Variablen (fifo, fifoHead und fifoTail)
//. im Parser sind, mußten eigene
//. Methoden zur Beschaffung der ältesten
//. GPS-Nachricht geschrieben werden

GpsData *Parser::LookGpsData() {
  GpsData *h;
  if (fifoTail == fifoHead)  return 0;

  h = &fifo[fifoHead];
  return h;
}



//- Konstruktor GpsData
//.

GpsData::GpsData() {
}

//- Destruktor GpsData

GpsData::~GpsData() {
}



//- Konstruktor Com

Com::Com() {
  init = 0;
  serv[0] = '\0';
  newAck = 0;
}

//- Destruktor Com

Com::~Com() {
}

//- Com::Initialize
//.
//. Initialisieren der 8fach seriellen Schnittstelle
//. und Übermittlung der Initialisierungssequenzen
//. an die GPS-Empfänger

int Com::Initialize(int Address,int Irq) {
  if  (BB1008.Initialize(Address,Irq) == -1) return -1;
  // Alle 8 Ports nach den von den GPS-Empfängern
  // bestimmten Baudraten initialisieren
  com[0] = BB1008.getPort(1,9600,NO_PARITY,8,1);
  com[1] = BB1008.getPort(2,2400,NO_PARITY,8,1);
  com[2] = BB1008.getPort(3,9600,NO_PARITY,8,1);
  com[3] = BB1008.getPort(4,4800,NO_PARITY,8,1);
  com[4] = BB1008.getPort(5,9600,NO_PARITY,8,1);
  com[5] = BB1008.getPort(6,9600,ODD_PARITY,8,1);
  com[6] = BB1008.getPort(7,9600,NO_PARITY,8,1);
  com[7] = BB1008.getPort(8,9600,NO_PARITY,8,1);

  // Initialisierungssequezen an die Empfänger absenden
  InitReceiver(0);
  InitReceiver(1);
  InitReceiver(2);
  InitReceiver(3);
  InitReceiver(4);
  InitReceiver(5);

  // Parser für die einzelnen GPS-Empfänger auf die
  // korrekten Protokolle einstellen
  pars[0].Initialize(parStellar);
  pars[1].Initialize(parNavSymm);
  pars[2].Initialize(parMotorola);
  pars[3].Initialize(parTrimble);
  pars[4].Initialize(parMagellan);
  pars[5].Initialize(parRockwell);
  pars[7].Initialize(parServer);
  init = 1;
  return 0;
}

//- Com::InitReceiver
//.
//. Generierung der Initalisierungssequenz für
//. den Empfänger Number

int Com::InitReceiver(int Number) {
  char str[80];
  int Len;
  switch(Number) {
    case 0:
    // Stellar
      sprintf(buf,"unlogall\r");
      com[0]->PutToWriteBuf(buf,strlen(buf));
      sprintf(buf,"log status\r");
      com[0]->PutToWriteBuf(buf,strlen(buf));
      sprintf(buf,"log taf\r");
      com[0]->PutToWriteBuf(buf,strlen(buf));
      break;
    case 1:
      sprintf(buf,"$PTELS,CRR,SIV,0\015\012");
      com[1]->PutToWriteBuf(buf,strlen(buf));

      sprintf(buf,"$PTELS,CRR,SAL,0\015\012");
      com[1]->PutToWriteBuf(buf,strlen(buf));
      sprintf(buf,"$PTELS,CRR,GPS,1\015\012");
      com[1]->PutToWriteBuf(buf,strlen(buf));
      sprintf(buf,"$PTELS,CRR,SFM,1\015\012");
      com[1]->PutToWriteBuf(buf,strlen(buf));
      break;
    case 2:
      // Motorola
      Len = 3;
      MessageMotorola("Ba\001",Len);
      com[2]->PutToWriteBuf(buf,Len+5);
      Len = 17;
      MessageMotorola("Bn\001\001\000\012\001\000\000"
                      "\001\000\000\000\000\000\000\000",
                      Len);
      com[2]->PutToWriteBuf(buf,Len+5);
      Len = 3;
      MessageMotorola("AB\004",Len);
```

```
      com[2]->PutToWriteBuf(buf,Len+5);
      Len = 15;
      char st[80];
      st[0] = 'A';
      st[1] = 's';
      LongToChar(173506970L,st+2);
      LongToChar(58929509L,st+6);
      LongToChar(25318L,st+10);
      st[14] = 1;
      MessageMotorola(st,Len);
      com[2]->PutToWriteBuf(buf,Len+5);
      Len = 3;
      MessageMotorola("At\001",Len);
      com[2]->PutToWriteBuf(buf,Len+5);
      break;
    case 3:
      // Trimble
      sprintf(buf,">FTM00010000<");
      com[3]->PutToWriteBuf(buf,strlen(buf));
      sprintf(buf,">FST00010000<");
      com[3]->PutToWriteBuf(buf,strlen(buf));
      sprintf(buf,">FPV00000000<");
      com[3]->PutToWriteBuf(buf,strlen(buf));
      break;
    case 4:
      // Magellan
      Len = 19;
      MessageMagellanA("PMGLI,00,K03,2,B,00",Len);
      com[4]->PutToWriteBuf(buf,Len+6);
      Len = 9;
      MessageMagellanB("K\001\001\000\000\000\000"
                       "\000\000",Len);
      com[4]->PutToWriteBuf(buf,Len+4);
      break;
  }
  return 0;
}

//- Convenience-Methode Com::MessageMotorola
//.
//. Diese Methode versieht eine Nachricht mit
//. Header, Checksumme und Trailer nach dem
//. Kommunikationsprotokoll des Motorola-Binärprotokolls

int Com::MessageMotorola(const char *adr, int Len) {
  buf[0] = '@';
  buf[1] = '@';
  memcpy(buf+2,adr,Len);
  buf[Len+2] = Checksum(buf+2,Len);
  buf[Len+3] = '\015';
  buf[Len+4]='\012';
  return 0;
}

//- Convenience-Methode Com::MessageMagellanA
//.
//. Diese Methode versieht eine Nachricht mit
//. Header, Checksumme und Trailer nach dem
//. Kommunikationsprotokoll des Magellan-ASCIIprotokolls

int Com::MessageMagellanA(const char *adr, int Len) {
  char c;
  buf[0] = '$';
  memcpy(buf+1,adr,Len);
  c = Checksum(buf+1,Len);
  buf[Len+1] = '*';
  sprintf(buf+Len+2,"%02X",(unsigned char) c);
  buf[Len+4] = '\015';
  buf[Len+5]='\012';
  return 0;
}

//- Convenience-Methode Com::MessageMagellanA
//.
//. Diese Methode versieht eine Nachricht mit
//. Header, Checksumme und Trailer nach dem
//. Kommunikationsprotokoll des Magellan-Binärprotokolls

int Com::MessageMagellanB(const char *adr, int Len) {
  char c;
  buf[0] = '$';
  buf[1] = '$';
  memcpy(buf+2,adr,Len);
  buf[Len+2]= Checksum(buf+2,Len);
  buf[Len+3]='\012';
  return 0;
}

int Com::LongToChar(long t,char *adr) {
  adr[0] = (t>>24) & 0xFF;
  adr[1] = (t>>16) & 0xFF;
  adr[2] = (t>>8) & 0xFF;
  adr[3] = t & 0xFF;
  return 0;
}

//- Checksum
//.
//. Berechnung einer Checksumme (exklusiv oder

char Com::Checksum(char * pre, int len) {
  int i;
  char check =0;
  for (i=0; i<len; i++) {
    check^=pre[i];
  }
  return check;
}

//- Com::Synchron
//.
//. Diese Methode wird regelmäßig aufgerufen (Polling)
//. Es wird hierbei geprüft, ob Nachrichten von
//. den einzelnen Empfängern vorliegen. Ist
//. das der Fall, so wird diese den entsprechenden
//. Parsern zur Analyse übergeben, die
//. --- sofern sie eine gültige Nachricht erkannt
//. haben --- einen Eintrag im GPS-FIFO des
//. entsprechenden Empfängers erzeugen.

int Com::Synchron() {
  char c;
  unsigned long t;
  if (servLen != 0) {
    if (com[7]->PutToWriteBuf(serv,servLen) == 0) {
      servLen = 0;
    }
  }
  while (com[0]->GetFromReadBuf(c,t) == 0) {
    pars[0].Parse(c,t);
  }
  while (com[1]->GetFromReadBuf(c,t) == 0) {
    pars[1].Parse(c,t);
  }
  while (com[2]->GetFromReadBuf(c,t) == 0) {
    pars[2].Parse(c,t);
  }
  while (com[3]->GetFromReadBuf(c,t) == 0) {
    pars[3].Parse(c,t);
  }
  while (com[4]->GetFromReadBuf(c,t) == 0) {
    pars[4].Parse(c,t);
  }
  while (com[5]->GetFromReadBuf(c,t) == 0) {
    pars[5].Parse(c,t);
  }
  while (com[7]->GetFromReadBuf(c,t) == 0) {
    pars[7].Parse(c,t);
  }
  return 0;
}

//- Com::GetGPSData
//.
//. Diese Methode  entnimmt --- sofern vorhanden ---
//. dem GPS-FIFO des Empfängers am Port Port
//. die älteste GPS-Nachricht
```

```
GpsData *Com::GetGpsData(int Port) {
  return (pars[Port].GetGpsData());
}

//- Com::LookGPSData
//.
//. Diese Methode  gibt --- sofern vorhanden ---
//. die älteste GPS-Nachricht im GPS-FIFO
//. des Empfängers vom Port Port zurück, entnimmt
//. die Nachricht allerdings dem GPS-FIFO nicht.

