HOCHSCHULE TRIER
Trier University of Applied Sciences
Informatik - Computer Science

# Multi-core-Kommunikationsmechanismus zwischen AUTOSAR und Linux

Multi-core communication mechanism between AUTOSAR and Linux

Sven Killig

Betreuer: Prof. Dr. Jörn Schneider

Schwabach, 18.6.2013

# Kurzfassung

Als Teil des Forschungsprojekts „econnect Germany“ wird an der Hochschule Trier ein Fahrzeugrechner für Elektroautos entwickelt. In diesem befindet sich eine dual-core-CPU, auf deren Kernen AUTOSAR und Linux gleichzeitig ausgeführt werden.

Zwischen diesen sollen Daten transferiert werden können. Die bereits in den Betriebssystemen eingebauten IPC-Kommunikationsmechanismen arbeiten nicht über Betriebssystemgrenzen hinweg. Deshalb wird ein Betriebssystem-übergreifender Kommunikationsmechanismus benötigt, dessen Entwicklung und Beschreibung Gegenstand dieser Arbeit ist.

# Inhaltsverzeichnis

**1 Grundlegende Begriffe und Konzepte ..... 1**
1.1 OS ..... 1
1.2 WCET ..... 1
1.3 RTOS ..... 1
1.4 AUTOSAR ..... 1
1.5 AUTOSAR-Tasks ..... 1
1.6 Arctic Core ..... 2
1.7 Arctic Studio ..... 2
1.8 CAN-Bus ..... 2
1.9 Dual core ..... 2
1.10 Linux Kernel-Modul ..... 2

**2 Einleitung ..... 3**
2.1 Forschungsprojekt „econnect Germany“ ..... 3
2.2 Gesamtkonzept Hub Trier ..... 3
2.3 Feldversuch zur Ermittlung der Nutzerakzeptanz ..... 4
2.4 Fahrzeugrechnersystem ..... 5

**3 Problemstellung ..... 6**
3.1 Remote Processor Messaging (RPMsg) ..... 7

**4 Aufgabenstellung und Zielsetzung ..... 9**

**5 Verwendete Hardware ..... 10**
5.1 ARM ..... 10
5.2 Pandaboard ..... 10
5.3 Schnittstellen ..... 11
5.4 OMAP4460 ..... 11
5.5 Probleme und Lösungen bei der Inbetriebnahme ..... 11
5.5.1 Dynamic clocking ..... 12
5.5.2 Kühlkörper ..... 12
5.5.3 SD Card ..... 12
5.5.4 NFS root ..... 12
5.5.5 HDMI ..... 13
5.5.6 DVI ..... 13
5.6 Netzteil ..... 13
5.7 Login über serielle Schnittstelle ..... 13
5.8 JTAG ..... 13
5.8.1 Bus Blaster ..... 13

5.8.2 OpenOCD ... 14
5.9 Bootvorgang ... 15

**6 Implementierung ... 17**
6.1 Algorithmus ... 17
6.2 producer-consumer.h ... 17
6.3 Producer ... 17
6.3.1 producer_init() ... 17
6.3.2 produce_message(int sequence_number, int command, char * payload) ... 18
6.4 producerTask() ... 18
6.5 Consumer ... 19
6.5.1 consumer.c ... 19
6.5.2 consumer_init() ... 19
6.5.3 intrpt_routine() ... 20
6.5.4 consumer() ... 20
6.5.5 write_file() ... 20
6.5.6 consumer_exit() ... 20

**7 Test ... 22**
7.1 AUTOSAR ... 22
7.2 Linux ... 22
7.2.1 Cross compiling ... 22
7.2.2 Start ... 23
7.2.3 kdb ... 24

**8 Ausblick ... 25**

**9 Fazit ... 26**

**Literatur ... 27**

**Index ... 28**

**Glossar ... 29**

**Anhang ... 30**

# Abbildungsverzeichnis

Abbildung 1 Gesamtkonzept Hub Trier. Quelle: [9] ..... 3
Abbildung 2 Feldversuch. Quelle: econnect-trier.de ..... 4
Abbildung 3 Systemarchitektur. Quelle: [10] ..... 5
Abbildung 4 Linux IPC: communication. Quelle: [12] ..... 7
Abbildung 5 Pandaboard (Originalgröße). Quelle: [16] ..... 10
Abbildung 6 Kühlkörper ..... 12
Abbildung 7 Bus Blaster. Quelle: dangerousprototypes.com ..... 14
Abbildung 8 ARM20TI14-Adapter. Quelle: tincantools.com ..... 14
Abbildung 9 Angepaßte Bootkette; nicht eingezeichnet der primäre ROM-Code. Quelle: [10] 15
Abbildung 10 Arctic Studio Debug Configurations ..... 22

# Tabellenverzeichnis

Tabelle 1 Memory map ..... 18
Tabelle 2 Aufbau des Puffers ..... 18
Tabelle 3 Aufbau einer message ..... 18

# 1 Grundlegende Begriffe und Konzepte

## 1.1 OS

Gemäß [1] ist ein Operating System/Betriebssystem „[…] there to manage all the pieces of a complex system. […] The job of the operating system is to provide for an orderly and controlled allocation of the processors, memories, and I/O devices among the various programs competing for them“. Es sorgt also für die Zuteilung von begrenzten Ressourcen.

Außerdem bildet es eine Abstraktionsschicht zwischen Anwenderprogrammen und Hardware.

## 1.2 WCET

Die Worst-case execution time ist die maximale Zeit, die ein Codeabschnitt auf einer bestimmten Hardware-Plattform zur Ausführung benötigt. Um sie automatisiert bestimmen zu können, sind Einschränkungen der in einem Programm erlaubten Operationen nötig. [2]

## 1.3 RTOS

Ein Realtime OS gibt bestimmte Garantien, was die Latenzen, also die zeitlichen Verzögerungen beim Reagieren auf Ereignisse angeht. Die hierfür notwendige WCET und formale Verifikation wird erst ermöglicht durch den geringen Umfang.

Abgegrenzt wird es zum Application OS (AOS)/Interactive OS wie z.B. Linux.

## 1.4 AUTOSAR

Die AUTomotive Open System ARchitecture [3] ist das Ergebnis einer Zusammenarbeit verschiedener Hersteller aus dem Automotive-Bereich. Sie definiert u.a. hardware-abhängige Basis- und –unabhängige Anwendungs-Software mit einem dazwischen liegenden virtuellen Funktions-Bus.

## 1.5 AUTOSAR-Tasks

Eine vom AUTOSAR-Scheduler aufrufbare Funktion. Es existieren sowohl Basic- als auch Extended-Tasks. Letztere können auf Ereignisse reagieren.

## 1.6 Arctic Core

Arctic Core stellt eine konkrete Open-Source- AUTOSAR 3.1-Implementierung von ArcCore AB aus Schweden dar.

## 1.7 Arctic Studio

Da AUTOSAR nicht nur Schnittstellen, sondern auch Strukturierungs- und Entwicklungsmethodiken definiert, hat ArcCore AB eine Eclipse-basierte IDE namens Arctic Studio veröffentlicht, um komfortabel im AUTOSAR-Umfeld entwickeln zu können.

## 1.8 CAN-Bus

Das Controller Area Network (CAN) ist ein vornehmlich im Automobil-Bereich eingesetztes Bussystem. Es wurde 1986 von der Robert Bosch GmbH entwickelt, dabei wurde besonders Wert auf die Echtzeit-Fähigkeit gelegt. Diese wird durch eine Arbitrierung mit hierarchischen Prioritäten ermöglicht.

## 1.9 Dual core

Da die Erhöhung der Taktfrequenz von CPUs an physikalische Grenzen stößt, geht der Trend hin zur Fertigung von CPUs mit mehreren identischen Kernen. [4]

Das Forschungsprojekt ProSyMig an der Hochschule Trier beschäftigt sich mit der Parallelisierung bestehender Software im Automotive-Bereich. [5]

## 1.10 Linux Kernel-Modul

Eigenständige binaries, die zur Laufzeit in den Kernel ge- und entladen werden können. Alle Peripherie-Treiber sind beispielsweise Module.

# 2 Einleitung

## 2.1 Forschungsprojekt „econnect Germany“

Im Rahmen des zweieinhalb Jahre dauernden Leuchtturmprojekts „econnect Germany“ haben sich sieben Stadtwerke und Forschungs- und Entwicklungspartner zusammengeschlossen, um elektromobile Verkehrsanwendungen (Smart Traffic) und die Integration der Elektromobilität in das intelligente Stromnetz der Zukunft (Smart Grid) mittels IKT zu erforschen. [6]

Das erfolgt dezentral in sogenannten „Hubs“. Der Hub Trier [7] wird gebildet durch die Stadtwerke Trier, die ABB AG sowie dem Forschungsverbund Verkehrstechnik und Verkehrssicherheit (FVV) [8]. Dieser besteht aus Mitgliedern der Fachbereiche Informatik, Maschinenbau und Elektrotechnik der Hochschule und der Abteilung allgemeine Psychologie und Methodenlehre der Universität Trier.

