

# *Embedded Systems - Einführung (1)*

Jürgen Quade, Hochschule Niederrhein

27.03.2024

## 1 Organisation

## 2 Theoretische Grundlagen

## 3 Betriebssysteme

## 4 Unix Systemkonzepte

## 5 Kommandoübersicht

## 6 Rechtliche Aspekte

**Prof. Dr.-Ing. Jürgen Quade**  
**Hochschule Niederrhein**

Wie viele Desktop-Rechner, Server, Notebooks und Netbooks  
sind im letzten Jahr *in etwa* verkauft worden?

Wie viele Desktop-Rechner, Server, Notebooks und Netbooks sind 2022 in etwa verkauft worden?

Ca. 260.000.000  
(260 Millionen)

Wie viele Smartphones, Mobiltelefone und Tablets sind im letzten Jahr verkauft worden?

Wie viele Smartphones, Mobiltelefone und Tablets sind 2020 verkauft worden?

**2.200.000.000**

(~2.2 Milliarden + 200 Millionen Wearables)

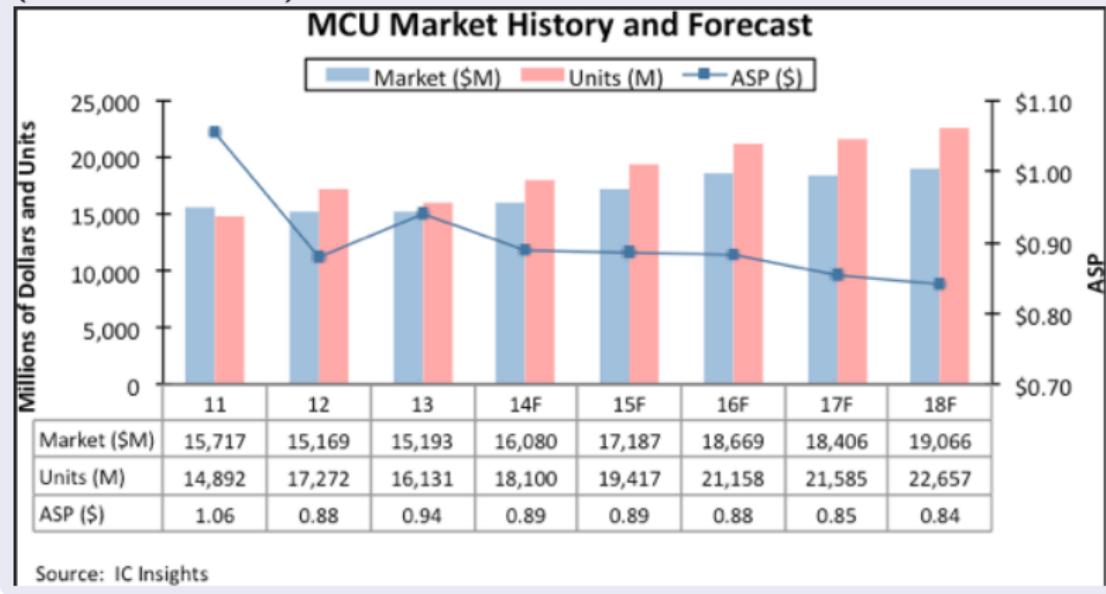
(bei einer Weltbevölkerung von ca. 7.8 Milliarden)

Wie viele Microcontroller sind im letzten Jahr in etwa verkauft worden?

