Entwicklung einer Softwarelösung zur hochgenauen Zeit- und Positionsbestimmung auf GPS-Basis für Car2X-Szenarien

Benedikt Kleinmeier

Hochschule München

Fakultät für Informatik und Mathematik

11. Januar 2015

Übersicht

Übersicht

Car2X-Kommunikation

Ziel: Erhöhung der Sicherheit

- Techniken wie ABS und ESP helfen während Gefahrensituationen.
- Car2X-Kommunikation warnt vor Gefahrensituationen.

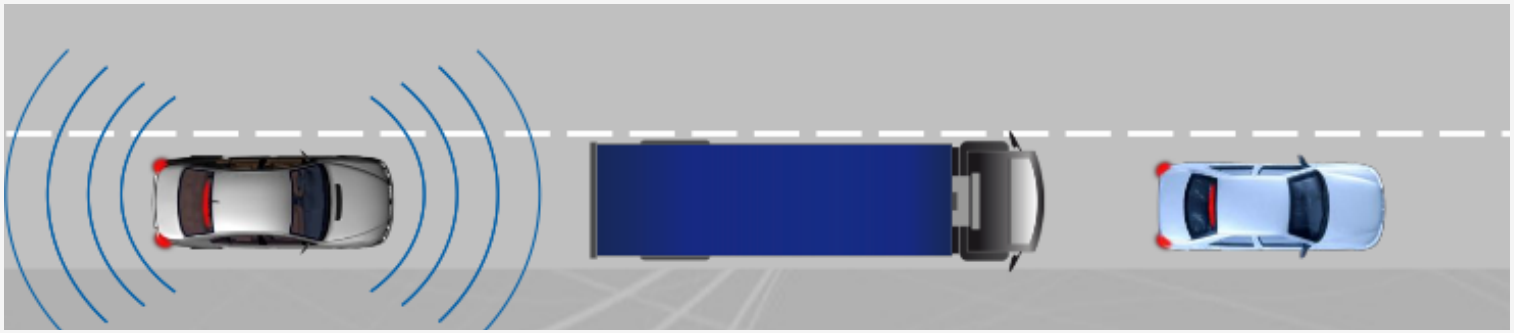


Abbildung: Car2X-Kommunikation

Car2X-Kommunikation: Standard

Standard

- ETSI spezifiziert herstellerübergreifenden Standard.