## 2.2 Gesamtkonzept Hub Trier

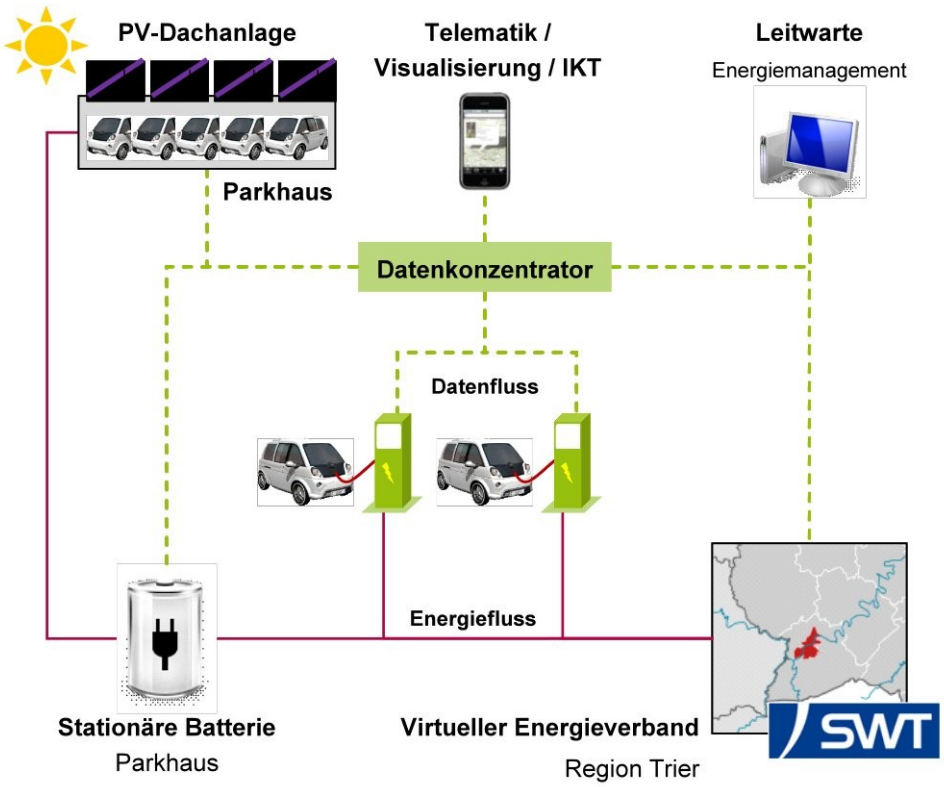


**Abbildung 1 Gesamtkonzept Hub Trier. Quelle: [9]**

Elektroautos bieten die Möglichkeit umweltfreundlicher Mobilität, allerdings nur, wenn der Strom aus erneuerbaren Energien gespeist wird; wird der Strom in Kohle- oder Ölkraftwerken produziert, wird die Umweltbelastung nur verlagert. Erneuerbare Energien stehen aber nicht kontinuierlich zu Verfügung, sondern sind z.B. vom Wetter abhängig. Außerdem sollten sie wegen dem problematischen Ausbau der Stromtrassen regional erzeugt werden. Aus diesen

Gründen wird ein Zusammenspiel von intelligenten Netzen und Elektro-Autos benötigt. Dabei können Elektroautos einerseits den Strom aufnehmen, wenn gerade eine Überkapazität zu Verfügung steht, anderseits auch wieder abgeben, wenn der Bedarf das Angebot übersteigt. Da hierfür die Nutzer auf einen jederzeit vollgeladenen Akku verzichten müssen, wird erstmals eine empirische Untersuchung zur Akzeptanz durchgeführt.

## 2.3 Feldversuch zur Ermittlung der Nutzerakzeptanz

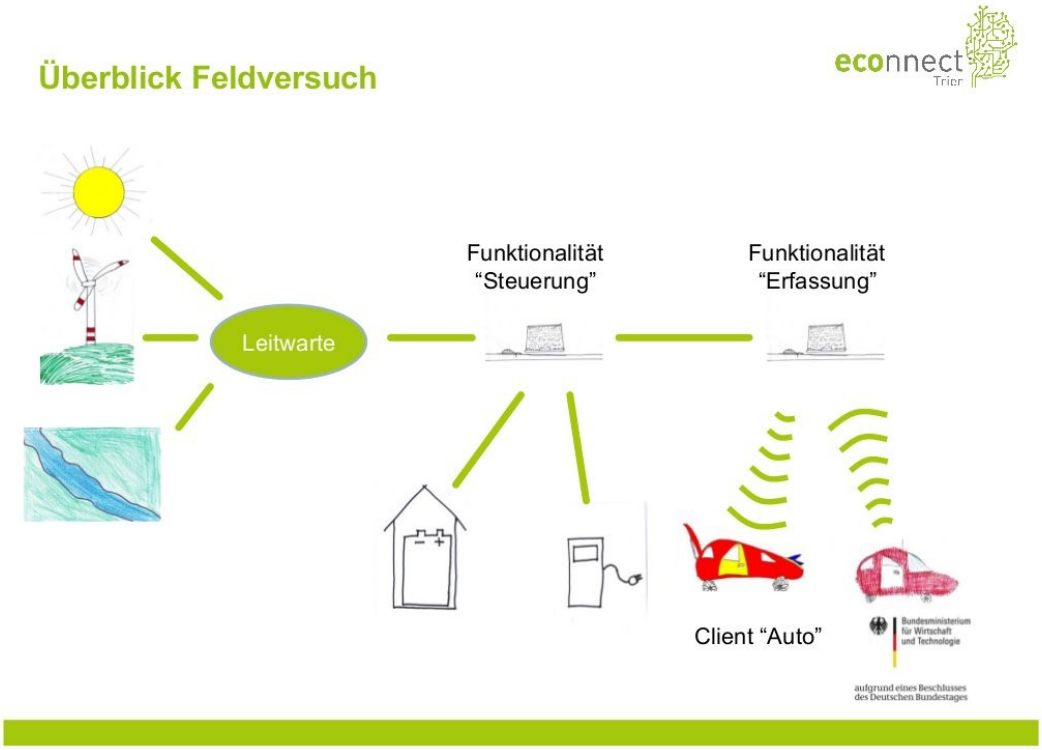


**Abbildung 2 Feldversuch. Quelle: econnect-trier.de**

80 Probanden werden innerhalb eines Jahres fünf Serien-Elektrofahrzeuge zu Verfügung gestellt. Diese werden mit einem an der Hochschule Trier entwickelten Fahrzeugrechner ausgestattet, der von der Energieleitzentrale der Stadtwerke Trier jederzeit den aktuellen Strompreis abfragen kann. Dieser wird kontinuierlich der Situation bei Stromerzeugung und Stromverbrauch angepaßt.

Über ein Touch-Display wird der Nutzer beispielsweise angeben können, daß am nächsten Tag um 8:00 eine Reichweite von 70 km und für spontane Fahrten Strom für 20 km erforderlich ist. Das bietet gleich drei Vorteile:

- Der Nutzer kann automatisiert zum günstigsten Tarif laden und evtl. vorhandene Überkapazitäten lohnend zurückspeisen.
- Die Stadtwerke können den Stromverbrauch über den Strompreis zeitlich steuern und den Einkauf von Fernstrom vermeiden.
- Die Umwelt wird durch den Einsatz regenerativer Energien geschont.

Um den zukünftigen Anteil der Benutzer, der sich an einem solchen System beteiligen wird, zu bestimmen, werden Daten und Rückmeldungen der Probanden von den Psychologen der Universität ausgewertet.

## 2.4 Fahrzeugrechnersystem

In der Teilvorhabensbeschreibung der Hochschule Trier (TVB) [9] wird die Realisierung des Fahrzeugrechners unter dem Arbeitspaket (AP) 2.5 - IKT zur Einbindung Fahrzeugbatterien" als "Realisierung Fahrzeugrechnersystem" und der Feldversuch unter AP 3.2 aufgeführt.

Zum Einsatz kommt ein dual core-System, auf dessen Kernen zwei verschiedene OS laufen. Das folgende Diagramm stellt die Systemarchitektur dar:

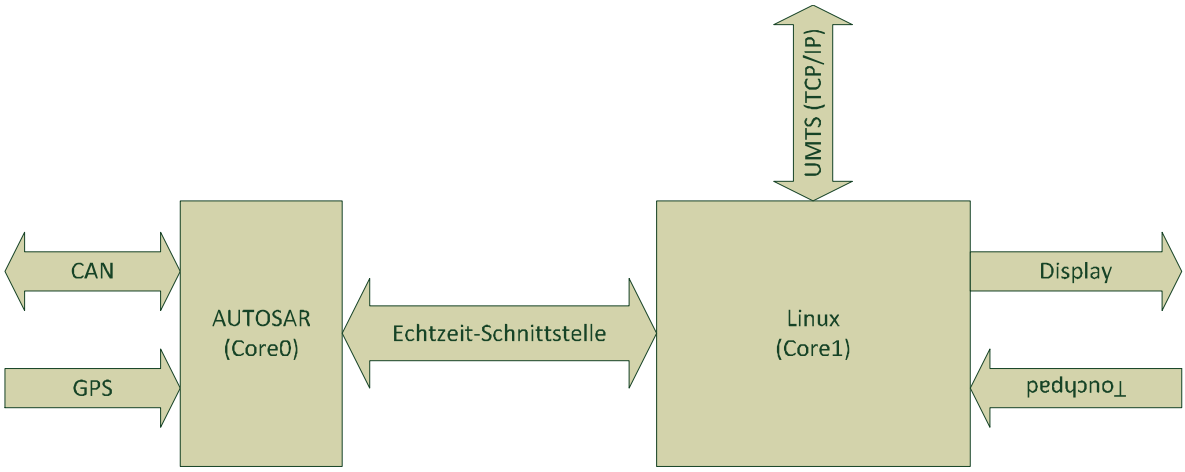


**Abbildung 3 Systemarchitektur. Quelle: [10]**

Gegenstand dieser Arbeit ist die Echtzeit-Schnittstelle zwischen den Cores.