## Wie viele Microcontroller sind 2021 in etwa verkauft worden?

> 50.000.000.000

(>50 Milliarden)



# 250 Milliarden ARM-Chips hergestellt

Embedded  
Systems -  
Einführung (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation  
Theoretische  
Grundlagen  
Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübersicht

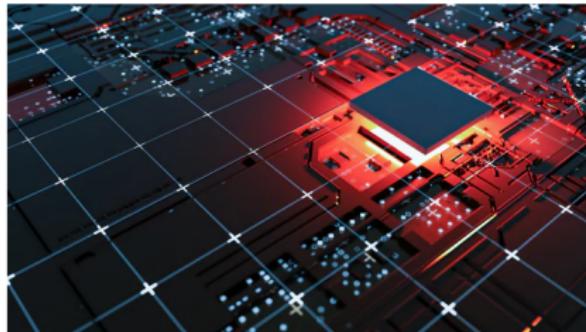
Rechtliche  
Aspekte

## Mehr als 250 Milliarden ARM-Chips hergestellt

ARM trommelt mit einer hohen Umsatzsteigerung für seinen kommenden Börsengang. Bei der Verbreitung wurde im September 2022 ein Meilenstein erreicht.

Lesezeit: 2 Min.  In Pocket speichern

   9



(Bild: cherezoff / Shutterstock.com)

07.02.2023 17:31 Uhr

Von Mark Mantel

Das zweite Jahr in Folge gibt ARM einen aktuellen Einblick in die eigenen Geschäftszahlen. Formell ist die privatwirtschaftliche Firma nicht dazu verpflichtet, allerdings spielt ARM das Wachstum in die Hände, um sich im Vorfeld des geplanten Börsengangs möglichst gut darzustellen.

# “Hier spielt die Musik”

Der Markt der eingebetteten Systeme hat eine erheblich größere Relevanz als der Markt der Desktop- und Server-Systeme.

In diesem Markt wird zunehmend und vorwiegend Linux eingesetzt.

14.01.2011 14:40

## ARM-Chef dämpft Spekulationen über Windows-Systeme und Server



ARM-CEO  
Warren East  
Bild: ARM

“Wir haben nie auf den PC-Markt gezielt”: Damit klärt **Warren East**[1], CEO der britischen CPU-Schmiede ARM, die Ambitionen seiner Firma. Im Interview[2] mit Networkworld nimmt er Stellung zu den aufgeregten diskutierten Neuheiten, Überraschungen und Spekulationen der vergangenen Monate: über eine **vollwertige Windows-Version**[3] auch für ARM-SoCs[4], ARM-SoCs für **Server**[5] und eine angeblich bald zu erwartende 64-Bit-Erweiterung der ARM-Architektur.

In jedem klassischen Windows-PC, so East, ist ARM bereits vertreten, meistens sogar mehrfach: ARM-Kerne stecken in den Controller-Chips von Festplatten, SSDs, optischen Laufwerken sowie in vielen Hostadapters und möglicherweise sogar in Mainboard-Chipsätzen. Gegen mehr als **1,5 Milliarden**[6] Chips mit ARM-Kernen pro Quartal nehmen sich die rund

**90 Millionen**[7] PCs und Notebooks geradezu kümmerlich aus. Der Markt der Prozessoren für typische PCs sei für ARM finanziell deshalb nicht interessant.

East spricht dabei allerdings ausschließlich für sein Unternehmen, das keine Chips verkauft, sondern lediglich Lizenzen und Geistiges Eigentum (IP-Cores). ARM-Kunde Nvidia möchte laut[8] seinem Chef Jen-Hsun Huang ausdrücklich ARM-SoCs für “PCs und Supercomputer” entwickeln.

Eine Abschätzung, wann eine künftige Windows-Version für ARM-SoCs erscheinen könnte, wagt Warren East nicht – er verweist auf Microsoft. Er betont jedoch, dass der Aufwand für eine solche Portierung eines Betriebssystems viel Arbeit erfordere, weil ja auch Gerätetreiber und Anwendungsprogramme nötig seien.

In Bezug auf die bereits in der Entwicklung befindlichen Server mit ARM-Prozessoren dämpft East die Erwartungen ebenfalls: “Das Projekt liegt im Zeitplan, aber ich habe immer gesagt: Bitte erwarten Sie nicht, dass ARM-Server vor 2014 spürbare Veränderungen bringen.”