Architektur

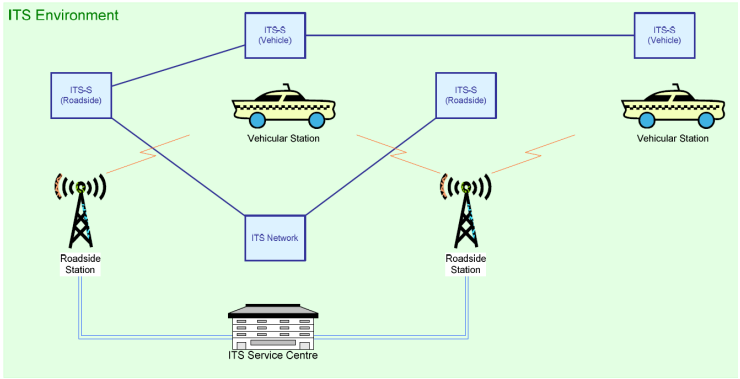


Abbildung: Architektur ITS-Anwendungen

- Vehicular Stations
- Roadside Stations
- ITS Service Centre

Anwendungsfälle aus ETSI-Standard

Emergency Electronic Brake Lights

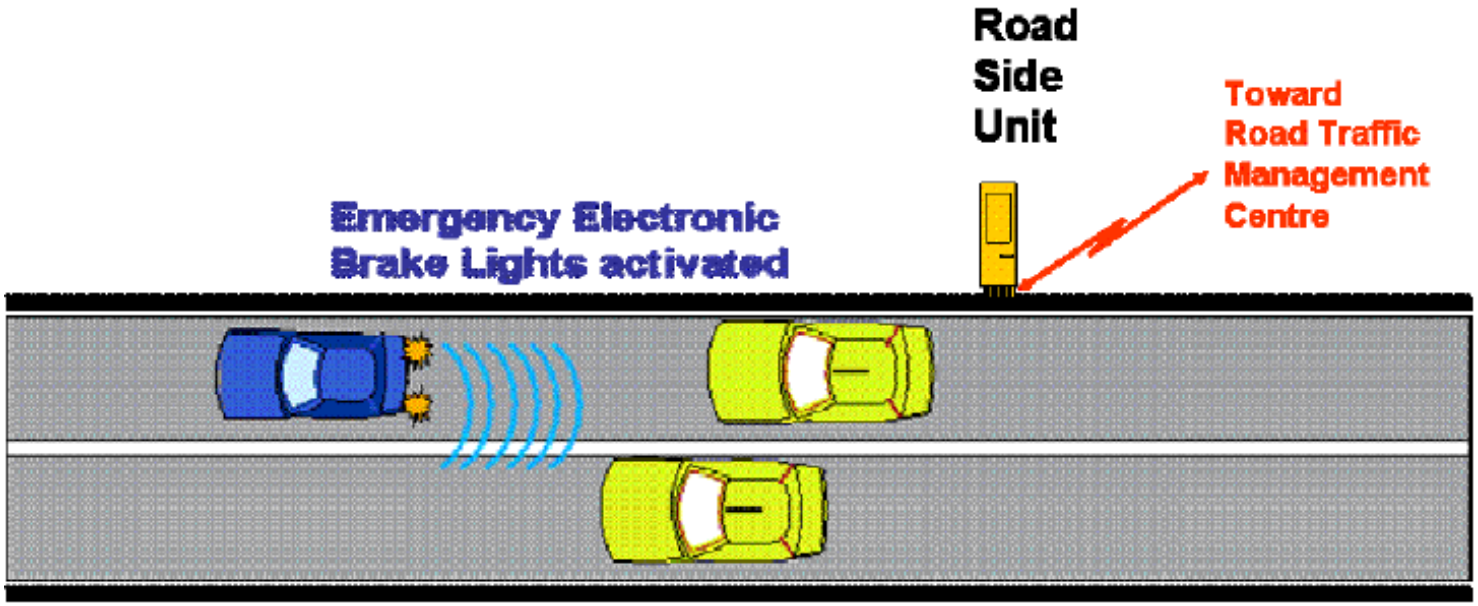


Abbildung: Anwendungsfall Emergency Electronic Brake Lights

Anwendungsfälle aus ETSI-Standard

Intersection Collision Warning

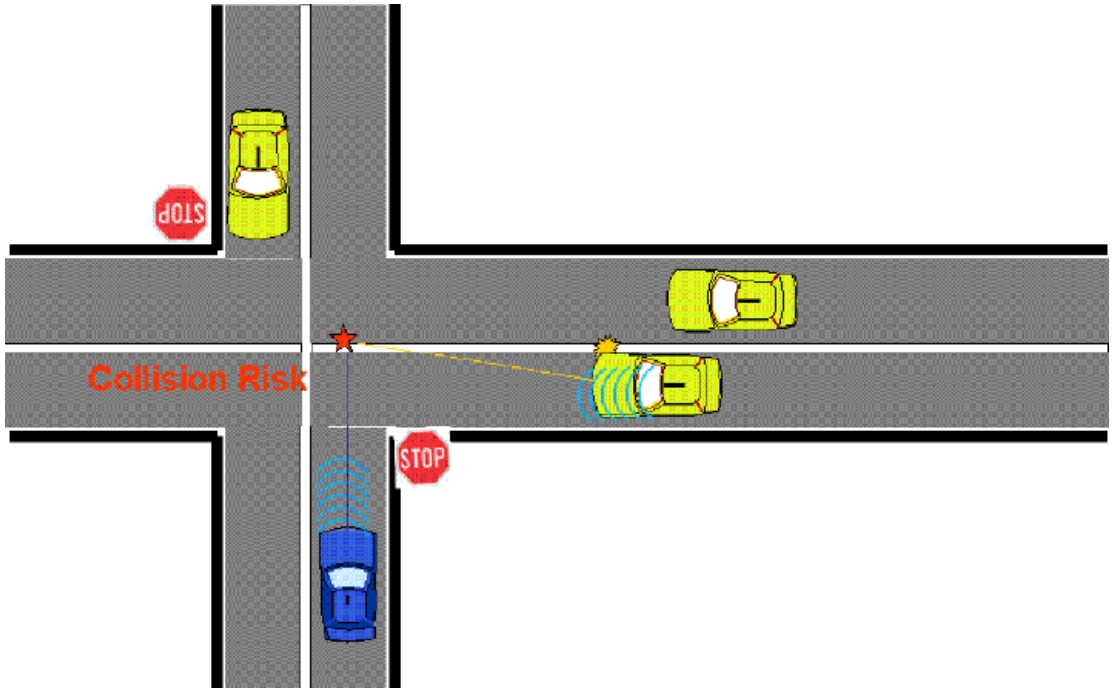


Abbildung: Anwendungsfall Intersection Collision Warning

Anwendungsfälle aus ETSI-Standard

Co-Operative Forward Collision Warning

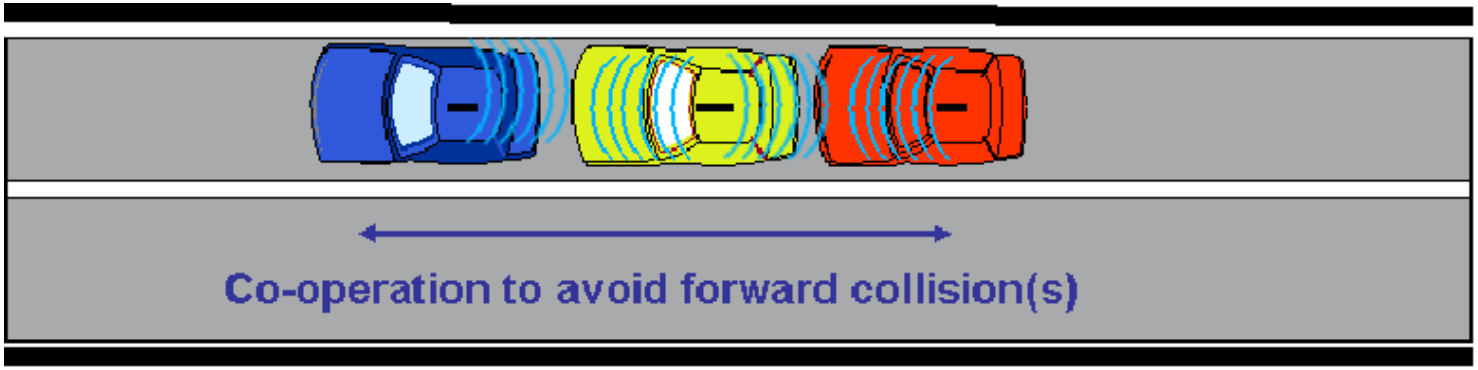


Abbildung: Anwendungsfall Co-Operative Forward Collision Warning

Übersicht

Stand der Technik

Car2X-Kommunikation bei Automobilherstellern

- Zeit- und Positionsbestimmung durch GPS.
- Fusionierung von GPS- und Fahrzeugdaten.
- Car2X-Kommunikation durch WLAN und Mobilfunk.

Satellitennavigation

- Weltweite Zeit- und Positionsbestimmung.
- Basiert auf Laufzeitmessung von Signalen.

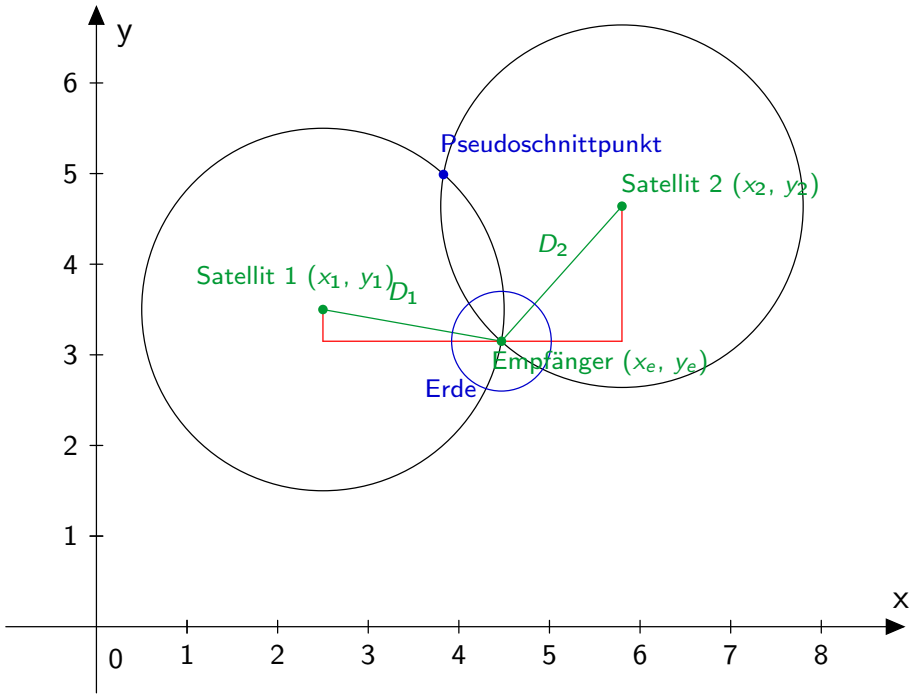


Abbildung: Positionsbestimmung durch Signale von zwei Satelliten

Positionsbestimmung

Positionsbestimmung (2D)

$$\begin{aligned}(x_e - x_1)^2 + (y_e - y_1)^2 &= D_1^2 \\ (x_e - x_2)^2 + (y_e - y_2)^2 &= D_2^2\end{aligned} \tag{1}$$

Distanzbestimmung

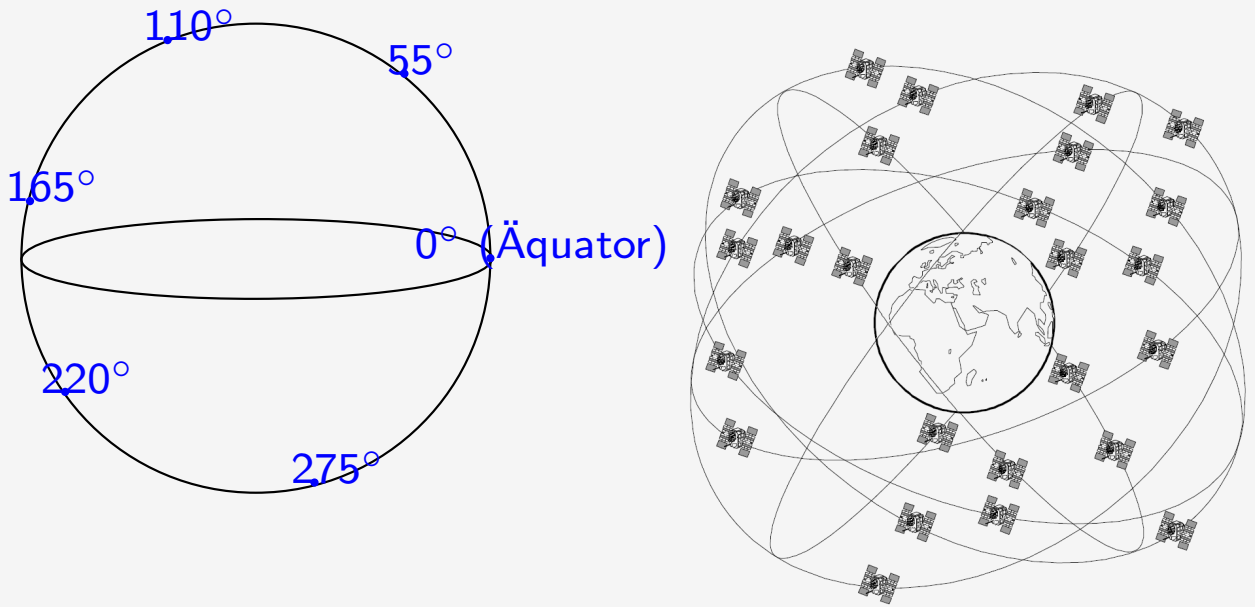
$$D = t_{\text{Empfänger}} - t_{\text{Sender}} \cdot c = \Delta t \cdot c \tag{2}$$

Positionsbestimmung (3D) mit konstantem Fehler

$$(x - x_n)^2 + (y - y_n)^2 + (z - z_n)^2 = [(\Delta t_n + e) \cdot c]^2 \text{ mit n} = [1, 4] \tag{3}$$

Global Positioning System (GPS)

- Seit 1995 in Betrieb.
- Weltweite Abdeckung durch 31 Satelliten.