Zunächst gehe ich in Kapitel 3 auf die Problemstellung ein.
In Kapitel 4 beschreibe ich die Aufgabenstellung und Zielsetzung.
Kapitel 5 behandelt dann die verwendete Hardware und Kapitel 6 schließlich die Implementierung.
In Kapitel 7 beschreibe ich, wie man diese ausführt.
In Kapitel 8 gebe ich einen Ausblick auf mögliche Verbesserungen.
Ich Kapitel 9 ziehe ich schließlich ein Fazit.

# 3 Problemstellung

Zur Aufteilung der Kerne zwischen Betriebssystem(en) gibt es unter anderem [11] folgende Möglichkeiten, die normalerweise auf dem Pandaboard gleichzeitig aktiv sind:

- SMP - Symmetric Multiprocessing
  Im normalen Betrieb läuft Linux auf den beiden Cortex A9-Kernen und verteilt die Prozesse und Threads des Betriebssystems und der Anwenderprogramme auf diesen.
- AMP - Asymmetric Multiprocessing
  Die Cortex M3-Kerne stehen nicht unter direkter Kontrolle von Linux, sondern dienen der Verwaltung der Multimedia-Beschleunigungs-Einheiten. Auf diesen läuft eine Firmware, die von Linux über RPMsg (s.u.) angesprochen wird.

Im Rahmen von econnect wird SMP durch Ausweitung von AMP auf die Cortex A9-Kerne ersetzt: Auf einem läuft AUTOSAR, um sich möglichst schnell nach dem Einschalten an der Fahrzeug-Kommunikation auf dem CAN-Bus beteiligen zu können. Auf dem anderen Linux, um die dafür vorhandenen Anwendungen und Peripherie-Treiber nutzen zu können und es für die Benutzerschnittstelle zu verwenden.

In einer anderen Arbeit wurden für AUTOSAR bereits Tasks für das Protokollieren von CAN- und GPS-Daten entwickelt. Diese Daten könnten auf der unter Linux gemounteten SD Card persistent abgelegt werden. Dazu ist ein Kommunikationsmechanismus notwendig.

Im Linux-Kernel sind bereits zahlreiche Interprozess-Kommunikations-Mechanismen (IPC) enthalten:

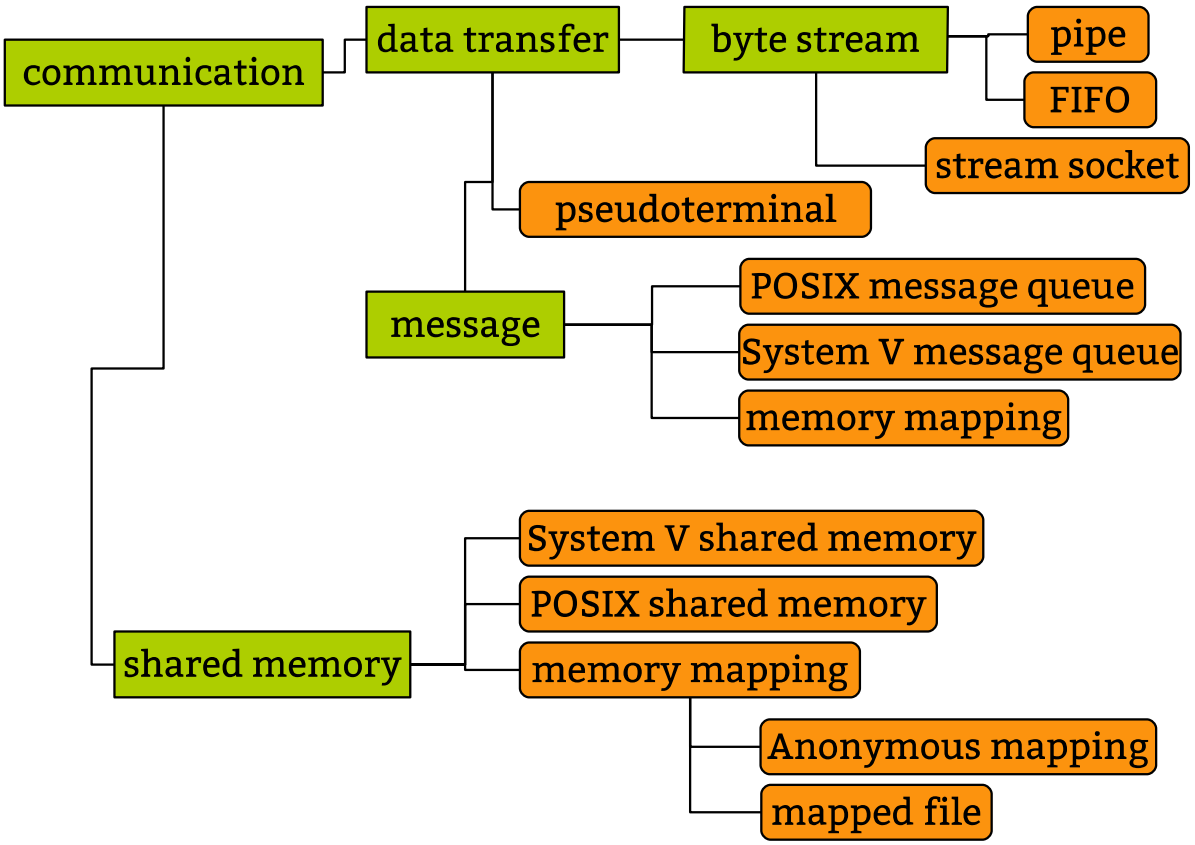


**Abbildung 4 Linux IPC: communication. Quelle: [12]**

Ende des Jahres soll noch kdbus, ein Nachrichten-basierter Mechanismus, der zuletzt auf dem Automotive Linux Summit vorgestellt wurde [12], hinzukommen.
All diesen Mechanismen ist jedoch gemein, daß für eine Kommunikation zwischen Betriebssystemen, die auf verschiedenen Kernen in einem Multiprozessor-System laufen auf sie nicht zurückgegriffen werden kann.

## 3.1 Remote Processor Messaging (RPMsg)

RPMsg [13] ist ein Inter-Prozessor-Kommunikations-Framework, das im OMAP4460 in Form des Mailbox-Subsystems implementiert ist. Dieses Subsystem kann für eine hardwareunterstütze Kommunikation genutzt werden, allerdings normalerweise nur zwischen den Cortex A9- und M3-Subsystemen, nicht innerhalb dieser. Wenn AUTOSAR auf einem der M3-Kerne laufen würde, wie in [14] vorgeschlagen, könnte mittels RPMsg sehr effizient kommuniziert werden. Da das ein spezifisches Merkmal von TI OMAP-SoCs ist, wäre die Portierbarkeit auf SoCs anderer Hersteller allerdings einschränkt.
Auf dem Pandaboard ist es normalerweise in Verwendung, wie man erkennen kann, wenn man sich mit `lsmod` die geladenen Kernel-Module anzeigen läßt:

```
Module                  Size  Used by
rpmsg_omx               9295  0
omaprpc                18485  0
omap_remoteproc         5503  4294967295
rpmsg_resmgr_common     3281  0
remoteproc             25653  3
rpmsg_omx,omap_remoteproc,rpmsg_resmgr_common
```

```
omap_rpmsg_resmgr       9585  0
rpmsg_resmgr            7049  2 rpmsg_resmgr_common,omap_rpmsg_resmgr
virtio_rpmsg_bus       11932  4 omapdce,rpmsg_omx,omaprpc,rpmsg_resmgr
virtio                  5489  2 remoteproc,virtio_rpmsg_bus
virtio_ring             8364  2 remoteproc,virtio_rpmsg_bus
```

Eine der Änderungen, die die Hochschule vorgenommen hat, ist die Deaktivierung von rpmsg und damit die Unterstützung der Multimedia-Beschleunigung durch die beiden M3-Kerne. Dadurch wären diese zwar frei für AUTOSAR, allerdings scheinen sie schlecht dokumentiert zu sein [15].

# 4 Aufgabenstellung und Zielsetzung

Durch die Senderichtung der protokollierten Daten von AUTOSAR nach Linux ergibt sich als Rahmenbedingung, daß auch für den Kommunikationsmechanismus vorerst diese Richtung genügt.
Insgesamt bietet es sich an, den Datenaustausch mit Hilfe des producer/consumer-Musters [1] zu implementieren. Dabei besetzt AUTOSAR die Rolle des producers, der einen Puffer mit Nachrichten befüllt, ein Linux-Kernel-Modul die Rolle des consumers, der sie entnimmt. Bei je nur einem producer und consumer existiert eine elegante Implementierung.

Um neue Nachrichten mitzubekommen gibt es folgende Möglichkeiten:

- Inter-Prozessor-Interrupts (IPI)
  IPIs sind Interrupts, die zwischen Prozessorkernen signalisiert werden. Hierüber kann eine effizientere Kommunikation ermöglicht werden, da nur dann Code ausgeführt wird, wenn tatsächlich einen neue Nachricht vorliegt; allerdings zum Preis einer höheren Komplexität bei der Einrichtung der Interrupt-Quellen und -Ziele und der Interrupt-Behandlung.
- Polling
  Beim Polling wird periodisch überprüft, ob neue Nachrichten vorliegen. Das bedingt zwar einen overhead durch vergebliche Überprüfungen, wenn keine neuen Nachrichten eingetroffen sind, dafür ist die Implementierung nicht an die Fähigkeiten bestimmter Hardware gebunden. Außerdem kann es zu keinen Störungen des Echtzeitbetriebs unter AUTOSAR kommen. Ich habe mich deshalb bei meiner Implementierung für diese Variante entschieden.