Noch länger dürfte es dauern, bis eine 64-Bit-taugliche ARM-Mikroarchitektur verfügbar ist. Mancher hat schon über “ARM64” spekuliert, doch laut East kommt zunächst der **LPAE**[9]-taugliche HyperVisor für **Cortex-A15**[10] (Eagle). 64-Bit-Tauglichkeit sei in der Zukunft zwar “unvermeidlich”, doch ARM sei “ein Wirtschaftsunternehmen (...) mit endlichen Ressourcen”. (ciw[11])

## Wo finden wir Mikroprozessoren und Mikrocontroller?

# Überall

Embedded  
Systems -  
Einführung (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation

Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübersicht

Rechtliche  
Aspekte



## Computer sind überall

Aber wir nehmen sie nicht mehr wahr. Sie sind *eingebettet und invisible*.

# Zielsetzung (1)

Wir werden uns mit der Architektur, der Entwicklung, dem Betrieb und der Programmierung eingebetteter Linux-Systeme beschäftigen.

## Sie lernen

- die Differenzierung eingebetteter Systeme
- welche Vor- und Nachteile eingebettete Systeme im Vergleich zu Standardsystemen haben,
- wie (eingebettete) Linux-Systeme **von Grund auf** funktionieren,
- welche grundsätzlichen Technologien zur Realisierung zur Verfügung stehen (Stichwort initramfs) und
- die grundsätzlichen Probleme im Umfeld eingebetteter Systeme (Flash, Entwicklungsmethodik).

# Zielsetzung (2)

Embedded  
Systems -  
Einführung (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation

Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübersicht

Rechtliche  
Aspekte

## Sie lernen

- das Bauen (eingebetteter) Linux-Systeme,
- eine Host/Target-Entwicklung kennen,
- welche Werkzeuge einen beim Bauen von Linux-Systemen unterstützen,
- das Aufsetzen einer Cross-Entwicklungsumgebung,
- wie Applikationen in eingebettete Linux-Systeme integriert werden,
- wie Updates authentifiziert und verschlüsselt verteilt werden und
- was sich hinter *Security Vulnerability Scanning* verbirgt.

# Zielsetzung (3)

Embedded  
Systems -  
Einführung (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation

Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübersicht

Rechtliche  
Aspekte

## Sie lernen

- wie Gerätetreiber programmiert werden,
- wie der Zugriff auf Hardware realisiert wird
- Kernelprogrammierung (Kernel-Threads, Timer),
- welche kritischen Bereiche es gibt und
- wie diese geschützt werden können.
- Aspekte des *professionellen* Betriebs (Ausrollen, Provisioning, Update)

# Voraussetzungen

*Embedded  
Systems -  
Einführung (1)*

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation

Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübersicht

Rechtliche  
Aspekte

- Programmierung in C
- Betriebssysteme
- Systemadministration
- Linux

## Section 1

### Organisation

# Allgemeines

Embedded  
Systems -  
Einführung (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation

Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübersicht

Rechtliche  
Aspekte

- **Seminaritsche Lehrveranstaltung in Präsenz**
  - Vorlesungsanteile zur Inhaltsvermittlung
  - Praktische Übungen (auch im Selbststudium) an PC und Raspberry Pi
  - Praktische Hausarbeiten
- Unterlagen finden Sie im Moodle-Lernraum
- Pro Woche müssen Sie gemäß Modulhandbuch 10 Zeitstunden für die Veranstaltung einplanen.

## Empfehlung

**Forum aktiv nutzen!**

# Vorlesungszeit

*Embedded  
Systems -  
Einführung (1)*

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation

Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübersicht

Rechtliche  
Aspekte

- Dienstag, 13.00 Uhr - 17:30 Uhr (5 SWS)

# Praktische Übungen und Hausarbeiten

*Embedded  
Systems -  
Einführung (1)*

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation  
Theoretische  
Grundlagen  
Betriebssysteme  
Unix Sys-  
temkonzepte  
Kommandoübersicht  
Rechtliche  
Aspekte

Sie benötigen ...