(a) 2D (b) 3D

Abbildung: GPS-Satellitenbahnen

GPS: Genauigkeit

Genauigkeit von GPS

- Horizontal: $\leq$ 9 m
- Vertikal: $\leq$ 15 m
- Zeit: $\leq$ 40 ns

Genauigkeit GPS-Empfänger u-blox NEO-7N

- Horizontal: 2,5 m (2,0 m mit DGPS)
- Update-Rate: 10 Hz

GPS-Empfänger: Aufbau

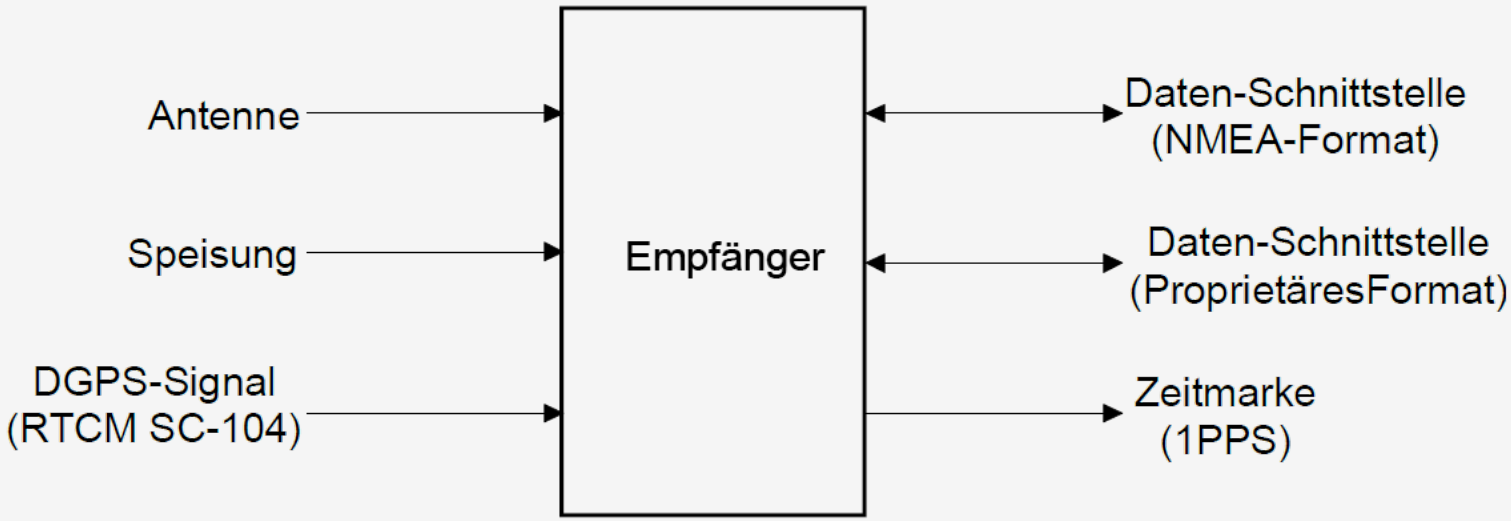


Abbildung: Aufbau GPS-Empfänger

Beispiel: NMEA-Satz RMC

$GPRMC,123005.00,A,4807.99950,N,01131.66952,E,2.027,,130814,,
,A*76

Pulse per Second (PPS)

Problem

Verzögerung zwischen Eintreffen der Satellitensignale an Empfängerantenne und Verarbeitung im Computer.

Idee

Empfänger erzeugt Interrupt, wenn Daten an Antenne anliegen.

Berechnung Verzögerungszeit t

1. Zeitstempel t_1 bei Interrupt.
2. Zeitstempel t_2, wenn Computer Daten von Empfänger verarbeitet.

$$t = t_2 - t_1 \tag{4}$$

Zeitmessung in Software

Software

immer abhängig von Hardware.

Timer

ist Hardware welche Zeitmessung ermöglicht.

Verfügbare Timer 80x86-Architektur

- RTC: 2 bis 8.192 Hz
- PIT: 100 bis 1.000 Hz
- HPET: Mindestens 10 MHz
- TSC: Mit CPU-Taktfrequenz

Scheduling in Linux

Linux ist GPOS

Linux kein RTOS $\Rightarrow$ Keine Garantie von Antwortzeiten!

Scheduler

- Timer erzeugt kontinuierlich Interrupts.
- Bei jedem Timer-Interrupt entscheidet Scheduler, ob Prozesswechsel stattfinden soll.

Scheduling-Strategien in Linux

Drei nicht echtzeitfähige Strategien

1. SCHED_OTHER
2. SCHED_BATCH
3. SCHED_IDLE

Zwei (bedingt) echtzeitfähige Strategien

1. SCHED_FIFO
2. SCHED_RR

Erhöhung der Echtzeitfähigkeit in Linux

Echtzeitfähigkeit

nur, wenn OS (fast) überall unterbrechbar.

In Linux

ISRs von keinem Prozess unterbrechbar.

CONFIG_PREEMPT_RT-Patch

- ISRs zu unterbrechbaren Kernel-Threads.
- Priority Inheritance für Semaphoren in Kernel-Code.
- Busy Waiting in Kernel Code ersetzt durch Mutexe.

Aufgaben Betriebssystem

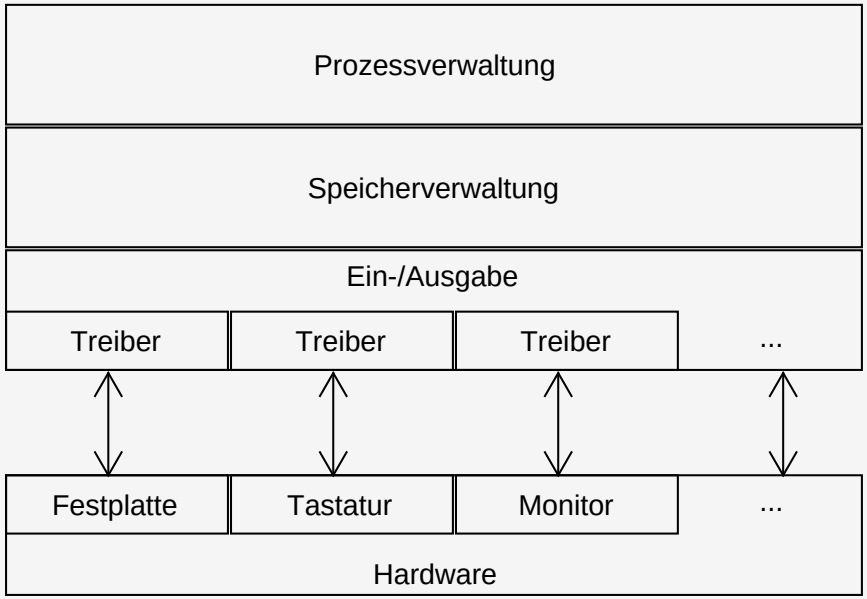


Abbildung: Aufgaben eines Betriebssystems

Ein-/Ausgabe in Linux

- Kapselung von Aufgaben in Module (separate Objektdateien).
- Ein-/Ausgabe in Subsyteme gegliedert, z.B. USB oder PCI.
- Unix-Philosophie: „everything is a file“
  $\Rightarrow$ Geräte unter `/dev` als Dateien zugänglich

Aufgaben eines Linux-Treibers

- Registrierung am entsprechenden Subsystem.
- Implementierung der Methoden die für Hardware zulässig sind.

Zusammenspiel Linux-Kernel und Treiber

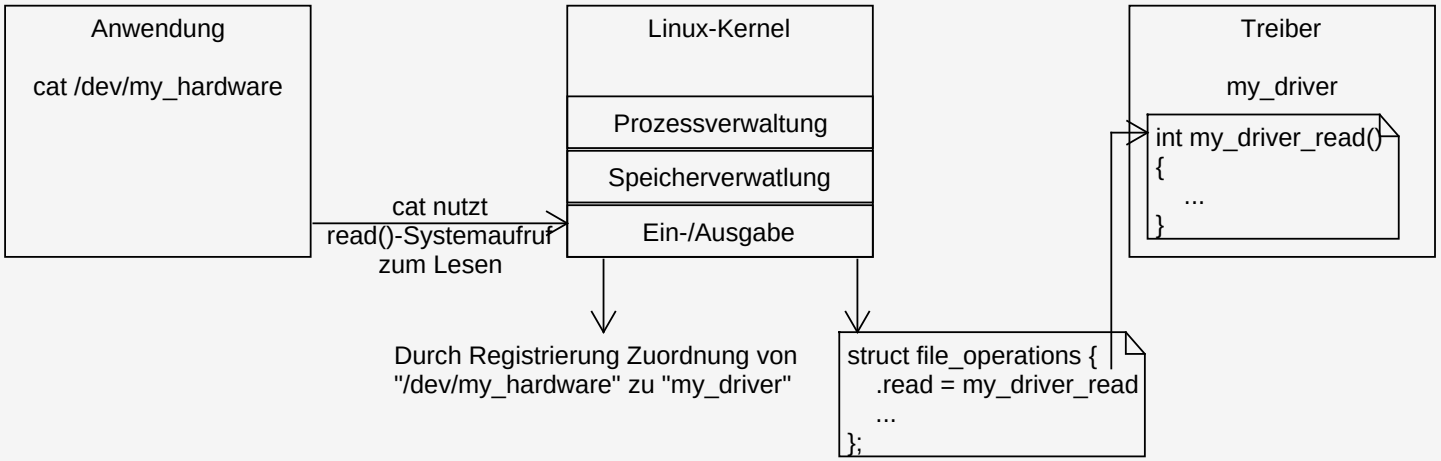


Abbildung: Zusammenspiel Linux-Kernel und Treiber

Übersicht

ARTiS PC

Abbildung: ARTiS PC in Metallgehäuse

- Intel Atom Z520PT 1.33 GHz
- 1 GB RAM
- u-blox NEO-7 GPS-Empfänger
- GNU/Linux Debian 7.5

GPS-Empfänger u-blox NEO-7

- Anbindung über USB.
- PPS-Detektion durch FPGA.