# 5 Verwendete Hardware

## 5.1 ARM

Während die x86-Architektur vornehmlich auf hohe Leistung hin optimiert ist, steht bei der schon häufig mobil eingesetzten ARM-Architektur die Energieeffizienz im Vordergrund, was sie für einen Fahrzeugrechner prädestiniert.
Außerdem ist die Integrationsdichte sehr hoch: ARM-CPUs werden als System on Chip (SoC) verbaut, die gleich häufig genutzte Peripherie beinhalten.
Sie bietet im Gegensatz zu klassischen Microcontrollern auch genug Leistungsfähigkeit, um auch anspruchsvollere OS wie Linux ausführen zu können.

## 5.2 Pandaboard

Das in dieser Arbeit verwendete Pandaboard ist ein evaluation board für das OMAP4460 ARM SoC von Texas Instruments mit 1 GB dual channel LPDDR2 RAM als Package on Package (PoP) [16].

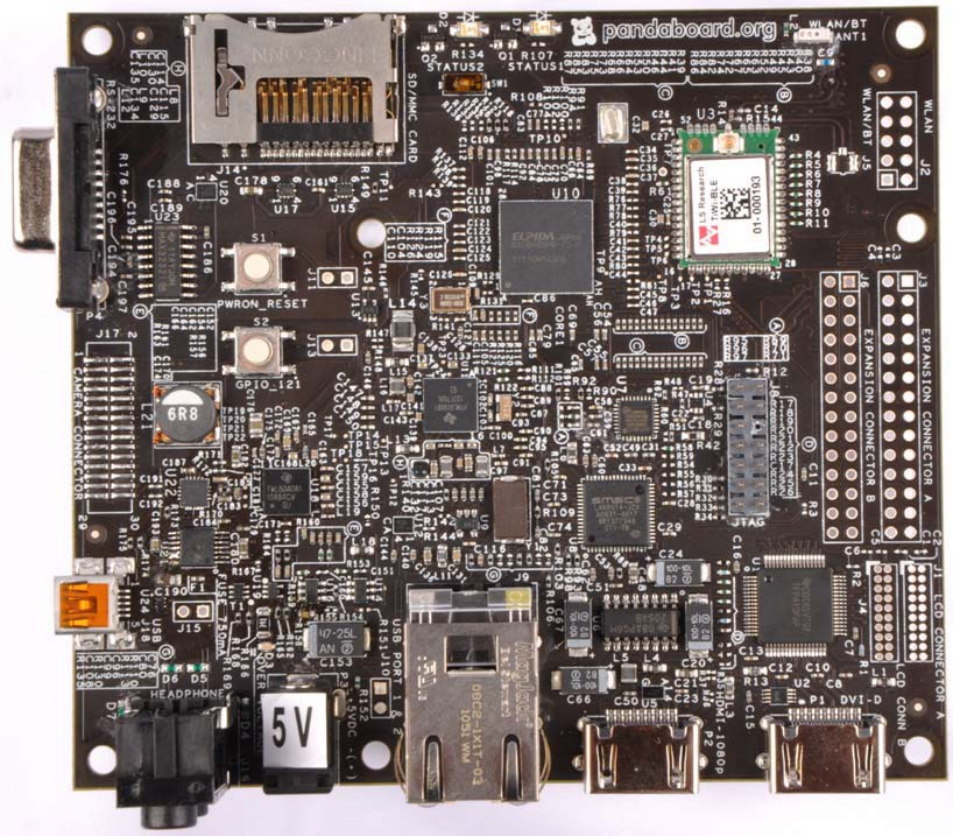


**Abbildung 5 Pandaboard (Originalgröße). Quelle: [16]**

## 5.3 Schnittstellen

- HDMI
- DVI via HDMI
- Ethernet 100 MBit/s
- WLAN 802.11n
- Bluetooth
- 1 USB host
- 1 USB host/device
- Kopfhörer Stereo
- Line in Stereo
- Serielle Schnittstelle
- SD Card
- 2 Tasten
- 2 LEDs
- JTAG TAP 14-polig

Unbestückt:

- GPIO
- SPI (genutzt für den Anschluß eines CAN-Controllers)
- Weitere serielle Schnittstelle (genutzt für den Anschluß eines GPS-Empfängers)
- 2 USB host
- Kamera
- Parallel/DSI LCD

## 5.4 OMAP4460

Der OMAP4460 ARM SoC ist in 45 nm gefertigt. Er beinhaltet folgende Kerne:

- 2 Cortex A9-Kerne mit 1,2 GHz Taktfrequenz
- 2 Cortex M3-Kerne
- Integrierte GPU

Daneben sind noch verschiedene Multimedia-Hardwarebeschleunigungs-Subsysteme vorhanden. Das ist auch nötig, da die A9-Kerne bis zu 40 mal langsamer bei Fließkomma-lastigem Java[1] und 6 mal langsamer bei Integer-lastigem C[2] als herkömmliche x86-CPUs sind, wie ich bei Tests von Projekten von mir gemessen habe.

## 5.5 Probleme und Lösungen bei der Inbetriebnahme

Nachfolgend einige Probleme, die bei mir während der Inbetriebnahme aufgetreten sind, und Vergleiche mit dem ebenfalls ARM-basierten QuickStartBoard von Freescale, das in meiner Projektarbeit[3] verwendet wurde.

[1] http://sven.killig.de/OpenCL
[2] http://sven.killig.de/SDR
[3] http://sven.killig.de/meego/Projektarbeit.pdf

### 5.5.1 Dynamic clocking

Bei längerem Betrieb ohne Last der im Oktober 2012 erhältlichen offiziellen Ubuntu-Image-Datei ist das System eingefroren. Abhilfe schaffte ein Deaktivieren der dynamischen Taktumschaltung:

```
# echo performance >
/sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
# echo performance >
/sys/devices/system/cpu/cpu1/cpufreq/scaling_governor
```

(Eingaben auf der Konsole werden auch im weiteren Verlauf mit # gekennzeichnet.)

### 5.5.2 Kühlkörper

Da der OMAP4460 bis zu 2,6 W Verlustwärme erzeugen kann [17], als PoP unter dem RAM verbaut ist und nicht sichergestellt ist, daß jederzeit die thermal management-Treiber aktiv sind, habe ich zur Sicherheit einen Kühlkörper auf ihm befestigt:

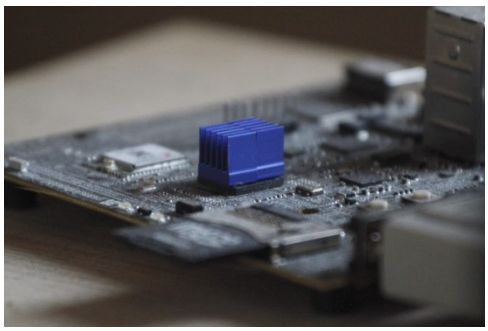

**Abbildung 6 Kühlkörper**

Die aktuelle Temperatur kann wie folgt ausgelesen werden:

```
# cat /sys/bus/platform/devices/temp_sensor_hwmon.0/temp1_input
39800
```

### 5.5.3 SD Card

Der Einsatz von herkömmlichen SD Cards als root filesystem ist wegen deren Optimierung auf lineares Schreiben und das FAT32-Dateisystem heikel. Freescale mußte beispielsweise bereits wegen geringer Performance dazu raten, die zum QSB mitgelieferte SD Card gegen eine andere auszutauschen. Besonders das Schreiben kleiner Blöcke kann zu einem write amplification-Faktor von bis zu 1024 führen [18].
Bei Einträgen wie den folgenden im syslog sollte auf eine Speicherkarte eines anderen Herstellers gewechselt werden:

```
[ 6028.380065] Buffer I/O error on device mmcblk0p2, logical block
2106480
[ 6246.707031] Aborting journal on device mmcblk0p2-8.
[ 6246.712463] EXT4-fs (mmcblk0p2): ext4_da_writepages: jbd2_start: 1023
pages, ino 333486; err -30
[ 6259.947723] EXT4-fs error (device mmcblk0p2):
ext4_journal_start_sb:327: Detected aborted journal
[ 6259.947723] EXT4-fs (mmcblk0p2): Remounting filesystem read-only
```

### 5.5.4 NFS root

Im Hinblick auf meine Probleme mit einer meiner SD Cards habe ich das mounten des root filesystems über NFS, daß ich so bereits beim QSB eingesetzt habe, wiederholen wollen. Allerdings gelang es mir nicht, den kompletten Ubuntu-Bootvorgang zum Laufen zu bringen.

Da mir auch von der Hochschule Probleme mit diesem init script-Sytem geschildert wurden, habe ich diese Versuche abgebrochen.

### 5.5.5 HDMI

Im Gegensatz zum QSB scheint es nicht möglich zu sein, beliebige Auflösungen wie z.B. 1280×800 zu konfigurieren. Das ist problematisch beim Einsatz von Monitoren ohne eigenen scaler.

### 5.5.6 DVI

Der Betrieb eines DVI-Monitors über die Buchse P1 wird unter Ubuntu 12.04 nicht unterstützt. Mit openfb statt opendrm scheint das möglich zu sein[4].

## 5.6 Netzteil

Das Pandboard läuft mit 5 V Spannung. [16] geht nicht auf die nötige Stromstärke ein; wenn man USB-Peripherie an einem self powered hub betreibt, genügt ein 2 A-Netzteil, wie es z.B. beim QSB mitgeliefert wird.