- einen Entwicklungsrechner unter Linux
  - Version beispielsweise Ubuntu 22.04
  - Kann in einer virtuellen Maschine installiert werden
- Raspberry Pi inklusive serielllem Interface
- Lan-Kabel zur Verbindung PC und Raspberry Pi

# Raspberry Pi

*Embedded  
Systems -  
Einführung (1)*

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation

Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübersicht

Rechtliche  
Aspekte

Wer benötigt einen Raspberry Pi?

# Material zur Veranstaltung

## Embedded Systems - Einführung (1)

Jürgen Quade,  
Hochschule  
Niederrhein

## Organisation

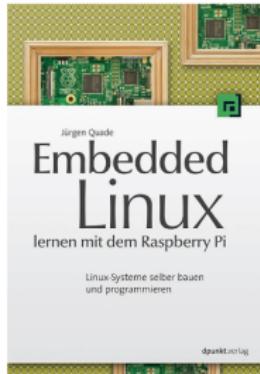
Theoretische  
Grundlagen

Betriebssysteme

Unix Systemkonzepte

Kommandoübersicht

Rechtliche  
Aspekte



# Prüfung

*Embedded  
Systems -  
Einführung (1)*

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation

Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübersicht

Rechtliche  
Aspekte

- schriftlich dokumentierte Projektarbeit
  - Dokumentation in Markdown
  - Zeit: rund sechs Wochen nach Vorlesungsende
  - Zu planender Aufwand: 40 Stunden (1 Woche)
- Aufgabenstellungen sind individualisiert
  - Gemäß Vorgabe wird ein eingebettetes System auf Basis Raspberry Pi aufgebaut

*Embedded  
Systems -  
Einführung (1)*

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation

Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübersicht

Rechtliche  
Aspekte

Fragen?

## Section 2

### Theoretische Grundlagen

# Definition Eingebettetes System

Embedded  
Systems -  
Einführung (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation

Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübersicht

Rechtliche  
Aspekte

## Integrierte, mikroelektronische Steuerung



# Ausprägungen

## Deeply Embedded

- 8/16/32 Bit (ohne MMU)
- Singlecore
- Software: Monitorprogramm plus Applikation, FreeRTOS
- **Systemsoftware und Applikation bilden eine Einheit**

## Open Embedded

- 32 Bit
- Single- und Multicore
- Software: Systemsoftware plus Applikation
- Systemsoftware und Applikation sind separate Komponenten

# Standard- versus Embedded-System

- Weniger Ressourcen
- Auf die Gerätefunktionen angepasste Hardwareplattformen
- Höhere Anforderungen
  - Kürzere Bootzeiten
  - Plötzliches Stromlos-Schalten
  - Dauerbetrieb
  - Langzeitbetrieb
  - Größe, Stromverbrauch
  - Robustheit (diskless, fanless, headless)
  - Safety und Security
- Exakte Anpassung der Systemsoftware an die gewünschte Funktionalität

# Standard- versus Embedded-System

## Probleme

- Eingeschränkte Ressourcen
  - Hintergrundspeicher
  - Hauptspeicher
  - Rechenleistung
- *Problematischer Hintergrundspeicher*
  - Limitierte Anzahl Schreibzyklen
- Cross-Entwicklung / Host/Target Entwicklung
- Update-Möglichkeiten

- Einsatz von Mikrocontrollern und SoC

## Advanced Risc Machine ARM

- Klarer Favorit: Advanced Risc Machine
- >6 Milliarden Prozessoren pro Jahr (!)
- Sehr gutes Verhältnis Leistung/Stromverbrauch
- ARM stellt das Design (Object, Source) zur Verfügung
- Unternehmen bauen die Prozessoren (SoCs)
  - Starke (HW-)Unterschiede zwischen den ARM-CPUs
- "Hier findet noch echte Entwicklung statt"