GpsData *Com::LookGpsData(int Port) {
  return (pars[Port].LookGpsData());
}
```

B.2.4 Behandlung des Triggerinterrupts

Header IO.H

```
#ifndef __IO_H    /* prevent multiple includes */
#define __IO_H

// Adressen der PIC 1 und 2
// (Programmable Interrupt Controller)
#define ICR  0x20  // Interrupt Control Register
#define IMR  0x21  // Interrupt Mask Register
#define ICR2 0xA0
#define IMR2 0xA1


#define EOI  0x20  // End of Interrupt-Befehlsbyte

//- Klasse Io zur Behandlung der Triggerinterrupts

class Io {
  int init;
  void interrupt (*oldvect)(...); // Zeiger auf die
                                  // Interruptroutine
public:
  int irq;              // IRQ der Triggerinterruptroutine
  int address;          // Adresse der parallelen Schnitt-
                           stelle
  Io();
  ~Io();
  int Initialize(int Address,int Irq);
};

#endif
```

Modul IO.CPP

```
//: IO.CPP - Modul zur Behandlung von Triggerinterrupts

#include <stdio.h>
#include <math.h>
#include <dos.h>
#include "io.h"
#include "timer.h"
#include "global.h"
#include "out.h"

Io *object;

// Interruptroutine für den Trigger

void interrupt inter(...) {
  unsigned long t;
  long htsDiff;
  // weitere Interrupts verhindern
  disable();
  // Zeitstempel holen
  t = tim.getTimed();
  // Je nach Systemzustand reagieren
  switch(phase) {
  // 1. Trigger eingetroffen -> Zeitstempel merken
  case PhTrigger1:
    tsTrigger1 = t;
    phase = PhTrigger2;
    break;
  // 2. Trigger eingetroffen -> Zeitstempel merken
  // und in den Normalbetrieb übergehen
  case PhTrigger2:
    tsTrigger2 = t;
    // Triggerlänge berrechnen (PC-Uhrfehler noch nicht
    // korrigiert)
    htsDiff = tim.diffTime(tsTrigger1,tsTrigger2);
    // Paßt der Trigger wenigstens grob (8 s +- 10 ms) ?
    if (fabs(htsDiff*PCTick-8.000000000)>.01) {
      phase = PhFatal;
      gerr = Gerr_TriggerLength;
    } else {
      // Trigger akzeptieren und Frequenzfehler der
      // PC-Uhr merken
      PCCorrection = htsDiff*(PCTick/8.0);
      phase = PhNormal;
    }
    break;
  case PhNormal:
    // Triggerlänge ausrechnen
    htsDiff = tim.diffTime(tsTrigger2,t);
    // Konsistenz der Timerwerte testen
    if (htsDiff<0) nTimerFail++;
    // PC-Uhrenfehlerkorrektur anbringen
    TriggerLength =htsDiff*PCTick/PCCorrection;
    // Trigger genau 8s +- refEpsilonTrigger (Toleranz) ?
    if (fabs(TriggerLength-8.000000000)<
        refEpsilonTrigger){
      // Neuen Trigger akzeptieren und
      // leeres (ungültiges) LA-Datenpaket in LA-Fifo
      // anhängen
      nTriggerErr = 0;
      nTriggerLost = 0;
      // Neue Triggerzeitstempelwerte eintragen
      tsTrigger1 = tsTrigger2;
      tsTrigger2 = t;
      // Berechnung des PC-Uhrenfehlers durch
      // Kalmanfilterung:
      //   am(i+1) = am(i)*kPC + a(i)*(1-kPC)
      PCCorrection = PCCorrection*(kPC)+
         (htsDiff*(PCTick/8.0))*(1-kPC);
      // Leeres Paket an LA-FIFO anhängen
      fifoTailLa &=0xFFFFFFFE;
      LaData *h = &laData[fifoTailLa % LaFifoLen];
      h->tsTrigger = tsTrigger1;
      h = &laData[(fifoTailLa+1) % LaFifoLen];
      h->tsTrigger = tsTrigger1+
        tim.diffTime(tsTrigger1,tsTrigger2)/2;
      fifoTailLa +=2;
    } else {
      // Trigger außerhalb des Fensters:
      // Triggerverlustzähler erhöhen
      nTriggerErr++;
    }
    break;
  case PhFirstLa:
    // keine ersten LA-Daten gekommen -> Fataler Fehler
    phase = PhFatal;
    gerr = Gerr_NoLaData;
    break;
  }
  // PIC von Interruptende informieren
  if (object->irq>=8) {
    outportb(ICR2,EOI);
  }
  outportb(ICR,EOI);
  enable();
}


//- Konstruktor Io

Io::Io() {
```

```
  init = 0;
  address = 0;
  object = this;
  irq = 0;
}

//- Destruktor Io
//.
//. Interrupt wieder deaktivieren

Io::~Io() {
  if (init != 0) {
    outportb(address+2,0x00);
    init = 0;
    char c,m;
    disable();
    if (irq>=8) {
      m = (1 << (irq-8));
      c = inportb(IMR2) | m;
      outportb(IMR2, c);
      setvect(0x68+irq,oldvect);
    } else {
      m = (1 << (irq));
      c = inportb(IMR) | m;
      outportb(IMR, c);
      setvect(0x08+irq,oldvect);
    };
    enable();
  };
}

//- Io::Initialize
//.
//. Initalisierung des Triggerinterrupts

int Io::Initialize(int Address,int Irq) {
  char c, m;

  if ((Address == 0) || (Irq<0) || (Irq>15))  return -1;

  init = 1;
  address = Address;
  irq     = Irq;

  disable();
  // Setzte Interruptvektor auf Routine inter
  if (irq>=8) {
    oldvect = getvect(0x68+irq);
    setvect(0x68+irq,inter);
    m = ~(1 << (irq-8));
    c = inportb(IMR2) & m;
    outportb(IMR2, c);
    c = inportb(IMR) & 0xFB;
    outportb(IMR, c);
  } else {
    oldvect = getvect(0x08+irq);
    setvect(0x08+irq,inter);
    m = ~(1 << (irq));
    c = inportb(IMR) & m;
    outportb(IMR, c);
  };
  outportb(ICR,EOI);
  // Datenregister löschen
  outportb(address,0x00);
  // Interrupt der ACK-Leitung aktivieren
  outportb(address+2,0x10);
  enable();
  return 0;
}
```

B.2.5 Speicherung und Übertragung der Pulsdaten an den Server-PC

Header SAVE.H

```
#ifndef __SAVE_H    /* prevent multiple includes */
#define __SAVE_H

#define LenSaveFifo  2000
#define LenErrFifo   1000
#define LenSepBuf    50

#define Timeout   2500000L

// --- Datenstruktur der Pulsdaten eines Kanals

struct OnePPS {
  unsigned int       flags;
   // Flags
  long int           offset;   // Ablage in ns
  unsigned long int  second;
  // Sekundenanteil als
                               // GPS-Sekundenwert
  int                delay1;
  // Verzögerung Nachricht 1
  int                delay2;
  // Verzögerung Nachricht 2
  long               free;
    // freier Datenwert
};


// --- Pulsdaten aller Kanäle für eine Sekunde

struct OnePPSBlock {
  unsigned valid;
             // Gültigkeitswert für
                               // Pulsdaten
  OnePPS onePPS[8];
           // Pulsdaten der einzelnen
                               // Kanäle
};


// --- Fehlpulsdatenstruktur

struct ErroneousPPS {
  int                savePort; // Kanal auf dem
                               // Fehlpuls auftrat
  unsigned long      ref;
     // Meßsekunde seit
                               // Meßbeginn als
                               // Fehlpuls auftrat
  unsigned           valid;
   // Gültigkeitswert wie
                               // für OnePPSBlock

  unsigned int       flags;
   // selbe Datenstruktur
  long int           offset;   // wie OnePPS
  unsigned long int  second;
  int                delay1;
  int                delay2;
  long               free;
};


//- Objekt Save
//.
//. Dient der Speicherung der Daten und der
//. sukzessiven Übermittlung an den Server-PC

class Save {
  unsigned long sepRef;
  int sepNum1,sepNum2,sepNum3;

  unsigned long errFifoHead;
```

```
  unsigned long errFifoTail;
  unsigned long errFifoShow;
  unsigned long errFifoSent;

  unsigned long saveFifoSent;
  unsigned long saveFifoHead;
  unsigned long saveFifoTail;
  unsigned long tsPuls;
  unsigned long tsFehl;
  ErroneousPPS sepBuf1[LenSepBuf];
  ErroneousPPS sepBuf2[LenSepBuf];
  ErroneousPPS sepBuf3[LenSepBuf];
  OnePPSBlock huge *saveFifo;
  ErroneousPPS huge *errFifo;
public:
  Save();
  ~Save();
  Synchron();
  int NewPulse(int Num,unsigned long ref, long offset,
               unsigned long second, int delay1,
               int delay2, unsigned flags, long free,
               unsigned long second2,long free2,
               long free3,long free4);
  int GeneralConditions(unsigned long ref,
                        int conditions);
  int Show(unsigned long ref);
  int Send(unsigned long ref);
  int Sendf(unsigned long number);
};



#endif
```

Modul SAVE.CPP

```
//: SAVE.CPP
//.
//. Modul zum Speichern der Pulsdaten und
//. zu deren Übermittelung an den Server-PC

#include <stdio.h>
#include <io.h>
#include <dos.h>
#include <dir.h>
#include <mem.h>
#include <string.h>
#include "global.h"
#include <conio.h>

#include "save.h"
#include "com.h"
#include "timer.H"
#include <alloc.h>

//- Konstruktor Save

Save::Save() {
  errFifoHead = errFifoTail = errFifoShow =
  errFifoSent = 0;
  saveFifoSent = saveFifoHead = saveFifoTail = 0;
  saveFifo = (OnePPSBlock huge *) farcalloc(LenSaveFifo,
             sizeof(OnePPSBlock));
  errFifo = (ErroneousPPS huge *) farcalloc(LenErrFifo,
             sizeof(ErroneousPPS));
  if (saveFifo == 0) {
    fprintf(stdout,"Zuwenig Speicher für saveFifo");
    getch();
  }
  if (errFifo == 0) {
    fprintf(stdout,"Zuwenig Speicher für errFifo");
    getch();
  }
  sepNum1 = 0;
  sepNum2 = 0;
  sepNum3 = 0;
  sepRef = 0;
  servLen = 0;
  newAck = 0;
  ack = 99;
  newAckErr = 0;
  ackErr = 99;
  tsPuls = tim.getTime();
  tsFehl = tim.getTime();
}

//- Destruktor Save

Save::~Save() {
}


//-Save::NewPulse

int Save::NewPulse(int Num,unsigned long ref, long offset,
    unsigned long second, int delay1, int delay2,
    unsigned flags,long free, unsigned long second2,
    long free2,long free3,long free4) {

  if (ref>sepRef+3) {
    int i;
    for (i=0; i<sepNum1; i++) {
      if (((errFifoTail+1) % LenErrFifo)
          == (errFifoHead% LenErrFifo)) {
        phase =PhFatal;
        gerr = Gerr_ErrFifoOverflow;
        return -1;
      } else {
        errFifo[errFifoTail % LenErrFifo] = sepBuf1[i];
        errFifoTail++;
      }
    }
    for (i=0; i<sepNum2; i++) {
      if (((errFifoTail+1) % LenErrFifo)
         == (errFifoHead% LenErrFifo)) {
        phase =PhFatal;
        gerr = Gerr_ErrFifoOverflow;
        return -1;
      } else {
        errFifo[errFifoTail % LenErrFifo] = sepBuf2[i];
        errFifoTail++;
      }
    }
    for (i=0; i<sepNum3; i++) {
      if (((errFifoTail+1) % LenErrFifo)
           == (errFifoHead% LenErrFifo)) {
        phase =PhFatal;
        gerr = Gerr_ErrFifoOverflow;
        return -1;
      } else {
        errFifo[errFifoTail % LenErrFifo] = sepBuf3[i];
        errFifoTail++;
      }
    }
    sepRef +=4;
    sepNum1 = 0;
    sepNum2 = 0;
    sepNum3 = 0;
  }

  // Fifo erhöhen falls notwendig
  if (ref>(saveFifoHead +LenSaveFifo-1)) {
    phase =PhFatal;
    gerr = Gerr_SaveFifoOverflow;
    return -1;
  }
  if (ref>=saveFifoTail+4) {
    saveFifoTail += 4;
  }