## 5.7 Login über serielle Schnittstelle

Um sich über die serielle Schnittstelle einloggen zu können, ist `/etc/init/ttyO2.conf` mit folgendem Inhalt anzulegen, um auf der Schnittstelle `ttyO2` einen login daemon zu starten:

```
start on stopped rc RUNLEVEL=[2345]
stop on runlevel [!2345]
respawn
exec /sbin/getty -L 115200 ttyO2
```

## 5.8 JTAG

Um den Arctic Core-Kernel komfortabel installieren und ausführen zu können, bietet sich JTAG an. Unter diesem Begriff, der eigentlich die Joint Test Action Group kennzeichnet, die die Test Access Port and Boundary-Scan Architecture entwickelt hat [19], versteht man eine standardisierte Schnittstelle für das Hardware-debugging.

### 5.8.1 Bus Blaster

Normalerweise wird von der Hochschule Trier zum Debuggen eine Lauterbach probe verliehen, da diese recht kostspielig ist. Der Bus Blaster von Dangerous Prototypes ist mit 30 € ein sehr günstiger OpenOCD-kompatibler Open-Hardware-JTAG-Debugger.

Eine Alternative stellt der Flyswatter(2) von Tin Can Tools dar. Dieser ist zwar zwei- bis dreimal so teuer, bietet aber auch gleich die notwendigen level shifter für die serielle Schnittstelle des Pandaboards.

[4] http://www.chalk-elec.com/?p=1478

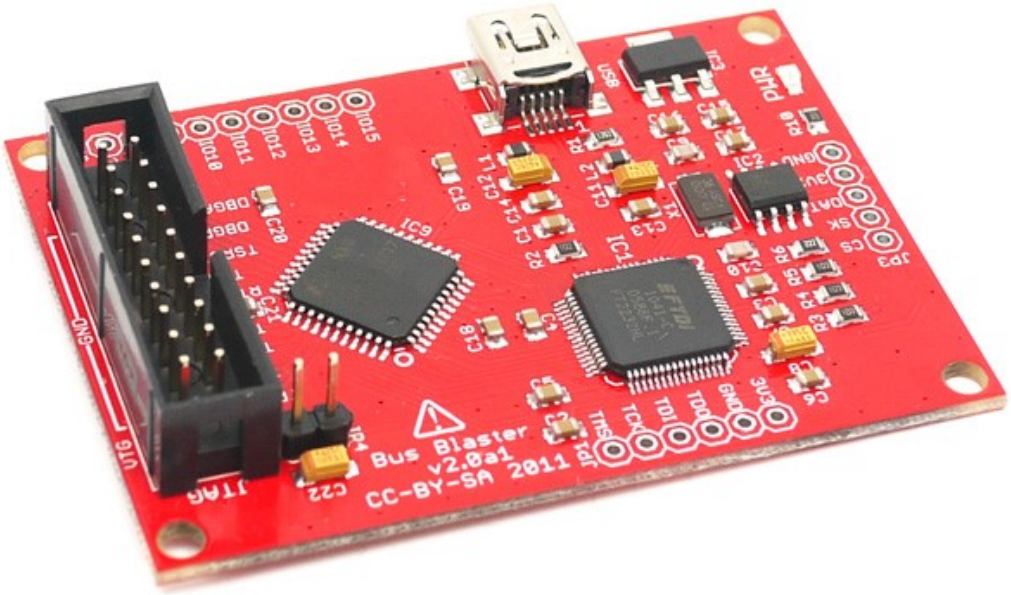


**Abbildung 7 Bus Blaster. Quelle: dangerousprototypes.com**

Eine zuverlässige Verbindung zwischen diesem und dem Pandaboard ist mit einem 20/14-poligen Adapter möglich:

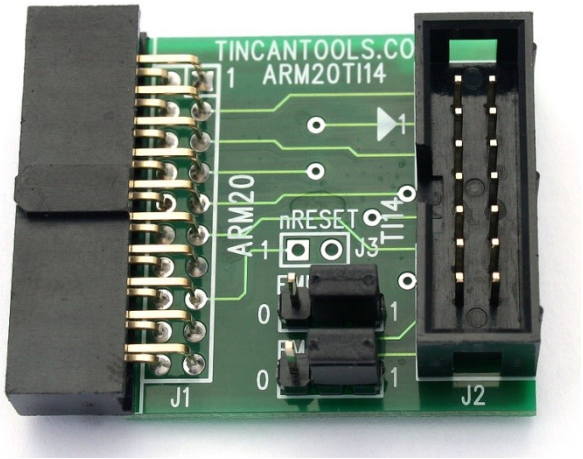


**Abbildung 8 ARM20TI14-Adapter. Quelle: tincantools.com**

### 5.8.2 OpenOCD

Die Software Open On-Chip Debugger wurde im Rahmen einer anderen Arbeit entwickelt [20] und stellt das Bindeglied zwischen gdb und dem JTAG-Debugger dar.

Für Windows sind Binaries erhältlich[5].

Start:

```
openocd-0.7.0\bin-x64\openocd-x64-0.7.0.exe -f interface\busblaster.cfg
-f board\ti_pandaboard_es.cfg
```

In der Version 0.7.0 kommt es zu folgender Fehlermeldung:

```
Runtime Error: embedded:startup.tcl:20: Unknown target type cortex_a,
try one of arm7tdmi, arm9tdmi, arm920t, arm720t, arm966e, arm946e,
arm926ejs, fa526, feroceon, dragonite, xscale, cortex_m, cortex_a8,
cortex_r4, arm11, mips_m4k, avr, dsp563xx, dsp5680xx, testee,
avr32_ap7k,  or hla_target
```

Zur Behebung ist in

```
openocd-0.7.0\scripts\target\omap4460.cfg
```

in der Zeile

```
target create $_TARGETNAME cortex_a -chain-position $_CHIPNAME.dap \
```

`cortex_a` durch `cortex_a8` zu ersetzen.

Vor dem ersten Starten einer Debug-Sitzung ist eine Telnet-Verbindung zu `localhost:4444` aufzubauen und das Kommando `halt` abzusetzen, da OpenOCD sonst abstürzt.

Somit ergibt sich folgende Kette:
Arctic Studio/eclipse CDT (DSF) – gdb – OpenOCD – MiniUSB-Kabel – Bus Blaster – ARM20TI14-Adapter – 14-poliges Flachbandkabel Pandaboard

## 5.9 Bootvorgang

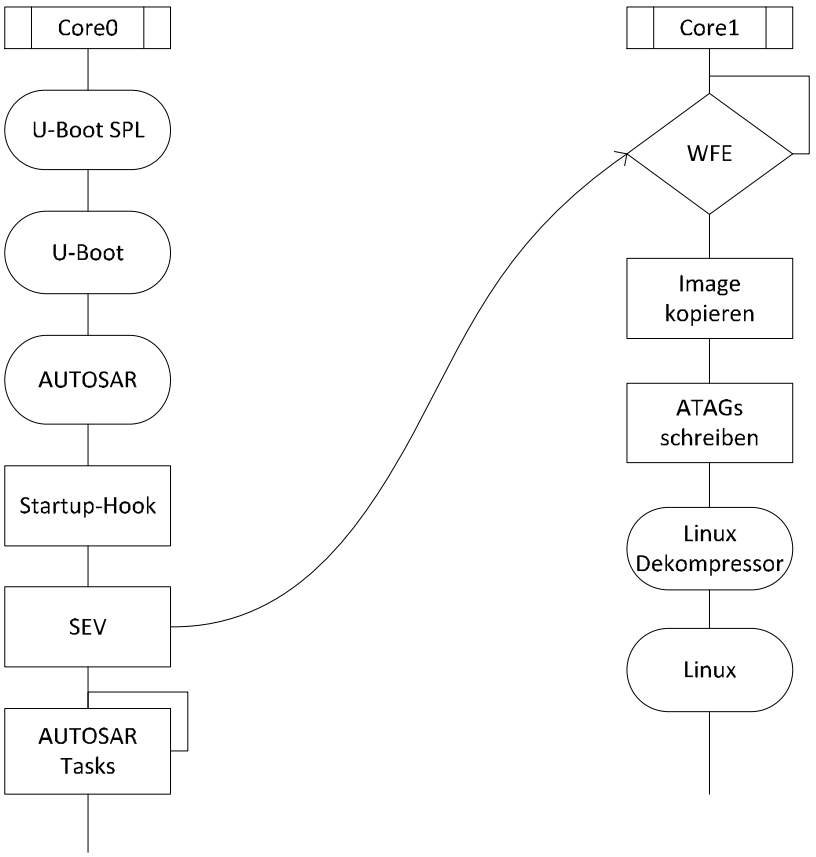


**Abbildung 9 Angepaßte Bootkette; nicht eingezeichnet der primäre ROM-Code. Quelle: [10]**

[5] http://www.freddiechopin.info/en/download/category/4-openocd

Im Gegensatz zum QSB ist auf dem Pandaboard kein Flash-Speicher verbaut, in dem der Bootloader uboot oder dessen Parameter gespeichert werden könnten. Deshalb liest der primäre ROM-Code uboot und ein Script aus der Datei `boot.scr` von der ersten Partition der SD Card.

Die Ausführungen zu AUTOSAR-Dateien beziehen sich im folgenden auf das Arctic Studio-Projekt „Econnect“ im branch „producer-consumer“ im AUTOSAR-git-repository der Hochschule.