## Neuer Stern am Himmel

### Risc-V (Risc-Five)

- Offene Befehlssatz-Architektur
- 32, 64 und 128 Bit
- Modular/skalierbar (RV32I, RV32M, RV32F, RV32D, RV32A, ...)
  - einfache Controller für Deeply Embedded Systems
  - komplexe CPUs für Open Embedded, Server und Mainframes
- Open-Hardware, BSD-Lizenz
  - Diverse Designs stehen im Internet zum Download bereit

# Software

*Embedded  
Systems -  
Einführung (1)*

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation

Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübersicht

Rechtliche  
Aspekte

- Systemsoftware
  - Firmware (Bios)
  - Bootloader
  - (Linux-)Kernel
  - Rootfilesystem (Userland)
- Funktionsbestimmende Anwendung

# Software des Raspberry Pi

Embedded  
Systems -  
Einführung (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation

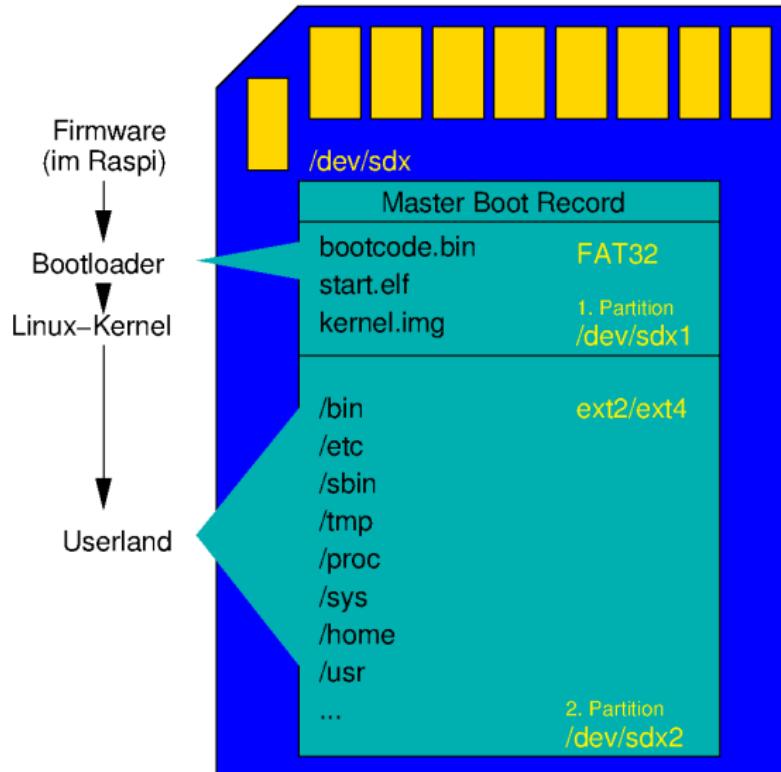
Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

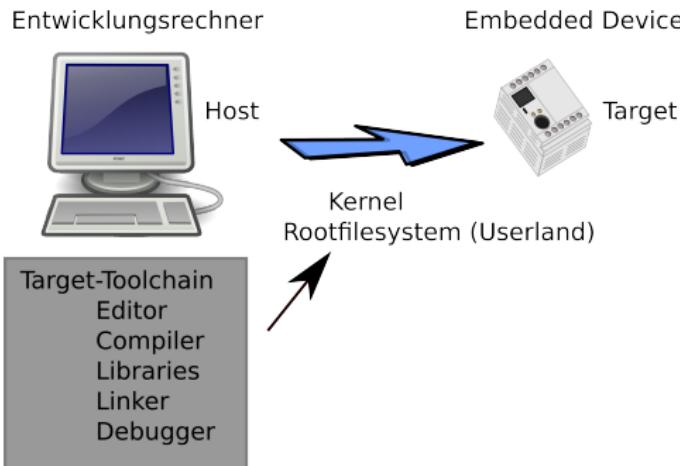
Kommandoübersicht

Rechtliche  
Aspekte



# Entwicklungsmethodik

- Cross-Entwicklung: Werkzeuge erzeugen Code für eine andere Architektur
- Host-Target: Entwicklungs- und Zielplattform (Host/Target) sind unterschiedlich