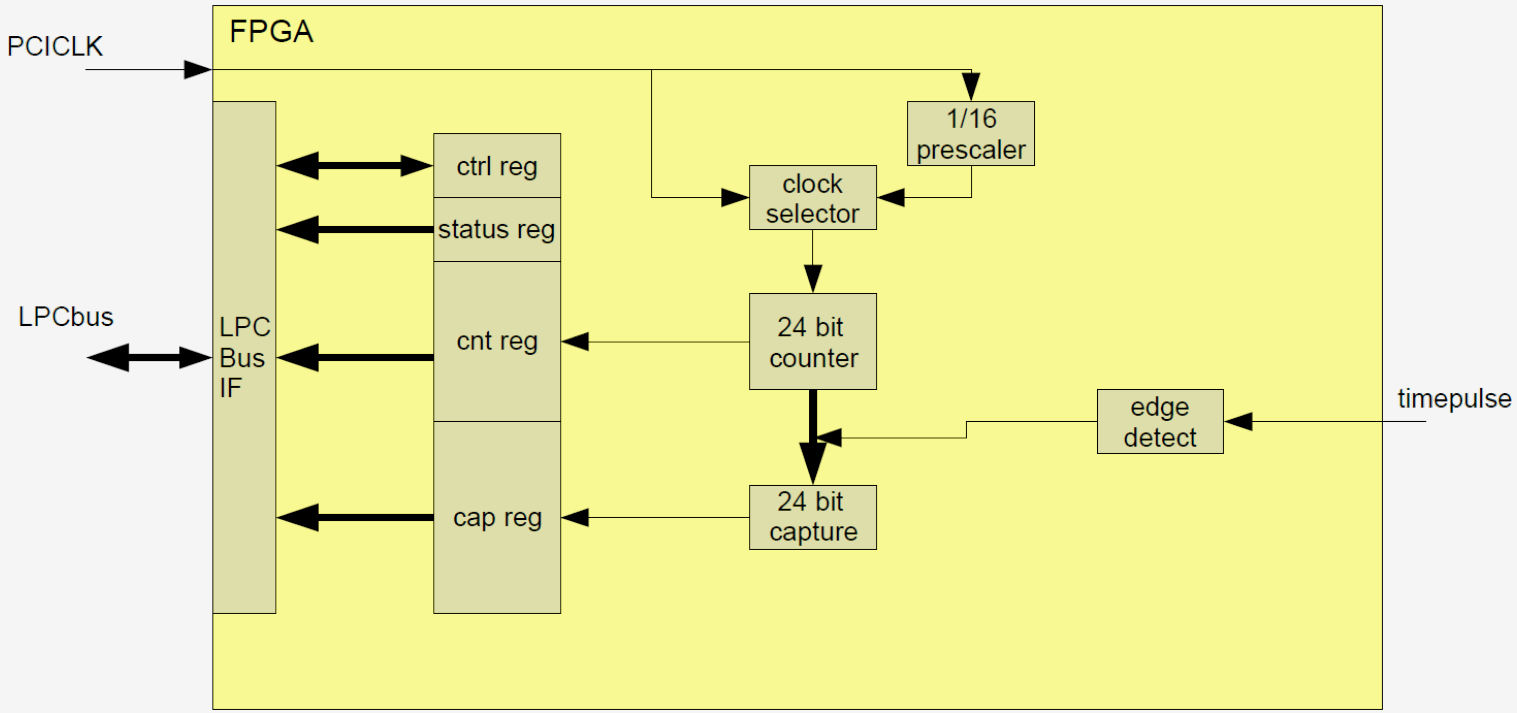


Abbildung: Blockschaltbild FPGA

Übersicht

Anforderungen und Verifikation

Anforderungen

- Fraunhofer-Institut: „Hardware bestmöglich nutzen.“
- Anwendungsfälle:
  - Car2X-Nachrichten minimal mit 10 Hz senden.
  - Positionsgenauigkeit ≤ 1 m.
- Ressourcensparsamkeit (CPU/RAM).

Verifikation

- Zeit gegen Zeit laut Network Time Protocol (NTP) prüfbar.
- Prüfbar, ob Update-Intervall von 10 Hz eingehalten werden kann.
- CPU-/RAM-Auslastung durch `top` prüfbar.

Testautomatisierung

- Auf Kernel-Ebene kein Unit-Testing-Framework.
- Bash-Skripte für automatisierte Tests.
- `awk`, `grep` und `sed` zur Aufbereitung/Filterung.
- Python-Skripte zur Konvertierung in CSV-Dateien.

Probleme durch Hardware

Probleme

- Hardware nicht verfügbar.
- Anbindung des GPS-Empfängers nicht dokumentiert.
- FPGA-Firmware fehlerhaft:
  - Auslesen der FPGA-Register nicht möglich.
  - Capture-Register nicht aktualisiert.

Inbetriebnahme

- GPS-Empfänger mit Linux-Daemon `gpsd` getestet.
- FGPA mit selbst geschriebenen Programm getestet (`ioperm()`, `inb()`, ...).

1. Ansatz: gpsd und ntpd

Konzept

- Linux-Daemon gpsd
  - übernimmt USB-Kommunikation mit GPS-Empfänger.
  - parst NMEA- bzw. UBX-Daten des Empfängers.
  - Daten werden über TCP-Port 2947 im JSON-Format bereitgestellt.
- Linux-Daemon ntpd wir genutzt um Zeit zu setzen.
  $\Rightarrow$ „Rückwärtslaufen“ der Uhr wird verhindert.

Notwendige Modifikation

Verarbeitung des des PPS-Signals über den LPC-Bus.