  OnePPSBlock huge *s = &saveFifo[ref % LenSaveFifo];
  OnePPS * p;  p = &(s->onePPS[Num]);
  OnePPS * pf;

  if (p->flags & 1)
  {
```

```
    ErroneousPPS huge * e =
      &errFifo[errFifoTail % LenErrFifo];
    switch(ref % 4) {
      case 0:
        if (((errFifoTail+1) % LenErrFifo)
           == (errFifoHead% LenErrFifo)) {
          phase =PhFatal;
          gerr = Gerr_ErrFifoOverflow;
          return -1;
        } else {
          errFifoTail++;
        }
        break;
      case 1:
        e = &sepBuf1[sepNum1++]; break;
      case 2:
        e = &sepBuf2[sepNum2++]; break;
      case 3:
        e = &sepBuf3[sepNum3++]; break;
    }
    // Fehlpuls
    e->savePort       = Num;
    e->ref            = ref;
    e->offset         = offset;
    e->second         = second;
    e->delay1         = delay1;
    e->delay2         = delay2;
    e->free           = free;
    e->flags          = flags | 1;
    e->valid          = s->valid;
  }  else {
    p->offset         = offset;
    p->flags          = flags | 1;
    if ((Num == 6) || (Num == 7))  {
    ;
    } else {
      p->second         = second;
      p->delay1         = delay1;
      p->delay2         = delay2;
      p->free           = free;
    }
    if (Num == 0) {
      pf = &(s->onePPS[6]);
      pf->second = second2;
      pf->free = free2;
      pf->delay1 = (int) (free3 & 0xFFFF);
      pf->delay2 = (int) ((free3>>16) & 0xFFFF);
      pf = &(s->onePPS[7]);
      pf->free = free4;
    }
  }

  return 0;
}

//- Save::GeneralConditions

int Save::GeneralConditions(unsigned long ref,
     int conditions) {
  if (ref>(saveFifoHead +LenSaveFifo-1)) {
    phase =PhFatal;
    gerr = Gerr_SaveFifoOverflow;
    return -1;
  }
  if (ref>=saveFifoTail+4) {
    saveFifoTail += 4;
  }
  unsigned fifoPoint = ref % LenSaveFifo;
  for (int j=0; j<8; j++) {
    saveFifo[fifoPoint].onePPS[j].flags = 0;
  }
  saveFifo[fifoPoint].valid = conditions;
  return 0;
}

//- Save::Show

int Save::Show(unsigned long ref) {
  int i;
```

```
unsigned flags;
char color;
// Block aus Fifo holen
OnePPSBlock huge * s= &saveFifo[ref % LenSaveFifo];
OnePPS *p,*ph,*pi;
flags = s->valid;
// Statuszeile für LA löschen

if (!(StatusFlag & StatusStop)) {
  out.Box(52,1,79,1,0x1f);
  sprintf(buf,"%10lu",ref);
  out.PosText(23,1,buf,0x16);
  out.PosText(27,12,buf,0x16);
  if (flags & SysLaDataProblem) {
    switch (flags>>4) {
      case 0:  out.PosText(52,1,"Keine LA -Daten",
                         0xcf); break;
      case 1:  out.PosText(52,1,"Kein Trigger in LA",
                         0xcf); break;
      case 2:  out.PosText(52,1,"Irregulärer Trigger ",
                         0xcf); break;
      case 3:  out.PosText(52,1,"Falscher Triggerlevel",
                         0xcf); break;
      case 4:  out.PosText(52,1,"Falsche Anzahl Ref100",
                         0xcf); break;
      case 5:  out.PosText(52,1,"Kein Triggerende",
                         0xcf); break;
      default: out.PosText(52,1,"Unbekannter Fehler",
                         0xcf); break;
    }
    // Leere Daten
    out.Box(11,3,79,11,0x1f);
  } else {
    if (flags & SysTriggerLost) {
      out.PosText(52,1,"Lost",0xcf);
    }
    if(flags & SysTriggerErr) {
      out.PosText(57,1,"Err",0xcf);
    }
    if (flags & SysTimerFail) {
      out.PosText(57,1,"Time",0xcf);
    }
    out.Box(11,3,79,11,0x1f);
    for (i = 0; i<8; i++) {
      p = &(s->onePPS[i]);
      unsigned flags = p->flags;
      if (flags & SaveFilled) {
        sprintf(buf,"%04X",flags);
        out.PosTextB(4,76,i+3,buf,0x1f);
        if (flags & SaveJumpyChannel)
          out.PosTextB(1,73,i+3,"J",0x1f);
        // Offset
        if (p->offset != InvalidLong) {
          sprintf(buf,"%11ld",p->offset);
          out.PosText(11,i+3,buf,0x1f);
        }
        if ((i != 6) && (i!=7)) {
               // Sekundenanteil
          if (p->second != InvalidLong) {
            sprintf(buf,"%10lu",p->second);
            out.PosText(23,i+3,buf,0x1f);
          }
          // Delay Message 1
          if (p->delay1 != InvalidWord) {
            color = (flags & SaveCorruptedMessage1)
                      ? 0x1C: 0x1f;
            sprintf(buf,"%5d",p->delay1);
            out.PosText(34,i+3,buf,color);
          }
          // Delay Message 2
          if (p->delay2 != InvalidWord) {
            color = (flags & SaveCorruptedMessage2)
                      ? 0x1C: 0x1f;
            sprintf(buf,"%5d",p->delay2);
            out.PosText(40,i+3,buf,color);
          }
        }
        if (flags & SaveDataValid)
          out.PosTextB(1,46,i+3,"V",0x1F);
```

```
        if (flags & SaveReceiverFault)
          out.PosTextB(1,47,i+3,"F",0x1C);
        if (flags & SaveAntennaFault)
          out.PosTextB(1,48,i+3,"A",0x1C);

        switch(i) {
          case 0:
            if (p->delay2 != InvalidWord) {
              ph = &(s->onePPS[6]);
              pi = &(s->onePPS[7]);
              sprintf(buf,"%10ld %10ld", p->free,
                (((long) ph->delay2)<<16)
                +((long) ph->delay1));
              out.PosText(50,i+3,buf,0x1f);
              sprintf(buf,"%10lu",ph->second);
              out.PosText(23,6+3,buf,0x1f);
              sprintf(buf,"%11ld %11ld ", ph->free,
                pi->free);
              out.PosText(50,6+3,buf,0x1f);
            }
            if (p->delay1 != InvalidWord) {
              sprintf(buf,"%2u", p->flags>>8);
              out.PosText(72,i+3,buf,0x1f);
            }
            break;
          case 1:
            if (p->delay2 != InvalidWord) {
              sprintf(buf,"%11ld ns",p->free);
              out.PosText(50,i+3,buf,0x1f);
            }
            break;
          case 2:
            if (p->delay1 != InvalidWord) {
              sprintf(buf,"%08lX",p->free);
              out.PosText(50,i+3,buf,0x1f);
            }
            break;
          case 3:
            if (p->delay1 != InvalidWord) {
              sprintf(buf,"%11ld",p->free);
              out.PosText(50,i+3,buf,0x1f);
            }
            break;
          case 4:
            if (p->delay1 != InvalidWord) {
              sprintf(buf,"%02X",p->flags>>8);
              out.PosText(50,i+3,buf,0x1f);
            }
            break;
          case 5:
            if (p->delay1 != InvalidWord) {
              sprintf(buf,"%11ld %11ld ",p->free,
                p->offset-p->free+1000000000L);
              out.PosText(50,i+3,buf,0x1f);
            }
            break;
        }
      }
    }
  }
}
// Fehlpulsliste darstellen
i = 0;
ErroneousPPS huge * e = &errFifo[errFifoShow %
  LenErrFifo];
if (!(StatusFlag & StatusStop))
  out.Box(1,14,79,19,0x1f);
while ((errFifoTail!=errFifoShow)
    && (e->ref <= ref))  {
  if (!(StatusFlag & StatusStop)) {
    if (i<6) {
      out.PosTextB(10,1,i+14,gpsName(e->savePort),
        0x1f);
      unsigned flags = e->flags;
      if (flags & SaveFilled) {
        sprintf(buf,"%04X",flags);
        out.PosTextB(4,76,i+14,buf,0x1f);
        if (flags & SaveJumpyChannel)
          out.PosTextB(1,73,i+14,"J",0x1f);
      // Offset
      if (e->offset != InvalidLong) {
        sprintf(buf,"%11ld",e->offset);
        out.PosText(11,i+14,buf,0x1f);
      }
      if ((e->savePort != 6) && (e->savePort!=7)) {
              // Sekundenanteil
        if (e->second != InvalidLong) {
          sprintf(buf,"%10lu",e->second);
          out.PosText(23,i+14,buf,0x1f);
        }
        // Delay Message 1
        if (e->delay1 != InvalidWord) {
          color = (flags & SaveCorruptedMessage1) ?
                   0x1C: 0x1f;
          sprintf(buf,"%5d",e->delay1);
          out.PosText(34,i+14,buf,color);
        }
        // Delay Message 2
        if (e->delay2 != InvalidWord) {
          color = (flags & SaveCorruptedMessage2) ?
                   0x1C: 0x1f;
          sprintf(buf,"%5d",e->delay2);
          out.PosText(40,i+14,buf,color);
        }
      }
      if (flags & SaveDataValid)
        out.PosTextB(1,46,i+3,"V",0x1F);
      if (flags & SaveReceiverFault)
        out.PosTextB(1,47,i+3,"F",0x1C);
      if (flags & SaveAntennaFault)
        out.PosTextB(1,48,i+3,"A",0x1C);

      switch(e->savePort) {
        case 0:
          if (e->delay2 != InvalidWord) {
            ph = &(s->onePPS[6]);
            pi = &(s->onePPS[7]);
            sprintf(buf,"%10ld", e->free);
            out.PosText(50,i+14,buf,0x1f);
          }
          if (e->delay1 != InvalidWord) {
            sprintf(buf,"%2u", e->flags>>8);
            out.PosText(72,i+14,buf,0x1f);
          }
          break;
        case 1:
          if (e->delay2 != InvalidWord) {
            sprintf(buf,"%11ld ns",e->free);
            out.PosText(50,i+14,buf,0x1f);
          }
          break;
        case 2:
          if (e->delay1 != InvalidWord) {
            sprintf(buf,"%08lX",e->free);
            out.PosText(50,i+114,buf,0x1f);
          }
          break;
        case 3:
          if (e->delay1 != InvalidWord) {
            sprintf(buf,"%11ld",e->free);
            out.PosText(50,i+14,buf,0x1f);
          }
          break;
        case 4:
          if (e->delay1 != InvalidWord) {
            sprintf(buf,"%02X",e->flags>>8);
            out.PosText(50,i+14,buf,0x1f);
          }
          break;
        case 5:
          if (e->delay1 != InvalidWord) {
            sprintf(buf,"%11ld %11ld ",e->free,
              e->offset-e->free+1000000000L);
            out.PosText(50,i+14,buf,0x1f);
          }
          break;
      }
    }
```

```
      }
    }
    errFifoShow++;
    e = &errFifo[errFifoShow % LenErrFifo];
    i++;
  }
  return 0;
}

//- Save::Send

int Save::Send(unsigned long ref) {
  char *data;
  int i;
  char check1,check2;
  if (servLen == 0) {
    data = (char *) &saveFifo[ref % LenSaveFifo];
    serv[0] = '$';
    serv[1] = 'P';
    serv[2] = 'U';
    serv[3] = 'L';
    serv[4] = 'S';
    long unsigned *lh = (long unsigned *) (serv+5);
    *lh = ref;

    memcpy((void *) &serv[9], (void *) data,
      sizeof(OnePPSBlock));
    servLen = sizeof(OnePPSBlock) + 5 +4 +2 ;
    check1 = 0; check2 = 0;
    for (i=1; i<=servLen-3; i++) {
      check1 ^= serv[i];
      check2 += serv[i];
    }
    serv[servLen-2] = check1;
    serv[servLen-1] = check2;
  } else {
    return -1;
  }
  return 0;
}