Zum Start von AUTOSAR wird die Fähigkeit von uboot verwendet, ELF-binaries zu laden und auszuführen. In dem hier verwendeten befinden sich sowohl AUTOSAR als auch Linux. AUTOSAR sorgt dabei für den Start von Linux auf Core 1.

Dem Linux-Kernel können ähnlich wie Kommandozeilenprogrammen beim Start Parameter übergeben werden. Auf ARM-Systemen werden diese über ein Tag einer so genannten ATAGs-Struktur an den Linux-Kernel übergeben. Das ist eine Tag-Length-Value (TLV)-Struktur, die beim Start des Kernels an einer festen Adresse stehen muß. Die gesamte Struktur wird in AUTOSAR in der Datei `src/core1/atags.c` angelegt. Dort findet sich auch die Funktion `setup_commandline_tag()`, mit der die Kernelparameter erzeugt und übergeben werden.

An dieser Stelle wird dann auch die Adresse des gemeinsam genutzten Kommunikations-Puffers in den Parametern `consume.buffer_start` und `consume.buffer_size` übergeben.

Zusätzliche Kernel-Parameter sind in `src/core1/board_init.c` in der Zeile

```
#define BOOTARGS                        "elevator=noop vram=40M
root=/dev/mmcblk0p2 rootfstype=ext4 rw rootwait maxcpus=1 fixrtc quiet
splash"
```

definiert. Ich habe das in

```
#define BOOTARGS                        "elevator=noop vram=40M
root=/dev/mmcblk0p2 rootfstype=ext4 rw rootwait maxcpus=1 fixrtc
console=ttyO2,115200n8 kgdboc=ttyO2,115200n8"
```

geändert:

~~`quiet`~~: Kernel-Meldungen ausgeben

~~`splash`~~: Graphisches Boot-Logo deaktivieren

`console`: Serielle Konsole aktivieren

`kgdboc`: Kernel debugger auf serieller Konsole aktivieren

# 6 Implementierung

## 6.1 Algorithmus

Der Datentransfer findet über einen Ringpuffer in einem gemeinsam verwendeten Speicherbereich statt, auf den beide OS zugreifen und dessen Adresse und Größe von AUTOSAR bestimmt und an Linux weitergegeben wird. In diesem Puffer befinden sich Zählervariablen und Nachrichten fester Größe, die aus einer momentan beliebig belegbaren Sequenz- und einer Kommandonummer und dem Nachrichtentext bestehen.

Vor dem Eintragen einer neuen Nachricht überprüft der producer, ob der Puffer bereits voll ist. Sollte das der Fall sein, passiert nichts und es wird zum Aufrufer zurückgekehrt. Ansonsten wird die Nachricht im Puffer eingetragen und `produceCount` erhöht.

Der consumer überprüft periodisch, ob `consumeCount` bereits `produceCount` erreicht hat. Solange das nicht der Fall ist, werden Nachrichten entnommen und `consumeCount` erhöht.

## 6.2 producer-consumer.h

In dieser auf beiden Seiten vorhandenen header-Datei befinden sich `#define`s, globale Variablen und eine `struct`:

- `BUFFER_OFFSET (Linux)`
  Physical address (s.u.) des Nachrichten-Puffers
- `BUFFER_SIZE`
  1 MB
- `CMD_*`
  Über diese Konstanten können verschiedene Aktionen beim consumer ausgelöst werden.
- `*pt (AUTOSAR: in os_simple.c)`
  Der eigentliche Puffer
- `*produceCount (AUTOSAR: in os_simple.c)`
  Anzahl der bereits produzierten Nachrichten
- `*consumeCount (AUTOSAR: in os_simple.c)`
  Anzahl der bereits konsumierten Nachrichten
- `typedef struct message`
  `struct` für einen komfortableren Zugriff auf die Bestandteile einer Nachricht
- `BUFFER_CAPACITY`
  Anzahl der Nachrichten, die sich gleichzeitig im Puffer befinden können.

## 6.3 Producer

AUTOSAR übernimmt die Rolle des producers.

### 6.3.1 producer_init()

Diese Funktion muß vor dem erstmaligen Versenden einer Nachricht aufgerufen werden.

Zunächst wird in `calc_mem_param()` die Endadresse des Linux-Kernel-Speichers und damit die Anfangsadresse des Puffers bestimmt. Da diese sowohl hier als auch bei der Generierung der Linux-Kernel-Parameter benötigt wird, habe ich diese Funktion in der Datei `src/system/ram_size.c` abgelegt, auf die von beiden Stellen zugegriffen wird.

Der gesamte Speicher ist unter AUTOSAR nicht gecached.

Dann werden die Adressen der Pointer produceCount und consumeCount gesetzt und deren Inhalt auf 0 gesetzt. Damit ergibt sich folgende memory map:

| | |
|---|---|
| 0xBFFFFFFF | Ende des Puffers |
| 0xBFF00000 | Anfang des Puffers |
| 0xA0000000 | Anfang des RAM-Abschnitts gemäß kernel command line mem=511M@0xA0000000 |

**Tabelle 1 Memory map**

| | |
|---|---|
| … | |
| 272 | `message 1` |
| 8 | `message 0` |
| 4 | `consumeCount` |
| 0 | `produceCount` |

**Tabelle 2 Aufbau des Puffers**

| | |
|---|---|
| 263 | `payload` Ende |
| 8 | `payload` Beginn |
| 4 | `command` |
| 0 | `seqence_number` |

**Tabelle 3 Aufbau einer message**

Mit `LDEBUG_PRINTF()` können Ausgaben in ein Protokoll im RAM erfolgen, daß man im Debugger untersuchen kann.

### 6.3.2 produce_message(int sequence_number, int command, char * payload)

Das ist die Funktion, mit der AUTOSAR-Tasks Nachrichten an Linux schicken können.

Wenn der Puffer nicht bereits voll ist, wird eine neue message angelegt. Mittels

```
char * strcpy(char *,const char *);
```

wird ein String als payload in die Nachricht kopiert.

Vor und nach dem Lesen von `*count` und Erhöhen von `produceCount` wird mit `mb()` eine memory barrier eingerichtet, die das u.g. Umstellen von Lese- und Schreibzugriffen über diese barrier hinweg verhindert.

## 6.4 producerTask()

Um den Test-Task bei AUTOSAR zu registrieren muß die Deklaration der Einstiegsfunktion und die Task-ID in der Datei `config/Os_Cfg.h` vorhanden sein. Bei der ID-Vergabe ist auf eine fortlaufende Nummerierung zu achten. Außerdem muß die Anzahl der Tasks erhöht werden:

```
#define TASK_ID_producerTask  2
```

```
void producerTask( void );
#define OS_TASK_CNT                    3
```

Zusätzlich muß für den Ablauf des Tasks ein Stack zur Verfügung stehen. Dieser wird in der datei `config/Os_Cfg.c` reserviert. Dazu steht ein Makro zur Verfügung:

```
DECLARE_STACK(producerTask,2048);
```

Um den Task dann zu registrieren, muß dieser durch das Makro `GEN_BTASK()` in der Task-Datenstruktur dort ergänzt werden.

Implementiert ist er dann in der Datei `src/os_simple.c` und ruft dort nacheinander `producer_init()` und `producer_test()` auf. Letztere Funktion erzeugt kontinuierlich Testnachrichten mit einer Pause zwischen den Nachrichten.

## 6.5 Consumer

Linux übernimmt die Rolle des consumers.

### 6.5.1 consumer.c

```
#define pr_fmt(fmt) KBUILD_MODNAME ": " fmt
```

sorgt dafür, daß den `dmesg`-Ausgaben mitels `pr_*()` der Modulname vorangestellt wird.
Die Modul-Parameter für Startadresse und Größe des Puffers werden durch das Makro `module_param` festgelegt.

### 6.5.2 consumer_init()

Alle Linux-Kernel-Module haben einen entry-point, der beim Laden aufgerufen wird. Die eigene Funktion wird mit

```
module_init();
```

registriert.
Hier wird zuerst das Mapping eingerichtet.
Im Linux-Kernel hat man es mit verschiedenen Adresstypen zu tun [21]: Physical addresses werden dabei von der Hardware, virtual addresses vom Kernel bereitgestellt.

- User virtual addresses
  Jeder user-space-Prozeß hat einen eigenen virtuellen Adressraum mit solchen Adressen.
- Physical addresses
  Die auf dem Adressbus zwischen CPU und Arbeitsspeicher vorhandenen Adressen.
- Bus addresses
  Die auf dem Adressbus zwischen Peripherie und Arbeitsspeicher vorhandenen Adressen.
- Kernel logical addresses
  Die normalerweise im Kernel verwendeten Adressen. `kmalloc` gibt solche Adressen zurück.
- Kernel virtual addresses
  Ähnlich wie Kernel logical addresses, allerdings nicht notwendigerweise mit der 1:1-Zuordnung zu Physical addresses. `vmmalloc` gibt solche Adressen zurück.

Da der Puffer außerhalb des vom Kernel verwalteten Speichers liegt, wird eine Umsetzung von einer Physical addresses auf eine Kernel virtual addresses benötigt.
Beim Zugriff auf Inhalte des RAMs über Kerne hinweg gilt es nicht offensichtliche Optimierungen zu beachten [22]. Es gibt auf der einen Seite Hardware-Optimierungen wie Caches, die dazu führen, daß nicht immer alle Kerne die gleichen Daten sehen. Auf der anderen Seite

Compiler-Optimierungen wie das Umstellen von Lese- und Schreibzugriffen. Einige Beispiele sind unter [23] aufgeführt.
Um die Caches zu deaktivieren, wird der Speicherbereich mit

```
void *ioremap_nocache(unsigned long phys_addr, unsigned long size);
```

gemappt.
Damit `intrpt_routine()` periodisch aufgerufen wird, habe ich einen kernel workqueue eingesetzt, der mittels

```
create_workqueue(name)
```

erstellt und mittels

```
int queue_delayed_work(struct workqueue_struct *wq, struct delayed_work
*work, unsigned long delay);
```

aktiviert wird. Diese kernel workqueues dienen dazu, eine Funktion eine gewisse Zeit in der Zukunft aufrufen zu lassen.