Evaluierung I

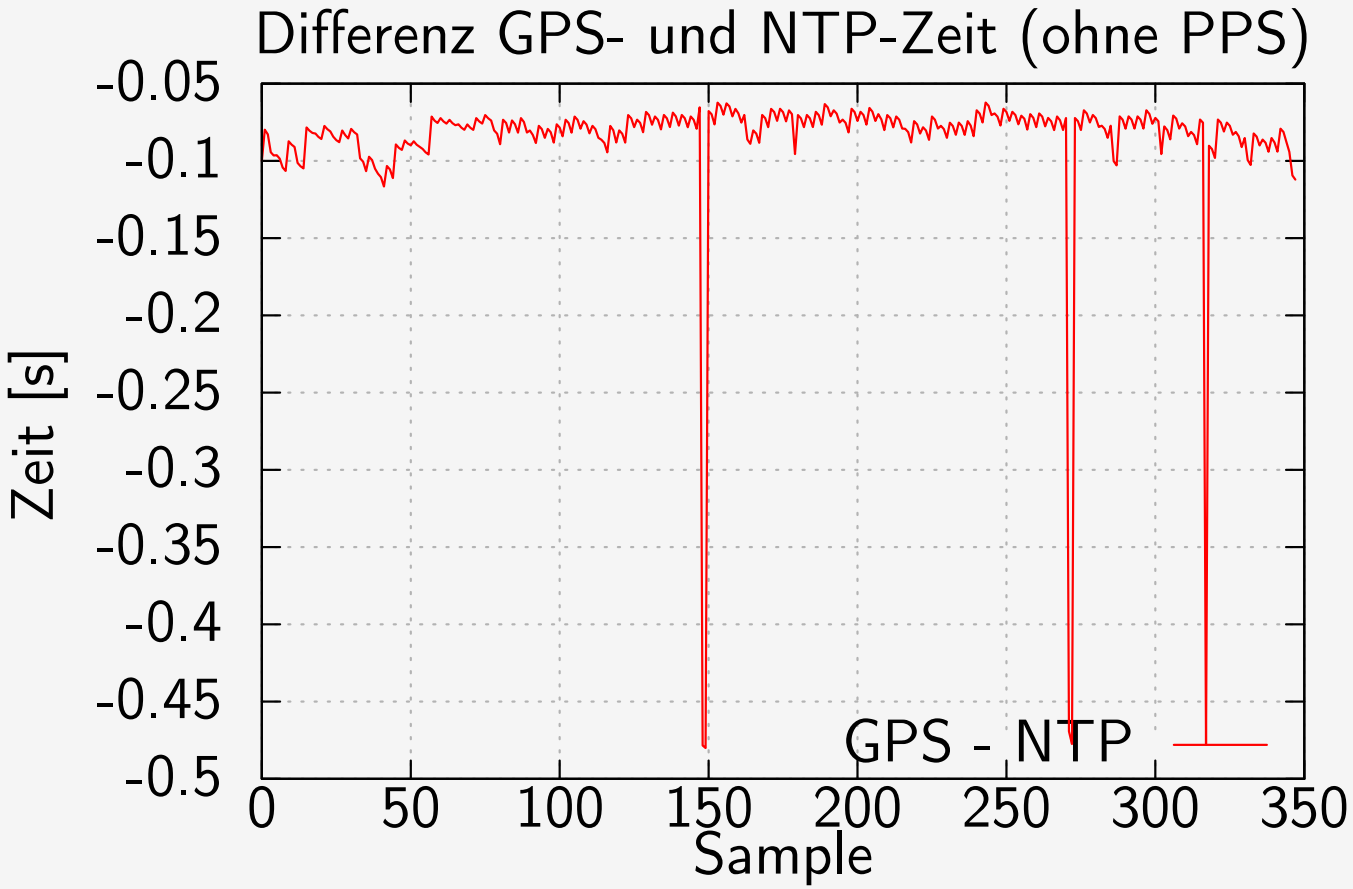


Abbildung: gpsd ohne PPS: Differenz GPS- und NTP-Zeit

Evaluierung II

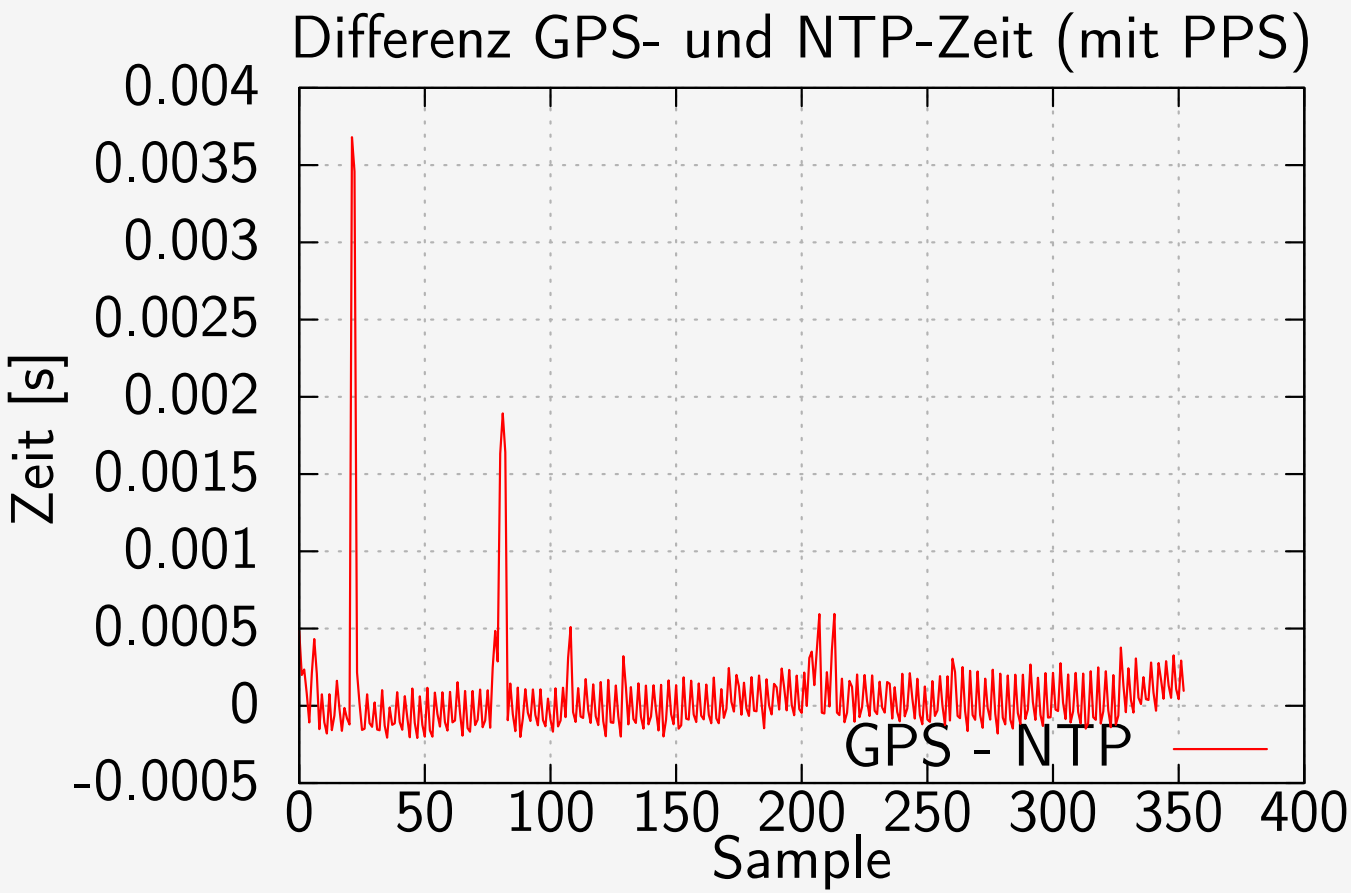


Abbildung: gpsd mit PPS: Differenz GPS- und NTP-Zeit

Evaluierung III

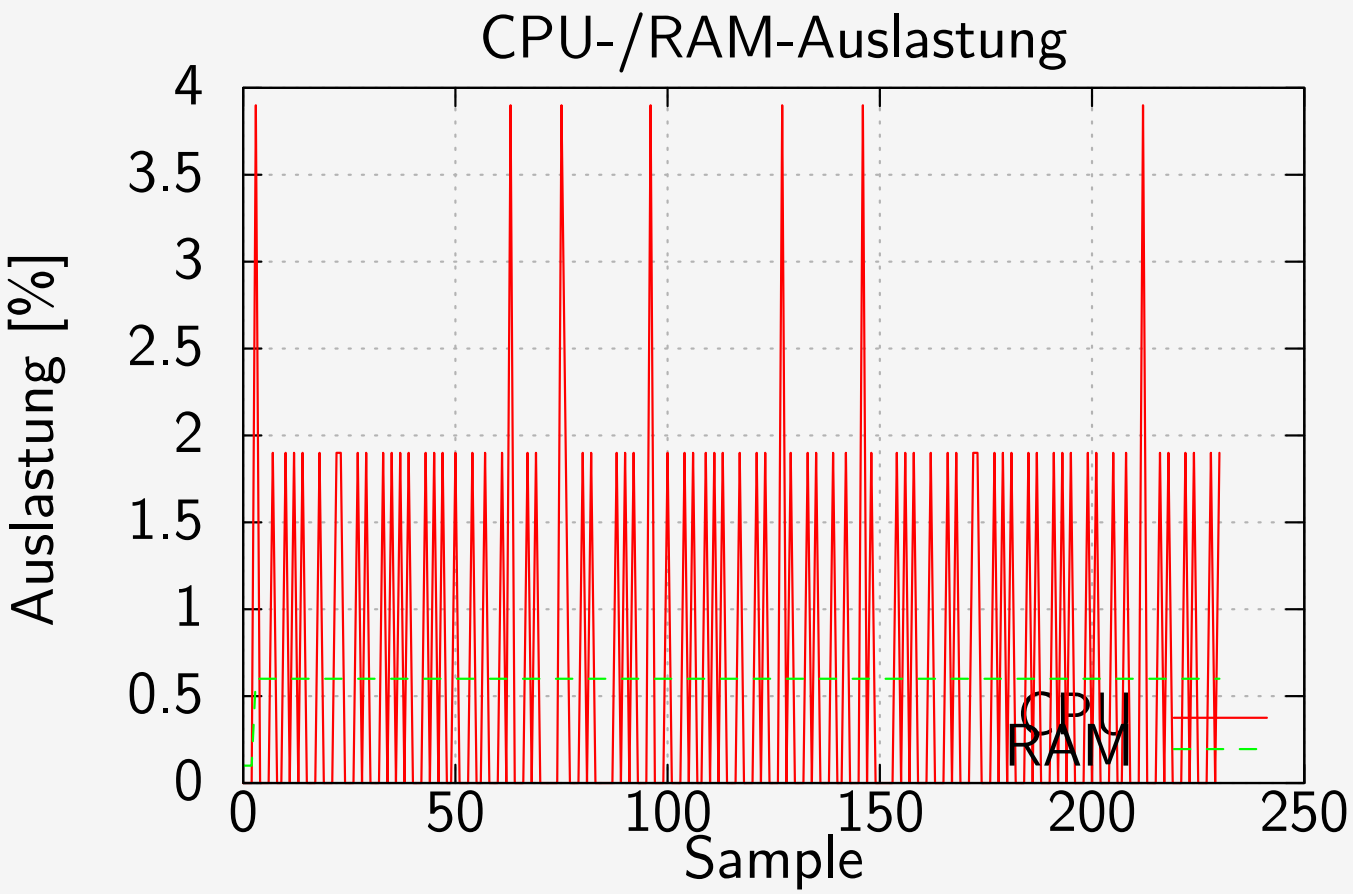


Abbildung: gpsd mit PPS: CPU- und RAM-Auslastung

2. Ansatz: Eigener Treiber als Kernel-Modul

Treiber

- Treiber ermöglicht Zugriff auf GPS-Empfänger.
- Abbildung als `/dev/ublox`:
  - `read()` liefert GPS-Daten zurück.
  - `ioctl()` liefert u.a. PPS-Daten zurück.
- Treiber pollt kontinuierlich GPS-Empfänger und puffert Daten.