//- Save::Sendf

int Save::Sendf(unsigned long number) {
  char *data;
  char check1,check2;
  int i;
  if (servLen == 0) {
    data = (char *) &errFifo[number % LenErrFifo];
    serv[0] = '$';
    serv[1] = 'F';
    serv[2] = 'A';
    long unsigned *lh = (long unsigned *) (serv+3);
    *lh = number;
    memcpy((void *) &serv[7], (void *) data,
       sizeof(ErroneousPPS));
    servLen = sizeof(ErroneousPPS) + 3 +4 +2 ;
    check1 = 0; check2 = 0;
    for (i=1; i<=servLen-3; i++) {
      check1 ^= serv[i];
      check2 += serv[i];
    }
    serv[servLen-2] = check1;
    serv[servLen-1] = check2;
  } else {
    return -1;
  }
  return 0;
}

//- Save::Synchron

int Save::Synchron() {
  char check1,check2;
  char *data;
  char *dest;
  // immer zwischen Puls und Fehlpuls wechseln
  // Sofern Puls vorhanden
  unsigned long t = tim.getTime();
```

```
// Pulsliste

if (saveFifoTail>saveFifoHead) {
  if (newAck != 0) {
    // Neu Bestätigung liegt vor
    newAck = 0;
    if (ack > saveFifoHead) {
    // offenbar gingen Acks verloren
      if (ack<saveFifoTail) {
        saveFifoHead = ack+1;
        if (saveFifoSent<saveFifoHead)
          saveFifoSent = saveFifoHead;
        // Zeitstempel für Timeout der nächsten Ack
        tsPuls = t;
      }
    } else if (ack == saveFifoHead) {
      // Eine weitere Message wurde damit übernommen
      saveFifoHead++;
      if (saveFifoSent<saveFifoHead)
        saveFifoSent = saveFifoHead;
      // Zeitstempel für Timeout der nächsten Ack
      tsPuls = t;
    } else {
      tsPuls = t;
    }
  } else {
    // Acknowledge Timeout -> von vorne beginnen
    if (tim.diffTime(tsPuls,t) > Timeout) {
      saveFifoSent = saveFifoHead;
      tsPuls = t;
    }
  }
  if (saveFifoTail>saveFifoSent) {
    if (Send(saveFifoSent)==0) saveFifoSent++;
  }
} else {
  tsPuls = t;
}

// Fehlpulsliste
if (errFifoTail>errFifoHead) {
  if (newAckErr != 0) {
    // Neu Bestätigung liegt vor
    newAckErr = 0;
    if (ack > saveFifoHead) {
    // offenbar gingen Acks verloren
      if (ackErr<errFifoTail) {
        errFifoHead = ackErr+1;
        if (errFifoSent<errFifoHead)
          errFifoSent = errFifoHead;
        // Zeitstempel für Timeout der nächsten Ack
        tsFehl = t;
      }
    } else if (ackErr == errFifoHead) {
      // Eine weitere Message wurde damit übernommen
      errFifoHead++;
      if (errFifoSent<errFifoHead)
        errFifoSent = errFifoHead;
      // Zeitstempel für Timeout der nächsten Ack
      tsFehl = t;
    } else {
      // Zeitstempel für Timeout der nächsten Ack
      tsFehl = t;
    }
  } else {
    // Acknowledge Timeout -> von vorne beginnen
    if (tim.diffTime(tsFehl,t) > Timeout) {
      // Zeitstempel für Timeout der nächsten Ack
      tsFehl = t;
      errFifoSent = errFifoHead;
    }
  }
  if (errFifoTail>errFifoSent) {
    if (Sendf(errFifoSent)==0) errFifoSent++;
  }
} else {
  tsFehl = t;
}
sprintf(buf,"%10lu",saveFifoTail);
```

```
  out.PosText(30,20,buf,0x17);
  sprintf(buf,"%10lu",saveFifoSent);
  out.PosText(50,20,buf,0x17);
  sprintf(buf,"%10lu",saveFifoHead);
  out.PosText(70,20,buf,0x17);

  sprintf(buf,"%10lu",errFifoTail);
  out.PosText(30,21,buf,0x17);
  sprintf(buf,"%10lu",errFifoSent);
  out.PosText(50,21,buf,0x17);
  sprintf(buf,"%10lu",errFifoHead);
  out.PosText(70,21,buf,0x17);

  return 0;
}
```

B.3 Low-Level-Treiber

B.3.1 Bedienung des Timers

Header TIMER.H

Modul TIMER.CPP

```
//: TIMER.CPP
//.
//. Modul zum Auslesen des PC-Timers
//. Es wird eine Klasse Tim definiert, über die
//. auf den Timer zugegriffen werden kann.
//. Für einen einfachen Zugriff wurde gleich eine
//. Instanz tim der Klasse Tim erzeugt
//.
//. 09.11.1995 Bug behoben: BIOS-Zähler bricht alle 24h
//.            den Timerzählwert um

#include "timer.h"

// Adresse des BIOS-Zählers (wird alle 65536 PC-Ticks
// um 1 erhöht

const unsigned long far * const dosTime =
    (const unsigned long far * const)MK_FP( 0x40, 0x6C );

//- Eine Instanz wird gleich definiert:

class Tim tim;

//- Konstruktor Tim
//.
//. Der Timer wird initialisiert.

Tim::Tim() {
  corr = 0;
  disable();
  last = *dosTime;
  outportb(0x43,0x34);
  outportb(0x40,0x00);
  outportb(0x40,0x00);
  enable();
}

//- Destruktor Tim
//.
//. Timer wieder zurücksetzen

Tim::~Tim() {
  disable();
  outportb(0x43,0x34);
  outportb(0x40,0x00);
  outportb(0x40,0x00);
  enable();
}

//- Convenience-Funktion difftime
//. zur Berechnung der Zeitdifferenz
//. zwischen zwei Zeitstempeln t1, t2
//. Liegen die Zeitstempel nicht zu weit auseinander
//. (<30  min), so ist das Vorzeichen der erhaltenen
//. Differenz korrekt

long Tim::diffTime(unsigned long t1, unsigned long t2) {
  return (long) t2- (long) t1;
}

//- Convenience-Funktion transTime
//. Ein Zeitstempel wird in Einheiten von Sekunden
//. umgerechnet

double Tim::transTime(unsigned long t) {
  return t*PCTick;
}

//- getTimed()
//. Ermittelt den momentanen Zeitstempel unter
//. der Vorraussetzung, daß alle Interrupts gesperrt
//. sind. Der Zeitstempel wird als unsigned long-Wert
//. erhalten, der aufgrund des Speicherumfangs eines
//. solchen Werts nur innerhalb einer Stunde eindeutig ist,
//. dann wiederholen sich die Werte wieder.


unsigned long Tim::getTimed() {
  unsigned long sample1;
  unsigned count;

  outportb(0x43,0xD2);
  unsigned char f1,f2;
  f1 = inportb(0x40);
  f2 = inportb(0x40);
  count = ~(f1+256*f2);
  sample1 = *dosTime;
  // Interrupt 0 hat sich gemeldet ?
  outportb(0x20,0xA);
  if (inportb(0x20) & 1) {
    sample1 = *dosTime+1;
    outportb(0x43,0xD2);
    f1 = inportb(0x40);
    f2 = inportb(0x40);
    count = ~(f1+256*f2);
  }
  if (sample1<last) {
last = sample1;
corr += 0x001800B0;
  }
  sample1 += corr;
  return (sample1<<16)+count;
}

//- getTime()
//. Ermittelt den momentanen Zeitstempel unter
//. der Vorraussetzung, daß die Interrupts nicht gesperrt
//. sind. Der Zeitstempel wird als unsigned long-Wert
//. erhalten, der aufgrund des Speicherumfangs eines
//. solchen Werts nur innerhalb einer Stunde eindeutig ist,
//. dann wiederholen sich die Werte wieder.

unsigned long Tim::getTime() {
  unsigned long sample1;
  unsigned count;

  disable();

  outportb(0x43,0xD2);
  unsigned char f1,f2;
  f1 = inportb(0x40);
  f2 = inportb(0x40);
  count = ~(f1+256*f2);
  sample1 = *dosTime;
```

```
  // Interrupt 0 hat sich gemeldet ?
  outportb(0x20,0xA);
  if (inportb(0x20) & 1) {
    sample1 = *dosTime+1;
    outportb(0x43,0xD2);
    f1 = inportb(0x40);
    f2 = inportb(0x40);
    count = ~(f1+256*f2);
  }
  enable();
  // Lästige Bios-Korrektur
  // Im Bios folgt auf den Wert 0x001800B0 der
  // Wert 0 (enstpricht Zeitraum von 24 h) ->
  // Diesen Umbruch rückgängig machen
  if (sample1<last) {
last = sample1;
corr += 0x001800B0;
  }
  sample1 += corr;
  return (sample1<<16)+count;
}
```

B.3.2 Bedienung des Bildschirms

Header OUT.H

```
#ifndef __OUT_H /* prevent multiple includes */
#define __OUT_H

class Out {
char *address;
public:
  Out();
  ~Out();
  void Box(int x1,int y1,int x2,int y2,char typ);
  void PosText(int x, int y, const char * string,char typ);
  void PosTextB(int len,int x, int y, const char * string,char typ);
  void Clear();
  void CenterText(int y, const char* string, char typ);
};

#endif
```

Modul OUT.CPP

```
//: OUT.CPP
//.
//. Modul zum Ansprechen des Bildschirms
//. Koordinaten beginnen mit x=1, y=1 in der linken
//. oberen Ecke

#include <dos.h>
#include <string.h>
#include "out.h"

// Instanz des Objekts gleich definieren

Out out;

//- Konstruktor Out
//.
//. Die Klasse wird initialisiert.

Out::Out() {
  address = (char *) MK_FP(0xb800,0x0000);
}

//- Destruktor Out
//.