### 6.5.3 intrpt_routine()

Diese Routine wird durch den kernel workqueue periodisch aufgerufen. Sollte das Flag `consumer_die` nicht gesetzt sein, wird `consumer()` aufgerufen. Danach stellt sie sich selber mit `queue_delayed_work()` erneut in den Queue. Die Angabe von `HZ/10` als `delay` (Einheit: Jiffies; die Zeit zwischen zwei Timer-Interrupts) bewirkt zehn Aufrufe pro Sekunde, da `HZ` als Jiffies pro Sekunde definiert ist. Geringere Latenzen sind auf Kosten von overhead durch häufigere Überprüfung auf neue Nachrichten mit Bruchteilen (z.B. `HZ/100` für 10 ms bzw. 100 Hz) möglich.

### 6.5.4 consumer()

Wenn der Puffer nicht leer ist, wird die nächste Nachricht ausgelesen und anhand des Inhalts von `command` weiter verzweigt. Hier könnten auch andere Kommandos implementiert werden. Der Mechanismus kommt ohne Sperren aus, da `produceCount` nur in `producer()` und `consumeCount` nur in `consumer()` verändert werden. Die zeitlichen Rahmenbedingungen eines Echtzeit-Betriebssystems wie AUTOSAR werden also nicht tangiert.

Da `produceCount` und `consumeCount` nur auf Gleich-/Ungleichheit getestet werden und keine Ordnungsrelationen mittels < oder > aufgebaut werden, stellen weder ein Überlauf noch der gleichzeitige Lese-/Schreib-Zugriff ein Problem dar.

Ein Problem könnte höchstens dann auftreten, wenn der producer $2^{32}$ messages erzeugt, bevor der consumer überprüft, ob neue vorhanden sind, und somit ein Überlauf auftritt. Das ist aber aufgrund des begrenzten Platzes nicht möglich.

Um kritische Stellen werden wieder memory barriers eingerichtet. Dabei gilt laut der Kernel-Dokumentation in Documentation/memory-barriers.txt: „All memory barriers [...] imply a compiler barrier.“

### 6.5.5 write_file()

Routine, um aus dem Kernel heraus eine Datei im Dateisystem anzulegen. Das wird zwar laut [24] nicht empfohlen, aber ebenda beschrieben und soll hier im Sinne eines proof of concept genügen. Korrekter wäre es, das im user space zu erledigen.

### 6.5.6 consumer_exit()

Die beim Entfernen eines Moduls aufzurufende Funktion wird mit

```
module_exit();
```

als exit-point analog zu `module_init()` registriert.

Durch setzen des Flags `consumer_die` wird das erneute Einstellen in den workqueue verhindert. Neue Nachrichten werden dann von Linux nicht mehr abgeholt und der Puffer wird zwangsläufig vollaufen, wenn AUTOSAR zu viele weitere Nachrichten einstellt.
Mittels

```
void flush_workqueue(struct workqueue_struct *queue);
```

wird auf das Ende noch laufender workqueues gewartet, mittels

```
void destroy_workqueue(struct workqueue_struct *queue);
```

wird der Queue dann gelöscht.
Daraufhin wird das Mapping des Speicherbereich mit

```
void iounmap(void * addr);
```

aufgehoben.

# 7 Test

## 7.1 AUTOSAR

Man kann während des Testens die Größe und damit die Übertragungsdauer der ELF-Datei erheblich senken, indem man in `linkscript_econnect_gcc.ldf` folgende Zeile auskommentiert und damit den Linux-Kernel aus der ELF-Datei herausnimmt:

```
#define INCLUDE_LINUX_KERNEL 1
```

Folgende Debug Configuration ist in Arctic Studio anzulegen:

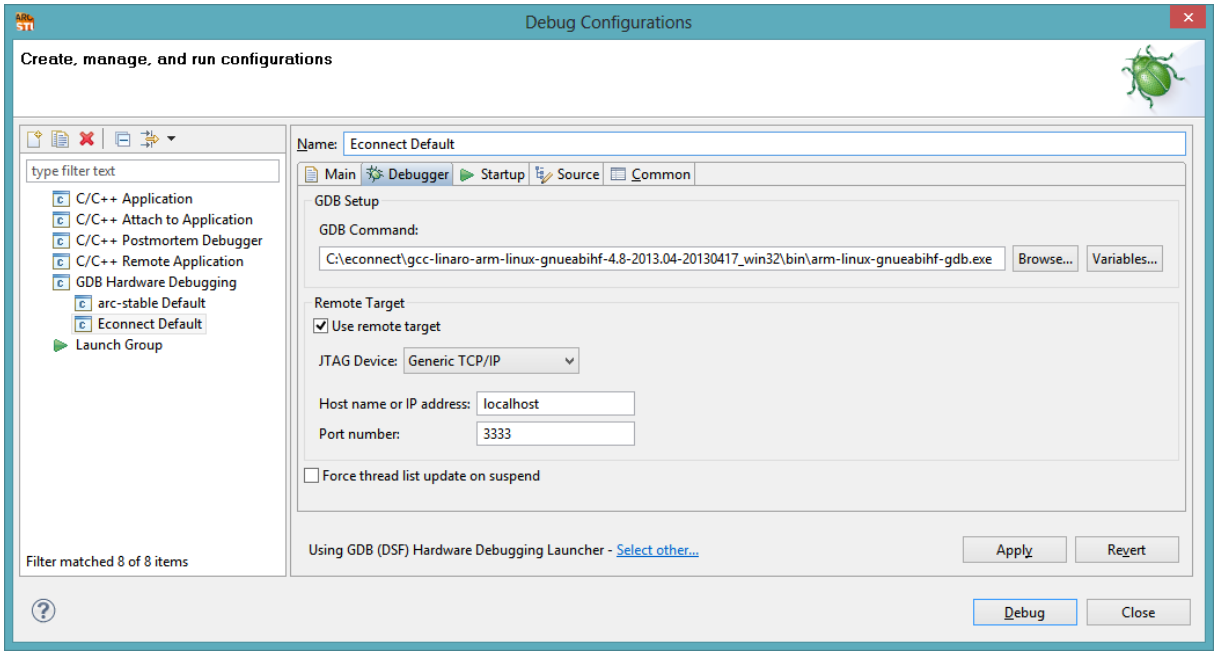


**Abbildung 10 Arctic Studio Debug Configurations**

Da Artic Studio nicht automatisch dafür sorgt, ist vor dem Ausführen immer manuell „Save All“ und „Build Project“ aufzurufen. Beim Start des Arctic Studio-Befehls „Debug“ wird die ELF-Datei übertragen und auf core 0 ausgeführt.
Der producer wird beim Starten von AUTOSAR initialisiert und erzeugt dann ca. 10 messages/Sekunde.

## 7.2 Linux

### 7.2.1 Cross compiling

Um auf einem schnelleren Intel-basierten PC kompilieren zu können, ist eine cross compile toolchain nötig. Diese ist von Linaro erhältlich, auf der Website[6] ist `gcc-linaro-arm-linux-gnueabihf-*_linux.tar.xz` herunterzuladen. In `gcc-linaro-arm-linux-gnueabihf-*_win32.zip` befindet sich der GNU Debugger gdb für Windows. Bei einem 64-bit-Debian sind mit

[6] https://launchpad.net/linaro-toolchain-binaries/+download

```
sudo apt-get install ia32-libs
```

noch die für 32-bit-binaries nötigen Bibliotheken zu installieren.
Um Kernel-Module außerhalb des kernel trees kompilieren zu können, ist zunächst ein konfigurierter und kompilierter kernel tree nötig, dessen Versionsnummer mit dem auf dem Pandaboard laufenden Kernel exakt übereinstimmt.
Nachdem beides in ein Verzeichnis entpackt wurde, wechselt man in das Kernel-Verzeichnis. Dann kann er kompiliert werden:

```
# make –j8 ARCH=arm CROSS_COMPILE=../gcc-linaro-arm-linux-gnueabihf-
*_linux/bin/arm-linux-gnueabihf-
```

Mein `Makefile` ermöglicht es, das implementierte Kernel-Modul mit dem gleichen `make`-Kommando wie beim Kernel zu kompilieren, nachdem man in das Verzeichnis gewechselt ist. Das darin definierte `KDIR` muß dafür auf den kernel tree zeigen.