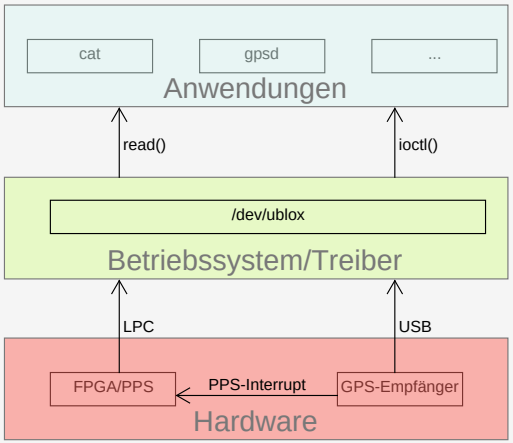


Abbildung: Konzept des Linux-Treibers

Implementierung

Aufgaben eines Linux-Treibers

- Registrierung am entsprechenden Subsystem.
  ⇒ USB: `usb_register(struct usb_driver)`
- Implementierung der Methoden die für Hardware zulässig sind.
  ⇒ `read()` und `ioctl()`
- Eigener Kernel-Thread pollt in regelbarem Intervall nach neuen GPS-Daten.

Listing 1: Pseudocode Kernel-Modul

```
1 while (module_is_loaded && no_error) {
2   poll_gps_receiver_via_usb();
3   parse_data();
4   filter_data_for_nav_sol();
5   sleep();
6 }
```