Out::~Out() {
}

//- Box(int,int,int,int,char);
//. Ausgabe einer Box mit Koordinaten x1,y1,x2,y2
//. und Farbe typ

void Out::Box(int x1,int y1,int x2,int y2,char typ) {
  int a1,a2,b1,b2;
  int i,j;
  if (x1<x2) {
    a1 = x1; a2 = x2;
  } else {
    a1 = x1; a2 = x2;
  }
  if (y1<y2) {
    b1 = y1; b2 = y2;
  } else {
    b1 = y1; b2 = y2;
  }

  if (a2<0) return;
  if (b2<0) return;
  if (a1>79) return;
  if (b1>24) return;
  if (a1<0) a1 = 0;
  if (b1<0) b1 = 0;
  if (a2>79) a2 = 79;
  if (b2>24) b2 = 24;

  int off = b1*160+a1*2;
  int sx = 160-2-2*(a2-a1);

  for (j=b1; j<=b2; j++) {
    for (i=a1; i<=a2; i++) {
      address[off] =' ';
      address[off+1] = typ;
      off += 2;
    }
    off += sx;
  }
  return;
}

//- Clear();
//. Löschen des Bildschirms mit hellblauem Hintergrund

void Out::Clear() {
  int i;
  for (i=0; i<25*80*2; i++) {
    address[2*i]= ' ';
    address[2*i+1] = 0x07;
  }
}

//- PosText(int,int,char*,char)
//. Ausgabe von Text beginnend mit der Position x,y und
//. dem String string mit der Farbe typ
//. Clipping für Normalbildschirm (80x25) ist
//. implementiert

void Out::PosText(int x, int y, const char * string,
  char typ) {
  const char *ad;
  ad = string;
  if ((x>79) || (y<0) || (y>24)) return;
  if (x<0) {
    if (x+strlen(string)>0){
      ad += -x;
      x = 0;
    }
  }
  int off = 160*y+2*x;
  while ((*ad != 0) && (x<=79)) {
    address[off] = *ad;  ad++;
    address[off+1] = typ;
    off +=2;
    x++;
  }
}

//- PosTextB(int,int,char*,char)
```

```
//. Ausgabe von Text beginnend mit der Position x,y und
//. der Länge len aus dem Char-Array string mit der Farbe
//. typ
//. Clipping für Normalbildschirm (80x25) ist
//. implementiert

void Out::PosTextB(int len,int x, int y,
   const char * string, char typ) {
  int i;
  if ((x<0) || (x>79) || (y<0) || (y>24)) return;
  int off = 160*y+2*x;
  for (i=0; i<len; i++) {
    if (*string) {
      address[off] = *string;
      address[off+1] = typ;
      string++;
    } else {
      address[off] = ' ';
      address[off+1] = typ;
    }
    off+=2;
    x++;
    if (x>79) break;
  }
}


//- CenterText(int,char*,char)
//. Ausgabe von zentriertem Text in der Zeile y und
//. dem String string mit der Farbe typ
//. Clipping für Normalbildschirm (80x25) ist
//. implementiert


void Out::CenterText(int y, const char * string,
     char typ) {
  int len = strlen(string);
  PosText(40-len/2,y,string,typ);
}


//- char hex(int)
//. Umwandlung einer Zahl in eine ASCII-Hexziffer

char hex(int zahl) {
  if (zahl>=10) return 'A'+zahl-10;
  return zahl+'0';
}
```

B.3.3 Bedienung der 8-fach seriellen Schnittstelle

Header SERIALBO.H

```
#ifndef __SERIALBO_H    /* prevent multiple includes */
#define __SERIALBO_H

// Buffergröße für einlangende Zeichen 512
#define ReadBufSize 0x200
// Buffergröße für ausgehende Zeichen 2048
#define WriteBufSize 0x800
// Anzahl der Port auf der seriellen Schnittstellenkarte
#define NumPorts 8


/*
    The 8250 UART has 10 registers accessible through
    7 port addresses. Here are their addresses relative
    to the baseadresses.  Note  that the baud rate
    registers, (DLL) and (DLH) are active only when
    the Divisor-Latch Access-Bit (DLAB) is on. The (DLAB)
    is bit 7 of the (LCR).

o TXR Output data to the serial port.
o RXR Input data from the serial port.
o LCR Initialize the serial port.
o IER Controls interrupt generation.
o IIR Identifies interrupts.
o MCR Send contorl signals to the modem.
o LSR Monitor the status of the serial port.
o MSR Receive status of the modem.
o DLL Low byte of baud rate divisor.
o DHH High byte of baud rate divisor.
*/
#define TXR    0       /*  Transmit register (WRITE) */
#define RXR    0       /*  Receive register  (READ)  */
#define IER    1       /*  Interrupt Enable          */
#define IIR    2       /*  Interrupt ID              */
#define LCR    3       /*  Line control              */
#define MCR    4       /*  Modem control             */
#define LSR    5       /*  Line Status               */
#define MSR    6       /*  Modem Status              */
#define DLL    0       /*  Divisor Latch Low         */
#define DLH    1       /*  Divisor latch High        */

// Eigenheit der BOCA-Karte:
// Über das Byte (baseaddress+7) steht ein
// Art Interrupt Pending Register zur Verfügung.
// Die einzelnen Bits geben Auskunft darüber, ob
// der entsprechende Port (Bit0 = port0,...)
// einen Interrupt wünscht
#define IPR    7

/*---------------------------------------------------*
  Bit values held in the Line Control Register (LCR).
bit meaning
----------
0-1 00=5 bits, 01=6 bits,
                        10=7 bits, 11=8 bits.
2 Stop bits.
3 0=parity off, 1=parity on.
4 0=parity odd, 1=parity even.
5 Sticky parity.
6 Set break.
7 Toggle port addresses.
 *---------------------------------------------------*/

#define NO_PARITY       0x00
#define EVEN_PARITY     0x18
#define ODD_PARITY      0x08


/*---------------------------------------------------*
   Bit values held in the Line Status Register (LSR).
bit meaning
----------
0 Data ready.
1 Overrun error - Data register
                        overwritten.
2 Parity error - bad transmission.
3 Framing error - No stop bit was
                        found.
4 Break detect - End to transmission
                        requested.
5 Transmitter holding register is
                        empty.
6 Transmitter shift register is
                        empty.
7               Time out - off line.
 *---------------------------------------------------*/

#define RCVRDY          0x01
#define OVRERR          0x02
#define PRTYERR         0x04
#define FRMERR          0x08
#define BRKERR          0x10
#define XMTRDY          0x20
#define XMTRSR          0x40
#define TIMEOUT 0x80

/*---------------------------------------------------*
  Bit values held in the Modem Output Control Register
      (MCR).
```

```
bit      meaning
----------
0 Data Terminal Ready. Computer
                        ready to go.
1 Request To Send. Computer wants
                        to send data.
2 auxillary output #1.
3 auxillary output #2.(Note: This
                        bit must be set to allow the
                        communications card to send
interrupts to the system)
4 UART ouput looped back as input.
5-7 not used.
 *------------------------------------------------------*/

#define DTR             0x01
#define RTS             0x02
#define MC_INT 0x08

/*------------------------------------------------------*
  Bit values held in the Modem Input Status Register
    (MSR).
bit meaning
----------
0 delta Clear To Send.
1 delta Data Set Ready.
2 delta Ring Indicator.
3 delta Data Carrier Detect.
4 Clear To Send.
5 Data Set Ready.
6 Ring Indicator.
7 Data Carrier Detect.
 *------------------------------------------------------*/

#define CTS             0x10
#define DSR             0x20

/*------------------------------------------------------*
  Bit values held in the Interrupt Enable Register (IER).
bit meaning
----------
0 Interrupt when data received.
1 Interrupt when transmitter
                        holding reg. empty.
2 Interrupt when data
                        reception error.
3 Interrupt when change in modem
                        status register.
4-7 Not used.
 *------------------------------------------------------*/

#define RX_INT          0x01

/*------------------------------------------------------*
  Bit values held in the Interrupt Identification
     Register (IIR).
bit meaning
----------
0 Interrupt pending
1-2             Interrupt ID code
00=Change in modem status
                           register,
01=Transmitter holding
                           register empty,
10=Data received,
11=reception error, or break
                           encountered.
3-7 Not used.
 *----------------------------------------------------*/

#define RX_ID           0x04
#define RX_MASK         0x07

/*
    These are the port addresses of the 8259
    Programmable Interrupt Controller (PIC).
*/
/* Interrupt Mask Register port */
#define IMR             0x21
/* Interrupt Control Port      */
#define ICR             0x20

#define IMR2        0xA1
#define ICR2 0xA0

/*
    An end of interrupt needs to be sent to the Control
    Port of the 8259 when a hardware interrupt ends.
*/
#define EOI             0x20   /* End Of Interrupt */


/*
    The (IMR) tells the (PIC) to service an interrupt
    only if it is not masked (FALSE).
*/

#define BUFOVFL         -1

class boca;
class comport;
extern void interrupt com(...);
extern void interrupt com2(...);
extern int SError;

class port {
  char readBuf[ReadBufSize];
  unsigned long TS[ReadBufSize];
  char writeBuf[WriteBufSize];
  int portAddress;
  int Init(int PortAddress);
  int Num;
public:
  unsigned readHead,readTail,writeHead,writeTail;
  port(): portAddress(0),readHead(0),readTail(0),
          writeHead(0),writeTail(0),Num(-1) {};
  ~port();
  void Edge();
  int Configure(int Speed, int Parity, int Bits,
                 int StopBits);
  int Transmit(char c);
  int Read(char &c);
  int TransInterrupt(int on);
  int GetMSR(char &c);
  int GetLSR(char &c);
  int GetIIR(char &c);
  int operator<<( char ch );      // Write
  int operator<<( char *str );
  int operator>>( char &ch );   // Read
  int PutToReadBuf(char c, unsigned long t);
  int PutToWriteBuf(char c);
  int PutToWriteBuf(char *string);
  int PutToWriteBuf(char *string,int len);
  int GetFromReadBuf(char &c, unsigned long & t);
  int GetFromWriteBuf(char &c);
  int GetTopOfWriteBuf(char &c);
  friend class boca;
  friend class comport;
};

//- Treiberobjekt für die üblichen Schnittstellen
//  COM1 und COM2

class comport {
  int cardAddress;
  int irq;
  void interrupt (*oldvect) (...);
public:
  port p;
  comport(int CardAddress, int Irq);
  ~comport();
  port *getPort(int Speed, int Parity, int Bits,
    int StopBits);
  friend class port;
  friend void interrupt com2(...);
};
```

```
//- Treiberobjekt für die BOCA 8fach serielle
//  Schnittstelle

class boca {
  int init;
  int cardAddress;
  int irq;
  port p[NumPorts];
  void interrupt (*oldvect) (...);
public:
  boca();
  ~boca();
  int Initialize(int CardAddress,int Irq);
  port *getPort(int Port, int Speed, int Parity,
    int Bits, int StopBits);
  friend class port;
  friend void interrupt com(...);
};

#endif
```

Modul SERIALBO.CPP

```
//: SERIALBO.CPP
//.
//. Modul zur Bedienung der 8fach seriellen
//. Schnittstellenkarte
//. BB1008 und der normalen seriellen Schnittstelle COM1
//.