### 7.2.2 Start

Das out-of-tree-building ist sehr praktisch zum Entwickeln, da nur mit einem kleinen Kernel-Modul, das man mit `insmod` laden und entladen kann, und nicht dem kompletten Kernel hantieren muß. Dafür muß momentan noch manuell die Startadresse und Größe des Puffers beim Laden des Moduls übergeben werden. Sobald das Modul in den kernel tree übernommen wird, sollte die Übergabe über die Kernel-Kommandozeilenparameter funktionieren.
Diese werden auf der Konsole durch

```
# cat /proc/cmdline
elevator=noop vram=40M root=/dev/mmcblk0p2 rootfstype=ext4 rw rootwait
maxcpus=1 fixrtc console=ttyO2,115200n8 mem=32M@0x80000000
mem=986M@0x82500000 autosar_stackp=0x40301ff8
consume.buffer_start=0xbff00000 consume.buffer_size=0x100000
```

angezeigt. Der Start erfolgt dann durch Laden des Kernel:Moduls:

```
# insmod consume.ko buffer_start=0xbff00000 buffer_size=0x100000
# cat /proc/kmsg
[   57.900878] consume: ioremapping 0x00100000 bytes of memory @
0xbff00000-0xbfffffff w/capacity of 3971 messages
[   57.900878] consume: sizeof(message): 264
[   57.900939] consume: ioremap returned f1e00000
[   57.900970] consume: produceCount: f1e00000
[   57.900970] consume: consumeCount: f1e00004
[   57.900970] consume: Start of messages: 0xf1e00008
```

Ausgegeben werden die Adressen des Kommunikations-Puffers.

Man kann mit

```
# cat /proc/kmsg
```

erkennen, wie fortlaufend Nachrichten empfangen werden:

```
[41204.083007] consume: m.sequence_number=0, command=2, payload=Test
[41204.176757] consume: m.sequence_number=1, command=2, payload=Test
[41204.176757] consume: m.sequence_number=2, command=2, payload=Test
```

Um die übertragenen Daten bei Verwendung von `CMD_WRITE_FILE` zu überprüfen:

```
# cat /tmp/test.txt
Test
```

### 7.2.3 kdb

OpenOCD ermöglicht in der verwendeten Version 0.7.0 noch nicht, beide A9-Kerne gleichzeitig zu debuggen. Mit dem Linux Kernel Debugger kann man sich über die serielle Schnittstelle u.a. den Inhalt des RAMs anzeigen lassen. Dazu ist er mit

```
echo g > /proc/sysrq-trigger
```

zu starten, das System wird daraufhin angehalten. Auf der seriellen Konsole erscheint jetzt der Prompt von kdb. Als Parameter für `md` ist die Adresse der Meldung `ioremap returned` anzugeben:

```
[910253.933044] SysRq : DEBUG

Entering kdb (current=0xed31d280, pid 4749) on processor 0 due to
Keyboard Entry

[0]kdb> md 0xf1e00000
0xf1e00000 00000004 00000004 00000001 00000001   ................
0xf1e00010 74736554 0000000a 00000000 00000000   Test............
0xf1e00020 00000000 00000000 00000000 00000000   ................
0xf1e00030-0xf1e0006f zero suppressed
0xf1e00070 00000000 00000000 00000000 00000000   ................
```

Ein Zugriff über die eigentliche Physical addresses ist allerdings nicht möglich, da diese außerhalb des Kernelspeichers liegt:

```
[0]kdb> mdp 0xbff00000
phys 0xbff00000
phys 0xbff00010
phys 0xbff00020
phys 0xbff00030
phys 0xbff00040
phys 0xbff00050
phys 0xbff00060
phys 0xbff00070
```

Mit

```
[0]kdb> go
```

beendet man kdb und läßt das System weiterlaufen.

# 8 Ausblick

Die Effizienz könnte durch genauere Analyse und Reduzierung der benötigten memory barriers erhöht werden.

Eine mögliche Erweiterung wäre, einen weiteren Puffer symmetrisch in die entgegengesetzte Richtung zu implementieren und AUTOSAR mittels eines periodischen Extended Task nach neuen Nachrichten prüfen zu lassen. Dann könnte zum einen Linux Zugriff auf den ggf. unter Kontrolle von AUTOSAR stehenden CAN-Controller erhalten, zum anderen wären unter Verwendung von `seqence_number` Rückgabewerte möglich.

Des weiteren könnte die Latenz, die durch das Polling verursacht wird, durch Umstellung auf IPIs vermieden werden.

# 9 Fazit

Am Anfang meiner Arbeit erschienen mir die Herausforderungen schwer überwindbar. Doch mit den Vorarbeiten der Hochschule, zwei verschiedene Betriebssysteme auf einer dual core-CPU zu starten, dem Ausräumen der Probleme mit dem Pandaboard und der erfolgreichen Inbetriebnahme eines günstigen JTAG-Debuggers konnte ich im Zuge meines Fernstudiums auch zuhause mit dieser komplexen, mir bisher unbekannten Software-Umgebung arbeiten. Eine überraschende Erkenntnis stellte die Unabhängigkeit der beiden Kerne dar: Während Linux auf dem einen Kern weiterläuft, kann auf dem anderen AUTOSAR ohne Beeinträchtigung gestartet und beendet werden, als ob man mit zwei eigenständigen Systemen agieren würde.

Mit dem von mir implementierten Mechanismus ist es möglich, über Prozessor- und Betriebssystemgrenzen hinweg Daten zu schicken und in eine Datei auf der SD Card zu schreiben, was der Anforderung entspricht. Auch unter Last war ein zuverlässiger Datenaustausch möglich.

Der Algorithmus kommt ohne Sperren aus, blockiert deshalb nicht und beeinträchtigt somit nicht die Echtzeit-Anforderungen von AUTOSAR. Das ist auch vorteilhaft im Hinblick auf die geringe Unterstützung für atomare Operationen auf ARM-CPUs.

Nachteilig ist die Tatsache, daß der Puffer vollaufen und dabei nicht das älteste Datum verfallen gelassen werden kann. Dies sollte zumindest in einem Flag festgehalten werden. Außerdem geht das verwendete Polling immer mit einer niedrigeren Effizienz einher als ein ereignisgesteuerter Ansatz.

# Literatur


**[1] A. S. Tanenbaum, Modern Operating Systems, Prentice Hall International, 2007.**

**[2] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat und P. Stenström, „The Worst-Case Execution Time Problem — Overview of Methods and Survey of Tools," *TECS,* Nr. 7 (3), 2008.**

**[3] AUTOSAR 3.1 Specification, 2010.**

**[4] H. Sutter, „The Free Lunch Is Over - A Fundamental Turn Toward Concurrency in Software," *Dr. Dobb's Journal,* Nr. 30(3), 2005.**

**[5] „Projektbeschreibung ProSyMig," [Online]. Available: http://www.hochschule-trier.de/index.php?id=8028.**

**[6] „econnect Germany," [Online]. Available: http://www.econnect-germany.de.**

**[7] „econnect Trier," [Online]. Available: http://www.econnect-trier.de.**

**[8] „Forschungsverbund Verkehrstechnik und Verkehrssicherheit (FVV)," [Online]. Available: http://www.fvv-trier.de/.**

**[9] J. Schneider, „TVB der FH Trier zum Technologiewettbewerb des BMWi: econnect Trier-FVV: IKT für Elektromobilität in der Region Trier – Vom Windrad zum Elektroauto," Fachhochschule Trier, 2011.**

**[10] T. Nett, „Systemarchitektur für Econnect," 2013.**

**[11] J. Wietzke, Embedded Technologies, Springer, 2012.**

**[12] G. Kroah-Hartman, „kdbus - IPC for the modern world".**

**[13] „RPMsg," [Online]. Available: http://www.omappedia.org/wiki/Category:RPMsg.**

**[14] J. Day, „Handling key-on CAN-timing limitations with multi-core processors in body electronics," [Online]. Available: http://johndayautomotiveelectronics.com/?p=6119.**

**[15] T. Nett und J. Schneider, „Running Linux and AUTOSAR side by side [unpublished manuscript]," Hochschule Trier, 2013.**

**[16] OMAP4460 Pandaboard ES System Reference Manual, pandaboard.org, 2011.**

**[17] Texas Instruments Incorporated, OMAP4460 Data Manual, Texas Instruments Incorporated, 2012.**

**[18] „Flash memory card design," [Online]. Available: https://wiki.linaro.org/WorkingGroups/KernelArchived/Projects/FlashCardSurvey.**

**[19] 1149.1 - IEEE Standard for Test Access Port and Boundary-Scan Architecture, 2013.**

**[20] D. Rath, Open On-Chip Debugger, FH Augsburg, 2005.**

**[21] J. Corbet, A. Rubini und G. Kroah-Hartman, Linux Device Drivers, O'Reilly, 2005.**

**[22] P. E. McKenney, Is Parallel Programming Hard, And, If So, What Can You Do About It?, 2013.**

**[23] „SMP Primer for Android," [Online]. Available: http://developer.android.com/training/articles/smp.html.**

**[24] G. Kroah-Hartman, „Driving Me Nuts - Things You Never Should Do in the Kernel," *Linux Journal,* Nr. 133/May, 2005.**


Die URLs wurden zum Zeitpunkt der Erstellung der Arbeit aufgerufen.
Microsoft Word-Literaturverzeichnisformat: IEEE 2006

# Index

***A***

Adresstypen ..... 19
AMP ..... 6

***C***

Cross compiling ..... 22

***I***

IPI ..... 9

***K***

kdb ..... 24

***O***

OMAP4460 ..... 11

***P***

Pandaboard ..... 10
Polling ..... 9
producer/consumer ..... 9

***R***

RPMsg ..... 7

***S***

SMP ..... 6

***U***

uboot ..... 16

# Glossar

| | |
|---|---|
| QSB | Quick Start Board |
| CAN | Controller Area Network |
| SoC | System on Chip |
| IDE | Integrated Development Environment |
| IPC | Inter-process communication |
| PoP | Package on Package |
| gdb | GNU debugger |
| kdb | Kernel debugger |

# Anhang

autosar/: Teile des Arctic Studio-Quellcode
producer-consumer/: Quellcode und Makefile des Linux-Moduls