Evaluierung I

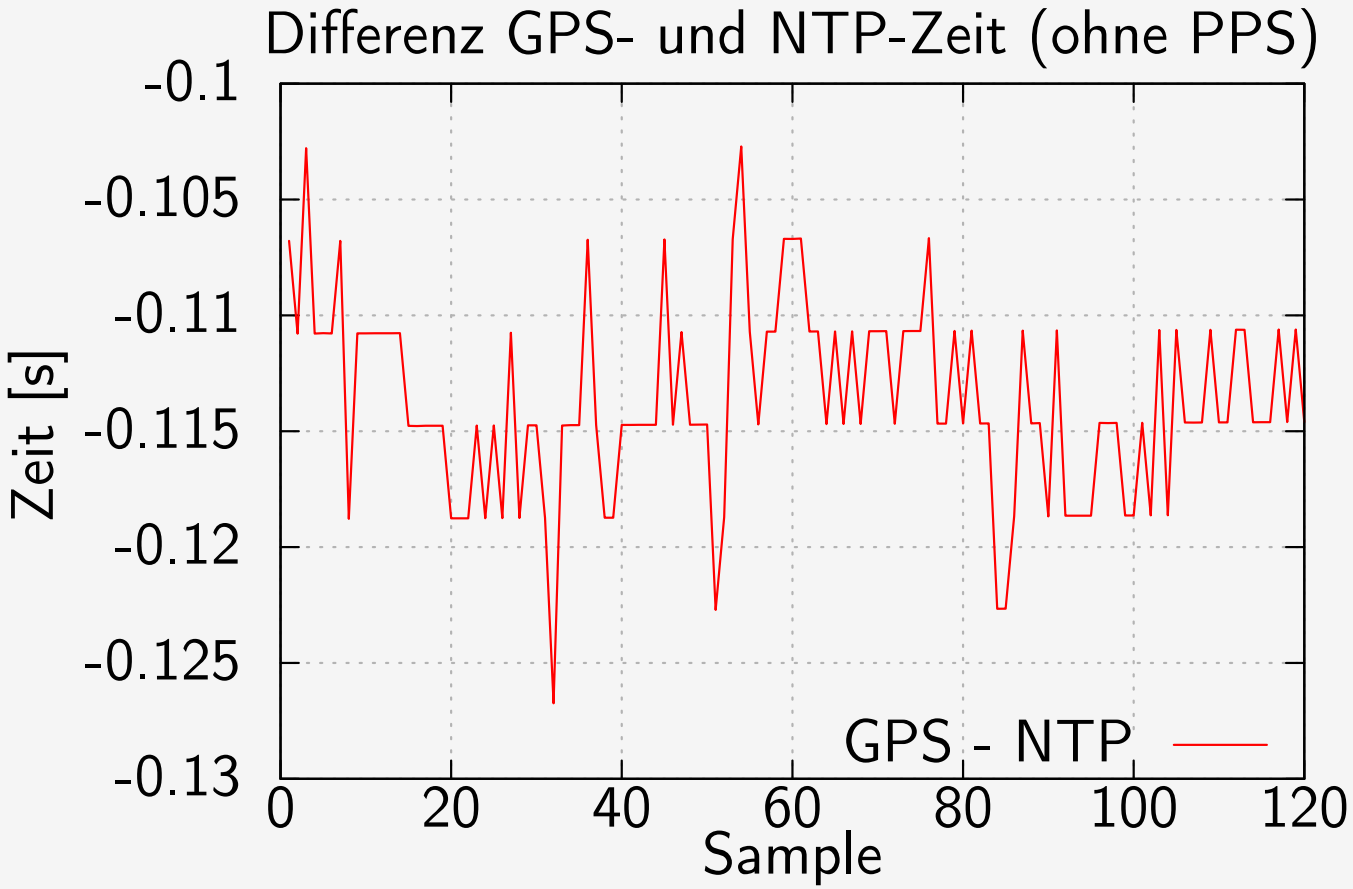


Abbildung: ublox ohne PPS: Differenz GPS- und NTP-Zeit

Evaluierung II

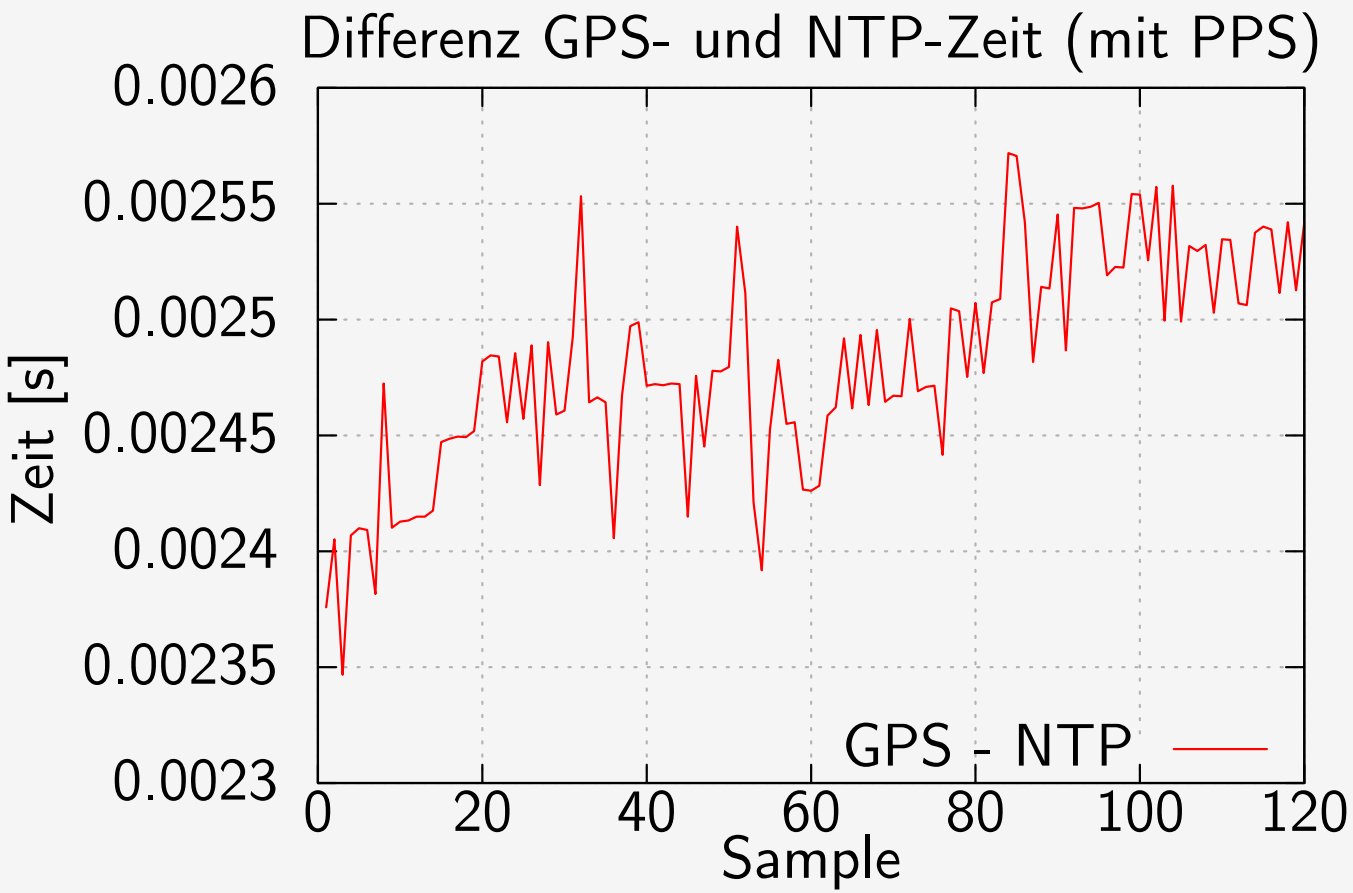


Abbildung: ublox mit PPS: Differenz GPS- und NTP-Zeit

Evaluierung III

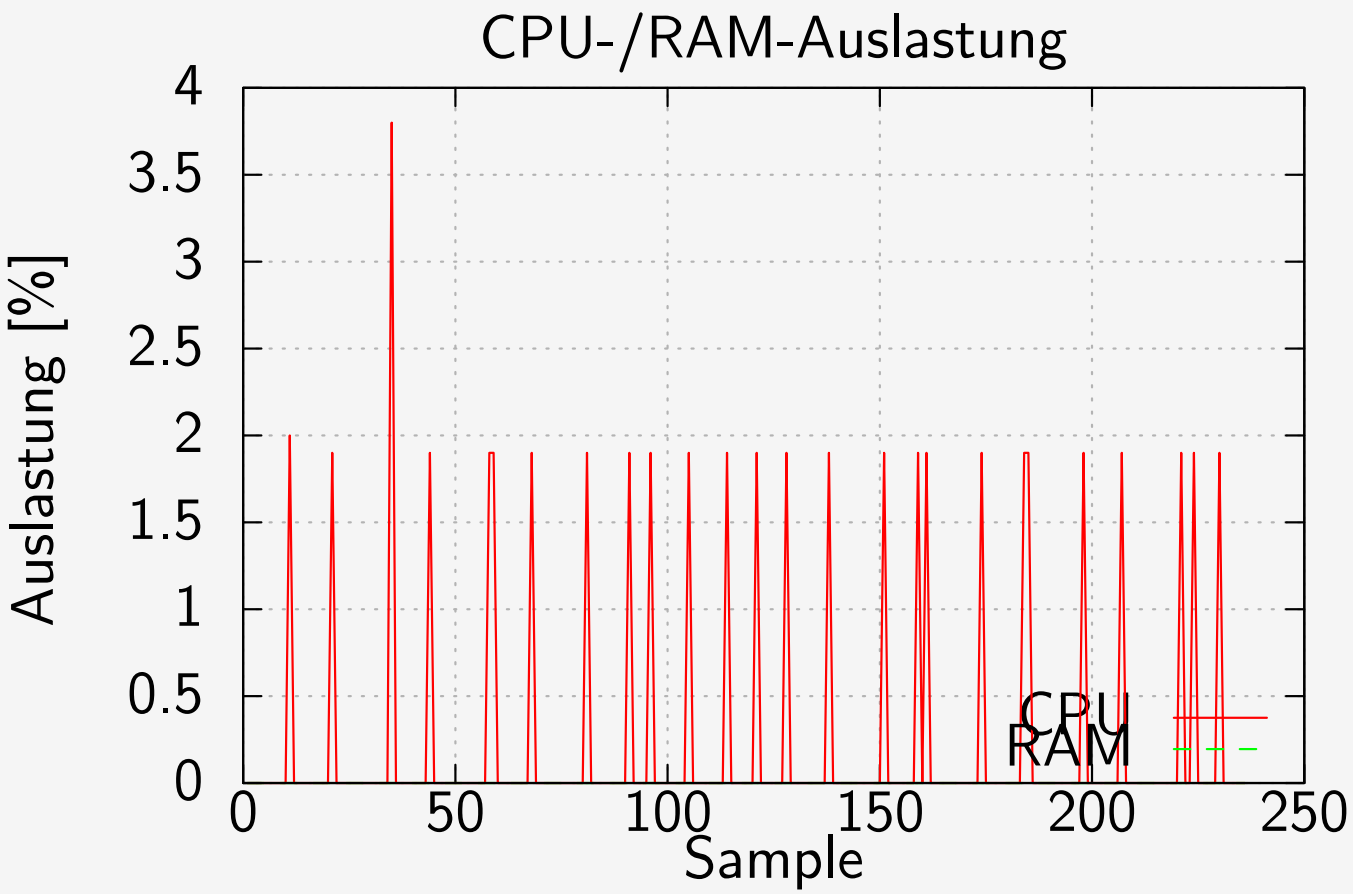


Abbildung: ublox mit PPS: CPU- und RAM-Auslastung

Aufwachzeiten Polling-Thread

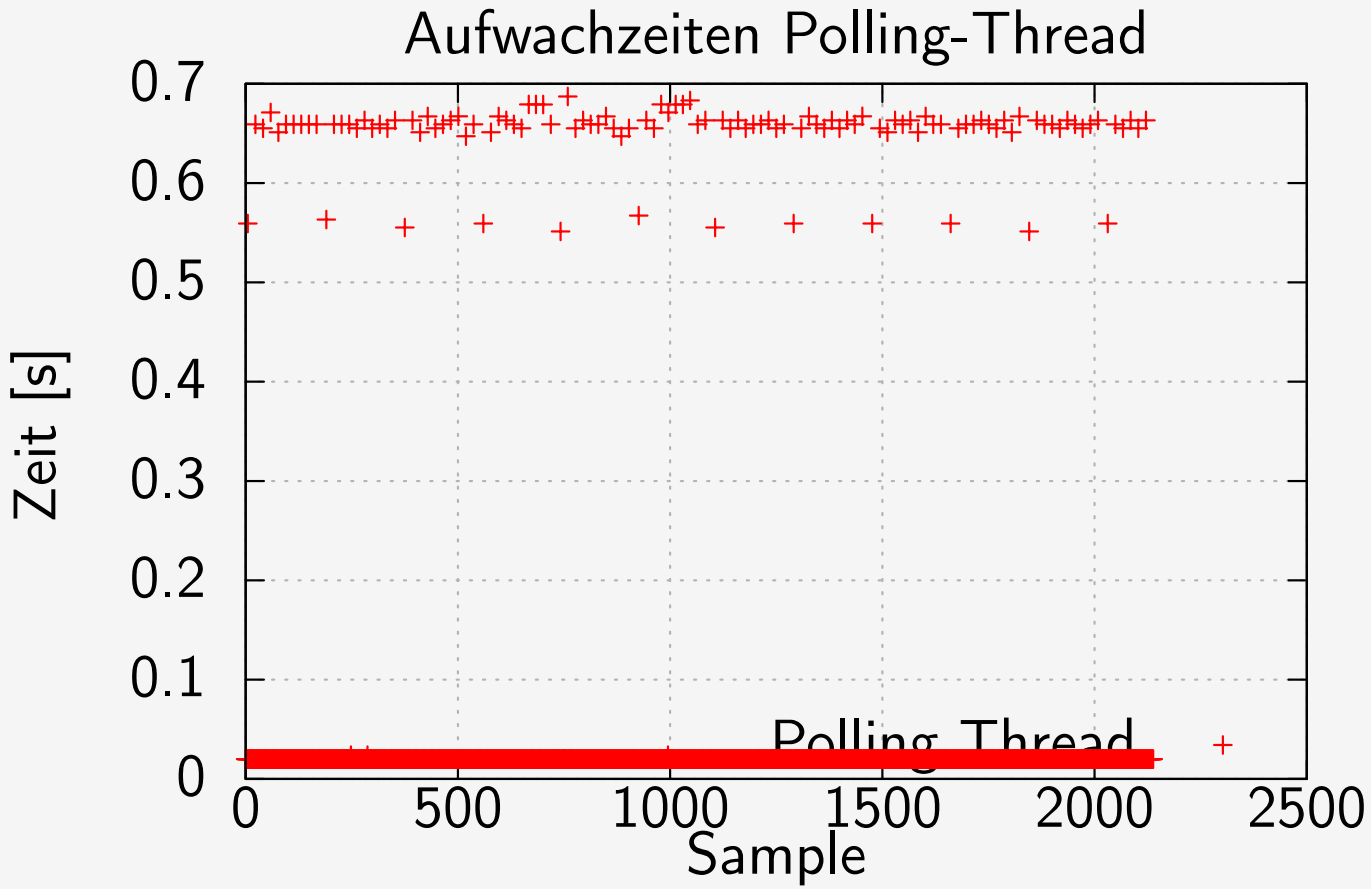


Abbildung: ublox: Differenz Aufwachzeiten Polling-Thread

Scheduling mit SCHED_FIFO und RT-Patch

Änderungen

- Scheduling pollender Thread SCHED_OTHER zu SCHED_FIFO und maximale Priorität.
- CONFIG_PREEMPT_RT-Patch

$\Rightarrow$ Garantie, dass Thread nicht unterbrochen wird.