#include <dos.h>
#include <string.h>
#include <stdio.h>
#include "serialbo.h"
#include "timer.h"

static boca *object;
static comport *object2;

static SError = 0;

//. Interruptroutine für die BOCA-Karte

void interrupt com(...) {
  disable();
  // Interrupt Pending Register
  int cardAddress = object->cardAddress;
  char ipr = inportb(cardAddress+IPR);

  unsigned char m;
  int i;

  m = 0;
  char w;

  for(i=0; i<8; i++) {
    if (((w=inportb(cardAddress+MCR+i*8)) & 0x08) != 0) {
      m |= (1 << i);
      outportb(cardAddress+MCR+i*8,w & 0xF7);
    };
  };

  char c;
  char lsr;
  for (i=0; i<8; i++) {
    port &P = (object->p[i]);
    if ((ipr & (1<<i) ) !=0) {
      char iir;
      P.GetIIR(iir);
      switch(iir & 6) {
case 0:
  char msr;
  P.GetMSR(msr);
  break;
case 2:
  if (P.GetTopOfWriteBuf(c) == 0) {
    if (P.Transmit(c) == 0) {
      P.GetFromWriteBuf(c);
    }
  } else {
    // empty WriteBuffer
    P.TransInterrupt(0);
  }
  break;
case 4:
  if (P.Read(c) == 0) {
    unsigned long t= tim.getTimed();
    if (P.PutToReadBuf(c,t) == 0) {
    }
  }
  break;
case 6:
  P.GetLSR(lsr);
  break;
      }
    }
  }
  // Signal end of interrupt

  if ((object->irq)>=8) {
    outportb(ICR2, EOI);
  }
  outportb(ICR, EOI);
  for(i=0; i<8; i++) {
    if (m & (1 << i)) {
      outportb(cardAddress+i*8+MCR,
               inportb(cardAddress+i*8+MCR) | 0x08);
    }
  }
  enable();
}

//- Interruptroutine für normale serielle Schnittstellen

void interrupt com2(...) {
   disable();
  char w,m;
  int i;
  // Interrupt Pending Register
  int cardAddress = object2->cardAddress;

  char c;
  char lsr;
  char iir;
  port &P2 = (object2->p);
  P2.GetIIR(iir);
  switch(iir & 6) {
  case 0:
    char msr;
    P2.GetMSR(msr);
    break;
  case 2:
    if (P2.GetTopOfWriteBuf(c) == 0) {
      if (P2.Transmit(c) == 0) {
P2.GetFromWriteBuf(c);
      }
    } else {
      P2.TransInterrupt(0);
    }
    break;
  case 4:
    if (P2.Read(c) == 0) {
      unsigned long t;
      if (P2.PutToReadBuf(c,t) == 0) {
      }
    }
    break;
  case 6:
    P2.GetLSR(lsr);
    break;
  }
  if ((object2->irq)>=8) {
    outportb(ICR2, EOI);
  }
  outportb(ICR, EOI);
  char ier = inportb(cardAddress+IER);
```

```
  outportb(cardAddress+IER,0);
  outportb(cardAddress+IER,ier);
  enable();
}

comport::comport(int CardAddress, int Irq) {
  char c,m;
  cardAddress = CardAddress;
  irq = Irq;
  object2 = this;

  p.Init(cardAddress);

  disable();
  if (irq>=8) {
    oldvect = getvect(0x68+irq);
    setvect(0x68+irq,com2);
    m = ~(1 << (irq-8));
    c = inportb(IMR2) & m;
    outportb(IMR2, c);
    c = inportb(IMR) & 0xFB;
    outportb(IMR, c);
  } else {
    oldvect = getvect(0x08+irq);
    setvect(0x08+irq,com2);
    m = ~(1 << (irq));
    c = inportb(IMR) & m;
    outportb(IMR, c);
  };
  enable();
}

comport::~comport() {
  char m,c;
  disable();
  if (irq>=8) {
    m = (1 << (irq-8));
    c = inportb(IMR2) | m;
    outportb(IMR2, c);
    setvect(0x68+irq,oldvect);
  } else {
    m = (1 << (irq));
    c = inportb(IMR) | m;
    outportb(IMR, c);
    setvect(0x08+irq,oldvect);
  };
  enable();
  cardAddress = 0;
  irq = 0;
}

port * comport::getPort(int Speed, int Parity,
                        int Bits, int StopBits) {
  if (p.Configure(Speed, Parity, Bits, StopBits)==-1)
    return 0;

  return &p;
}

//- Konstruktor boca

boca::boca(){
  init = 0;
  cardAddress = 0;
  irq = 0;
  object = 0;
}

// Destruktor boca
//.
//. Interrupt der Boca-Karte sperren und
//. alten Interruptvektor zurückschreiben

boca::~boca() {
  if (init) {
    char m,c;
    disable();
    if (irq>=8) {
      m = (1 << (irq-8));
      c = inportb(IMR2) | m;
      outportb(IMR2, c);
    } else {
      m = (1 << (irq));
      c = inportb(IMR) | m;
      outportb(IMR, c);
    };
    enable();
    disable();
    if (irq>=8) {
      setvect(0x68+irq,oldvect);
    } else {
      setvect(0x08+irq,oldvect);
    };
    enable();
    cardAddress = 0;
    irq = 0;
  }
}



//- boca::Initialize
//.
//. Initialisiere Boca-Karte

int boca::Initialize(int CardAddress, int Irq) {
  if ((CardAddress == 0) || (irq<0) || (irq>15))
    return -1;

  char c,m;
  cardAddress = CardAddress;
  irq = Irq;
  object = this;

  for (int i=0; i<8; i++) {
    p[i].Init(cardAddress+i*8);
  };

  disable();
  if (irq>=8) {
    oldvect = getvect(0x68+irq);
    setvect(0x68+irq,com);
    m = ~(1 << (irq-8));
    c = inportb(IMR2) & m;
    outportb(IMR2, c);
    c = inportb(IMR) & 0xFB;
    outportb(IMR, c);
  } else {
    oldvect = getvect(0x08+irq);
    setvect(0x08+irq,com);
    m = ~(1 << (irq));
    c = inportb(IMR) & m;
    outportb(IMR, c);
  };
  enable();
  return 0;
}

//- boca::getPort
//.
//. Erzeuge ein port-Objekt, über das
//. auf den Port mit den angegebenen Schnittstellendaten
//. zugegriffen werden kann.

port * boca::getPort(int Port, int Speed, int Parity,
      int Bits, int StopBits) {
  if ((Port<1) || (Port>8)) return 0;
  if (p[Port-1].Configure(Speed, Parity, Bits,
                          StopBits)==-1)
    return 0;

  return &p[Port-1];
}
```

```
int port::Init(int PortAddress) {
  readTail = 0;
  readHead = 0;
  writeTail = 0;
  writeHead = 0;
  Num = 0;
  portAddress = PortAddress;
  return 0;
}

int port::Configure(int Speed, int Parity,
                    int Bits, int StopBits) {
  char c;
  if (Num == -1) return -1;
  if (Bits < 5 || Bits > 8) return (-1);
  if (StopBits != 1 && StopBits != 2) return (-1);
  if (Parity != NO_PARITY && Parity != ODD_PARITY &&
       Parity != EVEN_PARITY)
    return (-1);
  if (Speed == 0) return(-1);

  int divisor = (int) (115200L/Speed);
  int setting  = Bits-5;
  setting |= ((StopBits == 1) ? 0x00 : 0x04);
  setting |= Parity;

  outportb(portAddress+LCR,0);
  outportb(portAddress+IER,0);

    outportb(portAddress+IER,3);


  char iir;
  do {
    inportb(portAddress+RXR);
    inportb(portAddress+LSR);
    inportb(portAddress+MSR);
    iir = inportb(portAddress+IIR);
  } while ((iir & 1) != 1);


  disable();
  c = inportb(portAddress + LCR);
  outportb(portAddress + LCR, (c | 0x80));
  outportb(portAddress + DLL, (divisor & 0x00FF));
  outportb(portAddress + DLH, ((divisor >> 8) & 0x00FF));
  outportb(portAddress + LCR, c);
  outportb(portAddress + LCR,setting);

  c = inportb(portAddress + MCR) | MC_INT | DTR | RTS ;
  outportb(portAddress + MCR, c);
  enable();

  return 0;
}


void port::~port() {
  if (Num != -1) {
    outportb(portAddress + MCR, 0);
    outportb(portAddress + IER, 0);
    outportb(portAddress + LCR, 0);
  };
};

int port::Read(char & c) {
  c = inportb(portAddress + RXR);
  return 0;
}

int port::GetMSR(char &c) {
  c = inportb(portAddress+MSR);
  return 0;
}

int port::GetLSR(char &c) {
  c = inportb(portAddress+LSR);
  return 0;
}

int port::GetIIR(char &c) {
  c = inportb(portAddress+IIR);
  return 0;
}

void port::Edge() {
  if (Num != -1) {
    char c = inportb(portAddress+IER);
    outportb(portAddress+IER,0);
    outportb(portAddress+IER,c);
  }
}

int port::TransInterrupt(int on) {
  if (on) {
    outportb(portAddress+IER,3);
  } else {
    outportb(portAddress+IER,1);
  }
  return 0;
}


int port::Transmit(char c) {
      outportb(portAddress+TXR, c);
      return 0;
}

int port::operator<<(char c) {
  if (writeTail == writeHead) {
    if (Transmit(c) == 0) {
      return 0;
    }
  }
  return PutToWriteBuf(c);
}

// Output a string to the serial port

int port::operator<<( char *string ) {
  int n = 1;

  while (*string) {
    if (((*this) << *string) != 0) return n ;
    string++;
    n++;
  }
  return 0;
}

// This routine returns the current value in the buffer
int port::operator>>( char &c )
{
  unsigned long t;
  return GetFromReadBuf(c,t);
}




int port::GetFromWriteBuf(char &c) {
  if (writeHead == writeTail) {
    return -1;
  }
  c = writeBuf[writeHead++];
  writeHead %= WriteBufSize;
  return 0;
}


int port::GetFromReadBuf(char &c,unsigned long & t) {
  if (readHead == readTail) {
    return -1;
  }
  t = TS[readHead];
  c = readBuf[readHead++];
```

```
  readHead  %= ReadBufSize;
  return 0;
}

int port::PutToWriteBuf(char *string) {
  disable();
  int i;
  int n = strlen(string);
  if (n>=WriteBufSize) return -1;
  if (writeTail < writeHead) {
    if (writeTail+n>=writeHead) {
      return -1;
    }
  } else {
    if ((writeTail+n) >=WriteBufSize) {
      if (((writeTail+n)-WriteBufSize)>=writeHead) {
return -1;
      }
    }
  }
  if (writeHead == writeTail) {
    if ((inportb(portAddress+LSR) & XMTRDY) != 0) {
      outportb(portAddress+TXR, *string);
      string++;
    }
  }

  while (*string) {
    writeBuf[writeTail++] = *string;
    writeTail %= WriteBufSize;
    string++;