## Section 3

### Betriebssysteme

# Definition Betriebssystem

Embedded  
Systems -  
Einführung (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation

Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübersicht

Rechtliche  
Aspekte

Betriebssystem=Bezeichnung für alle Softwarekomponenten, die die

- Ausführung der Benutzerprogramme und die
- Verteilung der Ressourcen

ermöglichen, steuern und überwachen.

Es stellt den Benutzerprogrammen **Dienste** zur Verfügung, beispielsweise zum Ablegen von Daten auf Hintergrundspeicher.

# Systemarchitektur

Embedded  
Systems -  
Einführung (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation

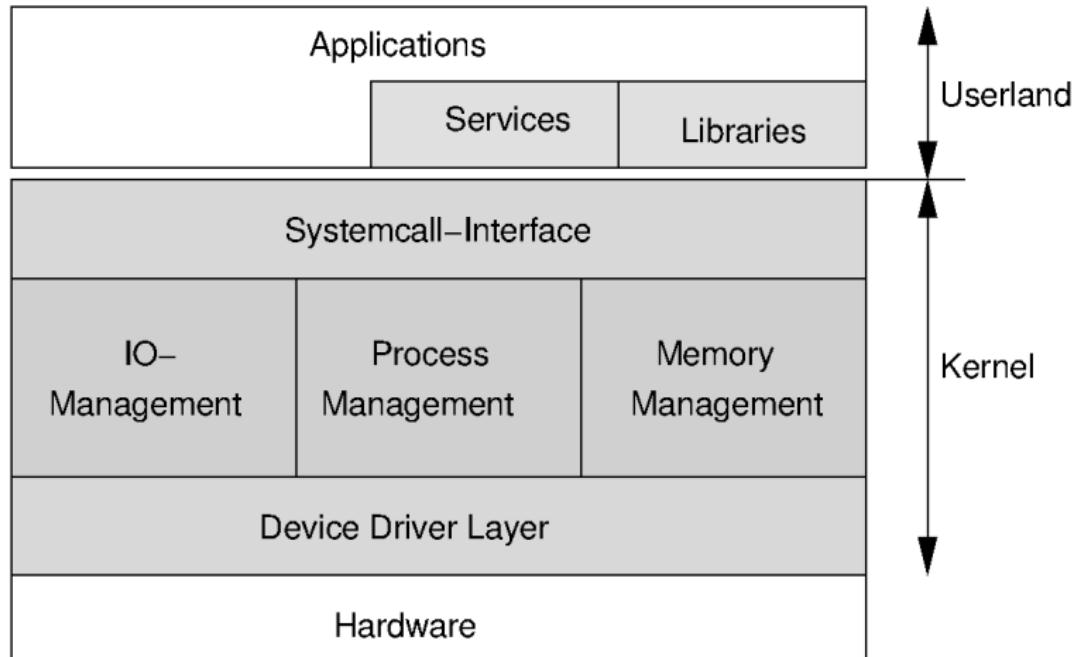
Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübersicht

Rechtliche  
Aspekte



# Kernel (Betriebssystemkern)

Embedded  
Systems -  
Einführung (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation

Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübersicht

Rechtliche  
Aspekte

- Zuständig für die Verteilung der Ressourcen
- Hardwarezugriffe

## Download

- Webseite: <https://www.kernel.org>
- Git-Archiv:  
`{git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git}`

## Linux-Versionsnummern

- 6.2.7
  - 6 = aktuelle Major-Version
  - 2 = Basisversion
  - 7 = Fehlerbereinigung