Evaluierung I

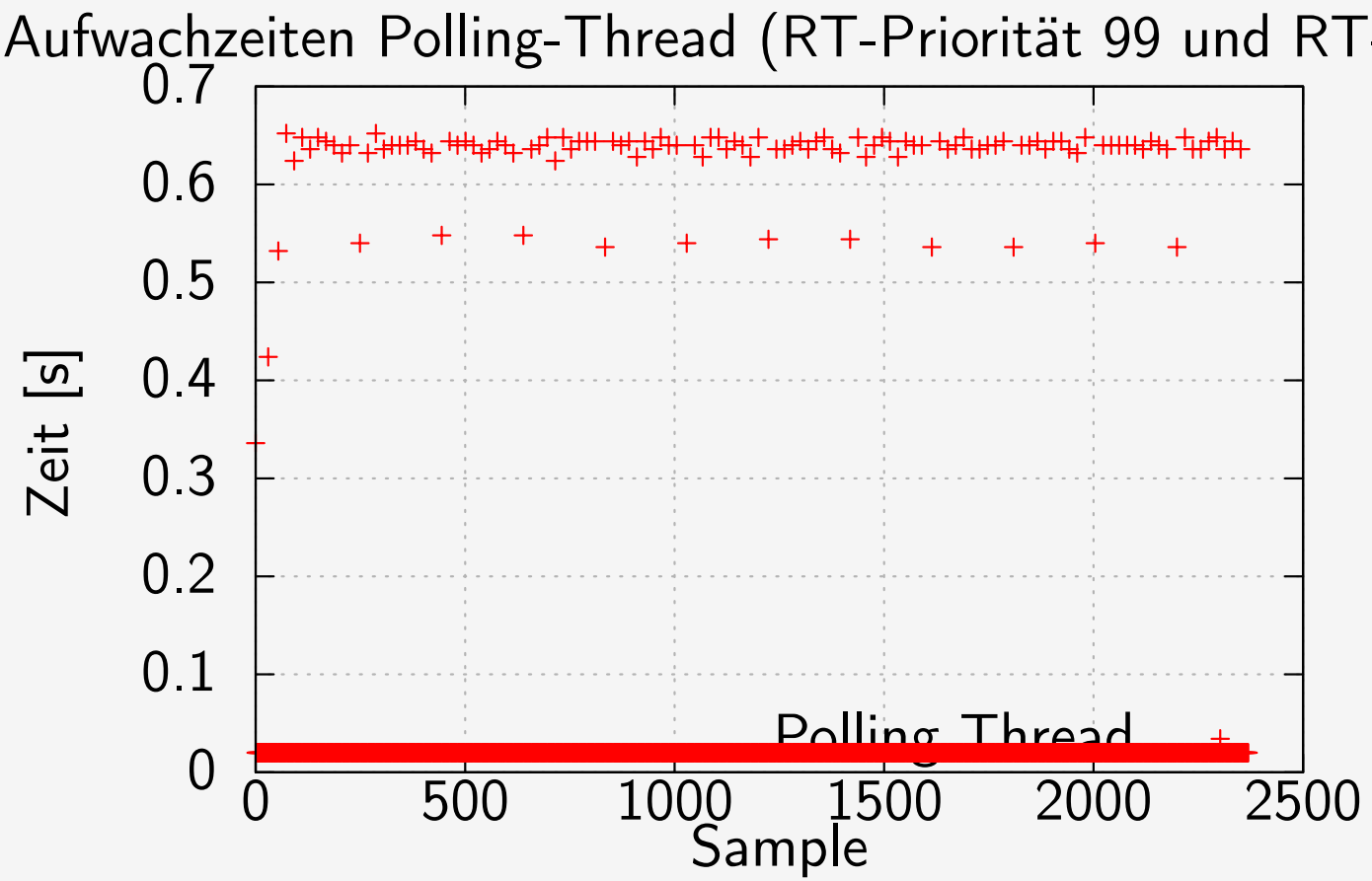


Abbildung: ublox: Differenz Aufwachzeiten Polling-Thread bei RT-Priorität 99

Evaluierung II

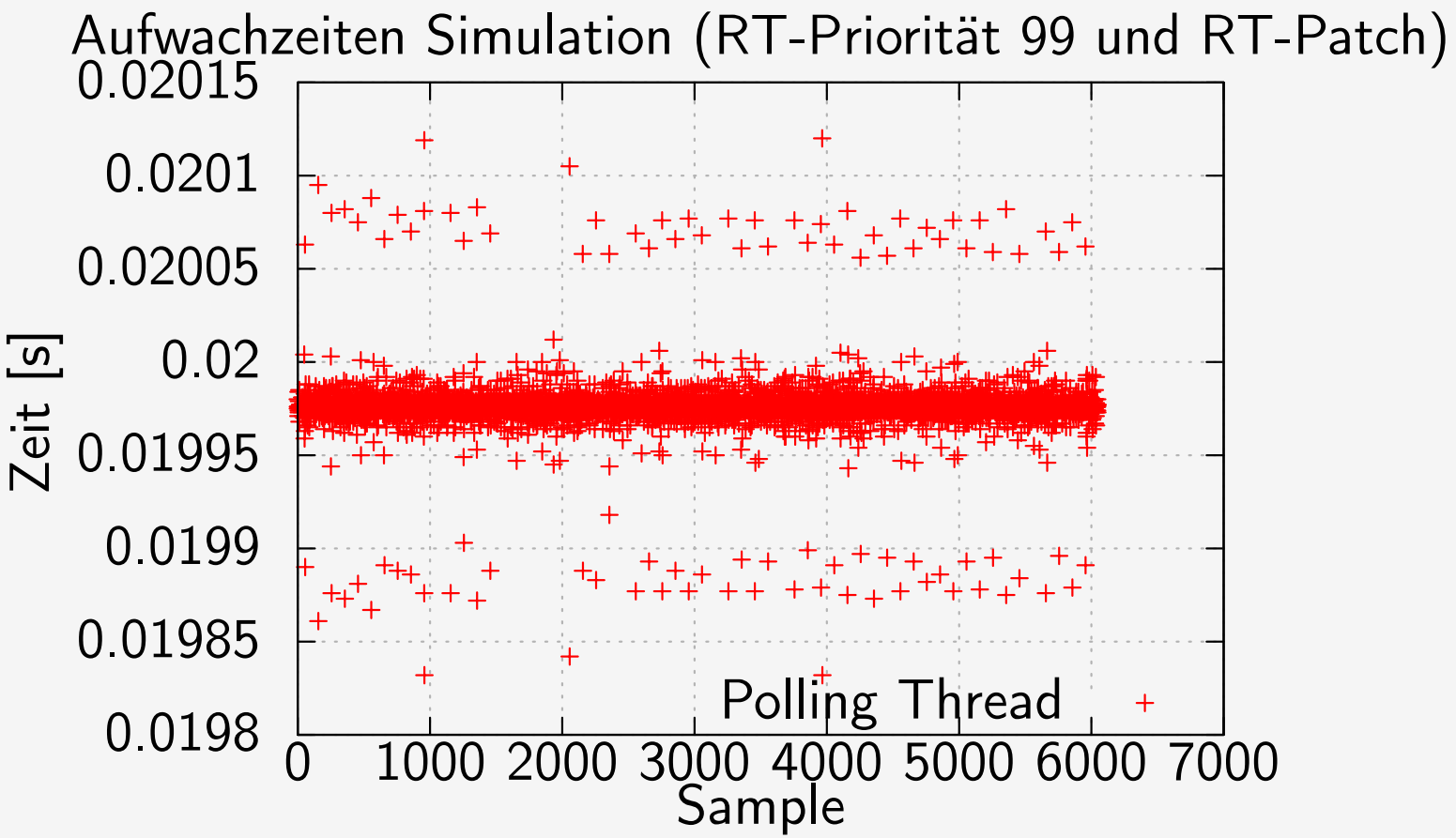


Abbildung: ublox: Differenz Aufwachzeiten Simulation bei RT-Priorität 99

Übersicht

Lessons Learned

- Besseres Verständnis für Betriebssystem.
- Linux ist auch „nur“ ein großes Framework.
  $\Rightarrow$ Nutze Open Source und schaue den Code an!
- (Eingebettete) Hardware funktioniert nie!
  $\Rightarrow$ Simulation notwendig.
- Testautomatisierung auf Kernel-Ebene vielleicht durch `Autotest`.

Danke für die Aufmerksamkeit.

Fragen und Diskussion.