  }
  enable();
  return 0;
}

int port::PutToWriteBuf(char *string,int len) {
  int i;
  disable();
  if (len>=WriteBufSize) return -1;
  if (writeTail < writeHead) {
    if (writeTail+len>=writeHead) {
      return -1;
    }
  } else {
    if ((writeTail+len) >=WriteBufSize) {
      if (((writeTail+len)-WriteBufSize)>=writeHead) {
return -1;
      }
    }
  }
  TransInterrupt(1);
  writeBuf[writeTail++] = *string;
  writeTail %= WriteBufSize;
  string++;
  len--;
  enable();

  for (i=0; i<len; i++) {
    writeBuf[writeTail++] = *string;
    writeTail %= WriteBufSize;
    string++;
  }
  return 0;
}



int port::PutToWriteBuf(char c) {
  if (((writeTail+1) % WriteBufSize) == writeHead) {
    return -1;
  }
  writeBuf[writeTail++] = c;
  writeTail %= WriteBufSize;
  return 0;
}

int port::PutToReadBuf(char c,unsigned long t) {
  if (((readTail+1) % ReadBufSize) == readHead) {
    return -1;
  }
  TS[readTail] = t;
  readBuf[readTail++] = c;
  readTail %= ReadBufSize;
  return 0;
}

int port::GetTopOfWriteBuf(char &c) {
 if (writeHead != writeTail) {
   c = writeBuf[writeHead];
   return 0;
 }
 return -1;
}
```

B.3.4 Bedienung der Logikanalysatoren

Das Modul `hp16500` wurde von Robert Klima im Rahmen eines Praktikums am Institut für Computertechnik geschrieben. Die Dateien `HP16500.H` und `HP16500.CPP` sind im Anhang Abschnitt C.1 in der Softwaredokumentation des Moduls abgedruckt.

C Dokumente

C.1 Softwaredokumentation des Moduls zur Bedienung der Logikanalysatoren

Die folgenden 30 Seiten geben die Dokumentation zum Modul `hp16500.cpp` wieder. Dieses Modul enthält Routinen zur Bedienung der Logikanalysatoren. Modul und Dokumentation wurden vom Praktikanten Robert Klima am Institut für Computertechnik der TU Wien geschrieben.

C.2 Pläne zur Frequenzteilerschaltung

Die folgenden 5 Seiten enthalten die von Fritz Mörth gezeichneten Schaltpläne der Frequenzteilerschaltung.

C.3 Circular T97 des BIPM

Das BIPM (Bureau International des Poids et Mesures) gibt monatlich ein Dokument heraus, das die Ablagen der lokalen Zeitskalen von TAI (oder gleichbedeutend UTC) der über die ganze Welt verstreuten Zeitinstitute, die an der Bildung der Zeitskalen TAI und UTC beteiligt sind, auflistet. Der Name dieses Dokuments ist Circular T. Es wird jeweils zwei Monate nach dem ersten enthaltenen Meßwert herausgegeben und enthält Daten für ein ganzes Monat. Diese Verzögerung ist aufgrund der bei der Erzeugung von TAI (bzw. UTC) verwendeten langfristigen Mittelungsverfahren unvermeidlich.

Dieses Dokument ist übrigens über anonymous ftp unter der Internetadresse: `tycho.usno.navy.mil` zugänglich. Zur Illustration sei hier das circular T97 vom 16. Februar abgedruckt.

```
                                                    ISSN 1143-1393


                                       Circular T 97 (1996 February 16)

1 - Coordinated Universal Time UTC. Computed values of UTC-UTC(k).

(From 1994 July 1, 0hUTC, to 1996 January 1, 0hUTC, TAI-UTC = 29 s)
(From 1996 January 1, 0hUTC, until further notice,  TAI-UTC = 30 s)

Date 1995/96  0h UTC          Dec 28     Jan  2     Jan  7     Jan 12
      MJD                      50079      50084      50089      50094
Laboratory k                     UTC-UTC(k)  (Unit is one nanosecond)

AOS  (Borowiec)      (1)        -172       -450       -291       -225
APL  (Laurel)                   1967       1992          -       2025
AUS  (Canberra)                 -401       -413       -442       -473
BEV  (Wien)                     9885       9683       9506       9332
BIRM (Beijing)                   -14        -93       -157       -225
CAO  (Cagliari)                    -          -          -          -
CH   (Bern)                      177        204        211        225
CRL  (Tokyo)                     311        302        292        275
CSAO (Lintong)                  -292       -256       -197       -221
CSIR (Pretoria)                 4855       4947       5051       5167
FTZ  (Darmstadt)                -256       -256          -          -
GUM  (Warszawa)                 -343       -334       -336       -335

IEN  (Torino)                      8         16         24         45
IFAG (Wettzell)                -4668      -4722      -4768      -4771
IGMA (Buenos Aires)  (2)         373         77         77         72
INPL (Jerusalem)               -2823      -2867      -2890      -2913
IPQ  (Monte de Caparica)       -8116      -8171      -8213      -8270
JATC (Lintong)                  1442       1553       1685       1620
KRIS (Taejon)                    284        299        306        307
LDS  (Leeds)                     206        212        202        198
MSL  (Lower Hutt)              -5326      -5342      -5316      -5297
NAOM (Mizusawa)                -3135      -3135      -3131      -3141
NAOT (Tokyo)                   -3168      -3137      -3072      -3041
NIM  (Beijing)                  8006       8011       8020       8021

NIST (Boulder)                    13         14         13         13
NPL  (Teddington)                 39         40         46         50
NPLI (New-Delhi)                   -          -          -          -
NRC  (Ottawa)                     44         59         67         75
NRLM (Tsukuba)                 -5342      -5278      -5201      -5133
OMH  (Budapest)                13262      13502      13740      13921
ONBA (Buenos Aires)  (3)       -8803      -8581      -8611      -8600
ONRJ (Rio de Janeiro)            655        948       1149       1259
OP   (Paris)                      53         60         56         56
```

ORB	(Bruxelles)	305	289	277	255
PTB	(Braunschweig) (4)	2221	2232	2232	2229
RC	(Habana)	-	-	-	-
ROA	(San Fernando)	122	127	139	146
SCL	(Hong Kong)	388	293	242	230
SO	(Shanghai)	1733	1676	1638	1615
SU	(Moskva)	-7268	-7273	-7284	-7292
TL	(Chung-Li)	-23	-21	-20	-26
TP	(Praha)	-296	-286	-280	-274
TUG	(Graz)	-217	-210	-186	-173
UME	(Gebze-Kocaeli)	-3115	-3103	-3105	-3098
USNO	(Washington DC)(USNO MC)	11	15	12	13
VSL	(Delft)	-231	-231	-242	-250

BIPM , T 97 (2)

1 - Coordinated Universal Time UTC. (Cont.)

Date 1996 0h UTC		Jan 17	Jan 22	Jan 27
MJD		50099	50104	50109
Laboratory k		UTC-UTC(k)	(Unit is one nanosecond)	
AOS	(Borowiec)	-133	-60	3
APL	(Laurel)	2046	2059	2068
AUS	(Canberra)	-483	-511	-510
BEV	(Wien)	9152	8957	8816
BIRM	(Beijing)	-314	-399	-490
CAO	(Cagliari)	-	-	-
CH	(Bern)	247	260	276
CRL	(Tokyo)	258	245	223
CSAO	(Lintong)	-191	-194	-207
CSIR	(Pretoria)	5293	5345	5332
FTZ	(Darmstadt)	-273	-277	-264
GUM	(Warszawa)	-338	-342	-343
IEN	(Torino)	59	70	83
IFAG	(Wettzell)	-4792	-4813	-4843
IGMA	(Buenos Aires)	70	70	67
INPL	(Jerusalem)	-2923	-2950	-2976
IPQ	(Monte de Caparica)	-8338	-8380	-8428
JATC	(Lintong)	1592	1501	1545
KRIS	(Taejon)	307	300	279
LDS	(Leeds)	207	189	199
MSL	(Lower Hutt)	-5240	-5246	-5254
NAOM	(Mizusawa)	-3142	-3144	-3144
NAOT	(Tokyo)	-2959	-2909	-2873
NIM	(Beijing)	8077	8058	8093
NIST	(Boulder)	13	14	15
NPL	(Teddington)	53	58	62
NPLI	(New-Delhi)	-	-	-
NRC	(Ottawa)	85	108	120
NRLM	(Tsukuba)	-5063	-4996	-4934
OMH	(Budapest)	13981	14095	14361
ONBA	(Buenos Aires)	-8757	-8704	-8650
ONRJ	(Rio de Janeiro)	1394	1589	1645
OP	(Paris)	49	45	36
ORB	(Bruxelles)	263	263	249
PTB	(Braunschweig)	2221	2211	2199
RC	(Habana)	-	-	-
ROA	(San Fernando)	137	138	134
SCL	(Hong Kong)	207	222	241

SO (Shanghai)	1609	1577	1592
SU (Moskva)	-7302	-7305	-7315
TL (Chung-Li)	-9	-6	27
TP (Praha)	-277	-275	-262
TUG (Graz)	-165	-153	-145
UME (Gebze-Kocaeli)	-3088	-3078	-3075
USNO (Washington DC)(USNO MC)	11	11	8
VSL (Delft)	-244	-249	-258

BIPM , T 97 (3)

2 - International Atomic Time TAI and local atomic time scales TA(k).

The following tables give the computed values of TAI-TA(k).

Date 1995/96 0h UTC		Dec 28	Jan 2	Jan 7	Jan 12
MJD		50079	50084	50089	50094
Laboratory k		TAI-TA(k) (Unit is one nanosecond)			
APL (Laurel)		3430	3455	-	3488
AUS (Canberra)		-66277	-66447	-66571	-66712
CH (Bern)		-63930	-63789	-63668	-63539
CRL (Tokyo)		63571	63781	63996	64203
CSAO (Lintong)		6736	6707	6701	6612
F (Paris)		150472	150630	150795	150952
IEN (Torino)		-500	-493	-478	-464
INPL (Jerusalem)		-317888	-318567	-319231	-319901
JATC (Lintong)		12632	12711	12792	12743
KRIS (Taejon)		2160	2215	2261	2301
NIM (Beijing)		-6948	-6926	-6891	-6870
NISA (Boulder)	(5)	-45138497	-45138715	-45138934	-45139151
NRC (Ottawa)		25333	25369	25398	25428
PTB (Braunschweig)	(4)	-361179	-361168	-361168	-361171
RC (Habana)		-	-	-	-
SO (Shanghai)		-45790	-45822	-45852	-45875
SU (Moskva)	(6)	27242732	27242727	27242716	27242708
USNO (Washington DC)	(7)	-34738589	-34738918	-34739249	-34739579

Date 1996 0h UTC		Jan 17	Jan 22	Jan 27
MJD		50099	50104	50109
Laboratory k		TAI-TA(k) (Unit is one nanosecond)		
APL (Laurel)		3509	3522	3531
AUS (Canberra)		-66858	-66986	-67095
CH (Bern)		-63399	-63268	-63133
CRL (Tokyo)		64408	64616	64821
CSAO (Lintong)		6578	6510	6432
F (Paris)		151110	151273	151430
IEN (Torino)		-449	-438	-428
INPL (Jerusalem)		-320563	-321252	-321950
JATC (Lintong)		12766	12717	12698
KRIS (Taejon)		2341	2375	2396
NIM (Beijing)		-6801	-6799	-6752
NISA (Boulder)	(6)	-45139369	-45139585	-45139802
NRC (Ottawa)		25460	25504	25538
PTB (Braunschweig)		-361179	-361189	-361201
RC (Habana)		-	-	-
SO (Shanghai)		-45884	-45913	-45892
SU (Moskva)	(7)	27242698	27242695	27242685
USNO (Washington DC)	(8)	-34739909	-34740238	-34740571

BIPM , T 97 (4)

3 - Notes on sections 1 and 2.

(1) AOS . Apparent time step of UTC-UTC(AOS) of - 370 ns on
MJD = 50083 due to calibration of the GPS link.

(2) IGMA. Time step of UTC(IGMA) of + 300 ns on MJD = 50083.0

(3) ONBA. Corrected values of UTC-UTC(ONBA)

MJD	UTC-UTC(ONBA)	MJD	UTC-UTC(ONBA)
49939	-3883 ns	50009	-9348 ns
49949	-4983 ns	50019	-9410 ns
49959	-6021 ns	50029	-9401 ns
49969	-6455 ns	50039	-9393 ns
49979	-7297 ns	50049	-9309 ns
49989	-7991 ns	50059	-9233 ns
49999	-8693 ns	50069	-9095 ns

(4) PTB . Apparent time step of UTC-UTC(PTB) and TAI-TA(PTB) of + 9 ns
on MJD = 50083 due to calibration of the GPS link.

(5) NIST. TA(NISA) designates the scale AT1 of NIST.

(6) SU . Listed values are TAI-TA(SU) - 2.80 seconds.

(7) USNO. TA(USNO) designates the scale A1(MEAN) of USNO.

BIPM , T 97 (5)

4 - [UTC - GPS time] and [TAI - GPS time].

[UTC - GPS time] = -10 s + C0 (until 1996 January 1, 0h UTC)
[UTC - GPS time] = -11 s + C0 (from 1996 January 1, 0h UTC)
[TAI - GPS time] = 19 s + C0.

Daily values of C0 are given in the following table. They are obtained as follows: the GPS data taken at the Paris Observatory, for highest elevation, are first corrected for precise satellite ephemerides and for measured ionospheric delays, and then smoothed to obtain daily values of [UTC(OP) - GPS time] at 0h UTC; daily values of C0 are derived from them using linear interpolation of [UTC - UTC(OP)].

For a given day, where N measurements are used for estimation of C0 :
- the dispersion of individual measurements is characterized by a standard deviation s,
- the daily C0 value is characterized by the standard deviation of the mean s/sqrt(N).

Date 1995/96 0h UTC	MJD	C0 (ns)	s (ns)	s/sqrt(N) (ns)
Dec 28	50079	51	40	8
Dec 29	50080	53	38	8
Dec 30	50081	53	46	10
Dec 31	50082	55	50	10
Jan 1	50083	56	33	7
Jan 2	50084	55	39	8
Jan 3	50085	51	34	7
Jan 4	50086	49	37	8
Jan 5	50087	51	46	10
Jan 6	50088	48	45	11
Jan 7	50089	40	46	10
Jan 8	50090	36	52	12
Jan 9	50091	35	40	9
Jan 10	50092	35	48	10
Jan 11	50093	35	39	8
Jan 12	50094	33	45	10
Jan 13	50095	34	36	7
Jan 14	50096	36	37	8
Jan 15	50097	39	46	10
Jan 16	50098	40	50	11
Jan 17	50099	40	35	8
Jan 18	50100	39	41	9
Jan 19	50101	38	47	10
Jan 20	50102	40	43	10
Jan 21	50103	41	30	7
Jan 22	50104	42	42	9
Jan 23	50105	44	57	12
Jan 24	50106	49	36	7
Jan 25	50107	54	55	12
Jan 26	50108	55	33	7
Jan 27	50109	54	39	8

BIPM , T 97 (6)

5 - [UTC - GLONASS time].

[UTC - GLONASS time] = C1 (modulo 1 s).

From his current observations of both the GPS and GLONASS satellite systems Prof. P. Daly, University of Leeds, establishes and reports [GPS time - GLONASS time] at five-day intervals, together with the standard deviation s of his daily GLONASS data. C1 is then derived using [UTC - GPS time] of section 4.

Date 1995/96 0h UTC	MJD	C1 (ns)	s (ns)
Dec 28	50079	-24569	60
Jan 2	50084	-24677	38
Jan 7	50089	-24812	37
Jan 12	50094	-24960	45
Jan 17	50099	-25088	42
Jan 22	50104	-25264	46
Jan 27	50109	-25390	45

6 - Difference between the normalized frequencies of EAL and TAI.

```
           Interval of validity                f(EAL)-f(TAI)

     1995 Dec. 28 - 1996 Feb. 26      50079-50139       7.34x10**-13

     New steering correction foreseen for March-April 1996

     1996 Feb. 26 - 1996 Apr. 26      50139-50199       7.33x10**-13


  7 - Duration of the TAI scale interval.

     The following table gives the duration of the TAI scale interval
    expressed as its departure d from the SI second on the rotating geoid,
    together with its relative uncertainty s. This is obtained, on the given
    period of estimation, by comparison of the TAI frequency :

      - with the frequency, corrected for the black-body radiation shift, of
    a given individual primary frequency standard (s is then the last
    communicated estimate of the uncertainty of the standard frequency), and

     - with a combination computed by the BIPM of all available measurements
    from PTB CS1, PTB CS2, PTB CS3, NIST-7 and SU MCsR 102 consistently
    corrected for the black-body radiation shift (s is then estimated by the
    BIPM taking into account the individual uncertainties and parameters
    characteristic of TAI stability).

        Standard         Period of           d         s
                         estimation      (10**-14 s) (10**-14)

        SU MCsR 102      49959-50019        +3.5       5.0
        SU MCsR 102      50019-50079        +4.3       5.0
        PTB-CS2          50049-50109        +2.1       1.5

        BIPM estimate    50049-50109        +2.7       1.0

--------------------------------------------------------------------------
Name: Hawai Konate              Bureau International des Poids et Mesures
E-mail: hkonate@bipm.fr         Section du temps - Pavillon de Breteuil
                                92312 - Sevres cedex - FRANCE

Phone Number : (+33) 1 45 07 70 72
Fax Number   : (+33) 1 45 07 70 59
--------------------------------------------------------------------------
```

C.4 Schaltsekunden in UTC

Seit Einführung der UTC-Zeitskala am 1. Jänner wurden 20 Schaltsekunden eingefügt. Wann eine Schaltsekunde eingeführt wird, wird vom *IERS*, dem *International Earth Rotation Service*, das die Erdrotationsparameter überwacht, entschieden. In folgender Tabelle sind alle bisher (1996) aufgetretenen Schaltsekunden aufgelistet:

1972 Juni 30	1972 Dez. 31	1973 Dez. 31	1974 Dez. 31
1975 Dez. 31	1976 Dez. 31	1977 Dez. 31	1978 Dez. 31
1979 Dez. 31	1981 Juni 30	1982 Juni 30	1983 Juni 30
1985 Juni 30	1987 Dez. 31	1989 Dez. 31	1990 Dez. 31
1992 Juni 30	1993 Juni 30	1994 Dez. 31	1995 Dez. 31

Seit der Schaltsekunde am Jahresende 1995/1996 beträgt die Differenz zwischen UTC und TAI exakt 30 Sekunden.

D Abkürzungen

AFB Air Force Base

A-S Anti-Spoofing

C/A Coarse/Aquisition

CIO Conventional International Origin

CSOC Consolidated Space Operations Center

DGPS Differential GPS

DLL Delay Lock Loop

DoD Departement of Defense

DOP Dilution of Precision

DoT Departement of Transportation

ECEF Earth-Centered-Earth-Fixed

FOC Full Operational Capability

FRP Federal Radio Navigation Plan

GDOP General Dilution of Precision

GLONASS Global Navigation Satellite System

GNSS Global Navigation Satellite System

GPS Global Positioning System

GPS CC GPS Composite Clock

GPST GPS Time

HDOP Horizontal Dilution of Precision

IERS International Earth Rotation Service

JD Julianisches Datum, Julian Date

MJD Modifiziertes Julianisches Datum, Modified Julian Date

NAVSTAR Navigation System with Timing and Ranging

PDOP Position Dilution of Precision

PLL Phase Locked Loop

PPS Precise Positioning Service

PRN Pseudo Random Noise

RAIM Receiver Autonomous Integrity Monitoring

RDS Radio Data System

RTCM Radio Technical Commission for Maritime services

SA Selective Availability

SPS Standard Positioning Service

TAI Temps Atomique International

TD Temps Dynamique

TDB Temps Dynamique Baricentrique

TDT Temps Dynamique Terrestrique

TDOP Time Dilution of Precision

USNO United States Naval Observatory

UT Universal Time

UTC Universal Time Coordinated

VDOP Vertical Dilution of Precision

WGS World Geodetic System

Literaturverzeichnis

Absolute Time Incorporation. *User Manual, Model 100 A/B GPS-Clock*, 1995.

David Allan. Time and frequency (time-domain) characterization, estimation, and prediction of precision clocks and oscillators. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, UFFC-34(6):647–654, November 1987.

C. Audoin. Caesium Beam Frequency Standards: Classical and optical pumped. *Metrologia*, 29:113–134, 1992.

Ball Efratom. *Description and Operation Manual, Subminiature Rubidium Oscillator Model FRS*, 1989.

Manfred Bauer. *Vermessung und Ortung mit Satelliten: NAVSTAR-GPS und andere Satellitengestützte Navigationssyteme*. Wichmann, Karlsruhe, 2., völlig neu überarbeitete Ausgabe, 1992.

Peter H. Dana. Global Positioning System (GPS) time dissemination for real-time applications. *to appear in Journal of Real-Time Systems*, 1996.

Torsten Fließbach. *Allgemeine Relativitätstheorie*. Spektrum Akademischer Verlag, Heidelberg, Berlin, Oxford, 2., überarbeitete Ausgabe, 1995.

Helmut Hellwig. Environmental sensitivities of precision frequency sources. *IEEE Transactions on Instrumentation and Measurement*, 39(2):301–306, April 1990.

Wolfgang Hilbert, Editor. *Funkuhrtechnik*. Oldenbourg, München, Wien, 1988.

Wolfgang Hilbert, Editor. *Funkuhren, Zeitsignale und Normalfrequenzen*. Verlag Sprache und Technik, Groß-Bieberau/Odenwald, Im Geisner 11, 1993.

Bernhard Hofmann-Wellenhof, Herbert Lichtenegger, und James Collins. *Global Positioning System, Theory and Practice*. Springer-Verlag, Wien, New York, dritte, revidierte Ausgabe, 1994.

P. Lesage und C. Audoin. Characterization and measurement of time and frequency stability. *Radio Science*, 14(4):521–539, 1979.

Jean Meeus. *Astronomische Algorithmen*. Johann Ambrosius Barth, Leipzig, Berlin, Heidelberg, 2., durchgesehene Ausgabe, 1994.

Motorola. *Oncore User's Guide*, Juni 1994.

NATO Navstar GPS Project Steering Committee. *Technical Characteristics of the NavSTAR GPS*, Juni 1991.

NAVSTAR GPS Joint Program. *GPS Interface Control Document ICD-GPS-200 GPS Space Segment and Navigation User Interfaces*, Juli 1992.

NavStar Systems Limited. *Installation and Operating Manual, Navsymm Timing and Frequency Receiver NTFR-S*, März 1995. Issue 2.0.

Rockwell. *NavCore Designer's Guide*, vorläufige Ausgabe, Oktober 1993.

Ulrich Schmid. Synchronized UTC for distributed real-time systems. *IFAC Real Time Programming*, Seiten 101–107, 1994.

Klaus Schossmaier und Ulrich Schmid. UTCSU function specification. *Technical Report 183/1-56, Vienna University of Technology, Departement of Automation*, July 1995.

Günter Seeber. *Satellitengeodäsie*. de Gruyter, Berlin, New York, 1989.

Samuel R. Stein. Frequency and time — their measurement an characterization. *Precision Frequency Control*, 2:191–416, 1985.

Trimble Navigation. *SVeeSix-CM2 Embedded GPS Core Module System Designer Reference Guide*, 1994.

D.E. Wells, N. Beck, D. Delikaraoglou, A. Kleusberg, E.J. Krawiwsky, G. Lachapelle, R.B. Langley, M. Nakiboglu, K.P. Schwarz, J.M. Tranquilla, und P. Vaníĉek. *Guide to GPS Positioning*. Canadian GPS Associates, Fredericton, N.B., Canada, 1986.

Stichwortverzeichnis