# Task-Management

- Verantwortlich für das Scheduling (Verwaltung der Ressource CPU)
  - Singlecore: Quasi-parallele Verarbeitung
  - Multicore: Real-parallele Verarbeitung
- Scheduling (ermöglicht Multitasking)

## Linux-Scheduling

Linux hat das Konzept der Scheduling-Klassen. Es ermöglicht die gleichzeitige Verwendung der folgenden Schedulingvarianten:

- Prioritätengesteuertes Scheduling
- Deadline-Scheduling (EDF)
- Zeitscheibenverfahren (Round Robin)
- First Come First Serve (FIFO)

# Memory Management

Embedded  
Systems -  
Einführung (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation  
Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübersicht

Rechtliche  
Aspekte

## Aufgaben

- ① Adressumsetzung
- ② Speicherschutz
- ③ Virtuellen Speicher zur Verfügung stellen
- ④ Erweiterten Speicher zur Verfügung stellen

## Virtueller Speicher

Unabhängig vom physisch vorhandenen Speicher kann der Anwender/Programmierer über beinahe den gesamten Adressraum verfügen.

## MMU

MMU=Memory Management Unit

Memory Management besteht aus *Hardware* und *Software*.  
Linux läuft nur auf Hardware mit MMU.

# IO-Management (1)

## Das IO-Management

- stellt dem Userland (Applikationen) ein einheitliches Programmierinterface zur Verfügung,
- ermöglicht die *systemkonforme* Integration von Hardware über die Gerätetreiber
- realisiert den performanten Zugriff auf Hardware (IO-Scheduling)
- implementiert Dateisysteme zur hierarchischen Ablage (Verzeichnisbaum) von Daten.

# IO-Management (2)

*Embedded  
Systems -  
Einführung (1)*

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation

Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübersicht

Rechtliche  
Aspekte

## Dateisystemtypen

- Linux-Filesystem ext2 oder ext4
- VFAT (SD-Karten, USB-Sticks, etc.)
- Flash-Filesysteme (!)
- Journaling Filesysteme
- Loop
- Overlay
- /proc und /sys

# Gerätetreiber

Embedded  
Systems -  
Einführung (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation  
Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübersicht

Rechtliche  
Aspekte

- Code, der den *professionellen* Zugriff auf Hardware steuert.
- Ermöglichen den performanten Zugriff.
- Gewährleisten die Betriebssicherheit (Safety).
- Teil des Betriebssystemkerns (häufig aber als Kernelmodul realisiert).

## Generierung

- Gerätetreiber müssen zum Kernel passen (Version).
- Zur Generierung werden die zum laufenden Kernel gehörenden Kernelquellen benötigt.

# Systemcalls

Embedded  
Systems -  
Einführung (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation  
Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübersicht

Rechtliche  
Aspekte

Dienste des Betriebssystemkerns, die von Applikationen über das **Systemcallinterface** genutzt werden.

Systemcalls werden über Software-Interrupts oder adäquate Befehle aufgerufen.

## Beispiele

- Zugriff auf Dateien und Peripherie: `open`, `close`, `read`, `write`
- Rechenprozesse erzeugen: `fork`, `clone`
- Zeit abfragen: `clock_gettime`
- ...

# Userland

*Embedded  
Systems -  
Einführung (1)*

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation  
Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübersicht

Rechtliche  
Aspekte

Userland = Rootfilesystem

Systemteile, die für den Betrieb notwendig sind, aber nicht im Kernel realisiert werden:

- Konfigurationsprogramme (z.B. dhcp)
- Bibliotheken
- Gerätedateien

## Section 4

### Unix Systemkonzepte

# Unix Systemkonzepte

## Ideen

- “Alles ist eine Datei”
- Pipes
- Dateitypen (werden über Attribute festgelegt)
  - Normale Dateien (ordinary Files)
  - Verzeichnisse (Ordner, Directories)
  - Gerätedateien (Char- und Blockgeräte)
  - Pipes
  - Links (symbolic, hard links)
- Verzeichnisbaum
  - Geräte werden zu Verzeichnissen abstrahiert; es gibt keine Laufwerke
- Virtuelle Dateisysteme (proc, sys)
- Superuser versus normaler User
- Eltern-Kind Taskhierarchie

# „Alles ist eine Datei“

- Zugriffsfunktionen: *lesen* und *schreiben*
- Peripherie (Tastatur, Motor, LED, Maus) wird als Datei (Gerätedatei) repräsentiert
  - Gerätedatei ist mit einem Gerätetreiber verknüpft
- Vorteil: Nur einen Satz von Funktionen merken.
- Auf Shellebene wird per cat und echo auf Dateien zugegriffen.

## Programmierinterface für den Dateizugriff

- `open` : *Kanal*- `close` : *Kanal*- `read` : Lesender Zugriff
- `write` : Schreibender Zugriff
- `fcntl` : Zugriffsmode einstellen (beispielsweise blockierender Zugriff)

# Pipes

Embedded  
Systems -  
Einführung (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation  
Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübersicht

Rechtliche  
Aspekte

- Alle Prozesse haben standardmäßig drei *Dateien* offen:
  - Standard-Eingabe (stdin, Tastatur)
  - Standard-Ausgabe (stdout, Bildschirm)
  - Standard-Fehler (stderr, Bildschirm)
- Ausgaben eines Programmes lassen sich mit den Eingaben eines nächsten Programmes (über eine Pipe) verknüpfen.

## Beispiel auf der Kommandozeile

Die Ausgabe des Programms `ls` ist die Eingabe des Programms `wc`. Mit dieser Kommandofolge wird die Anzahl der Dateien im aktuellen Verzeichnis bestimmt.

```
user@host:~> ls | wc
```

# Dateitypen

- Normale Dateien (ordinary Files)
- Verzeichnisse (Ordner, Directories)
- Gerätedateien (Char- und Blockgeräte)
- Pipes
- Links (symbolic, hard links)

```
quade@ezs-mobil:/delete/mydir$ ls -l
insgesamt 12
prw-r--r-- 1 root root      0 Jan 25 13:51 appl_data
brw-r--r-- 1 root root 242, 3 Jan 25 13:52 flash-partition3
-rw-r--r-- 1 root root      6 Jan 25 13:50 foo.c
-rw-r--r-- 1 root root     16 Jan 25 13:50 Makefile
drwxr-xr-x 2 root root  4096 Jan 25 13:51 subdir
crw-r--r-- 1 root root 242, 1 Jan 25 13:50 temp-sensor
quade@ezs-mobil:/delete/mydir$
```

# Verzeichnisbaum (mounten und ummounten)

- Unix arbeitet auf einem einzigen Dateibaum.
- Der Dateibaum beginnt in der Wurzel (root) /
- Unix kennt keine Laufwerksbuchstaben.
- USB-Sticks, SD-Karten, Festplatten etc. werden über Verzeichnisse eingebunden
  - /media/quade/usbstick/
  - /mnt
- Einhängen über mount, aushängen über umount.
  - ① Parameter: Gerätedatei
  - ② Parameter: Verzeichnis

```
root@ezs-mobil:~# mount /dev/sdb2 /mnt
```

# Virtuelle Dateisysteme (1)

Embedded  
Systems -  
Einführung (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation

Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübersicht

Rechtliche  
Aspekte

- Klassisches Dateisystem: Verzeichnisse und Dateien sind auf einem Hintergrundspeicher (SSD, Flash, Festplatte) abgelegt.
- Virtuelles Dateisystem: Verzeichnisse und Dateien werden beim Zugriff *dynamisch* durch den Kernel erzeugt.

## Vertreter

- /proc
- /sys
- tmpfs
- initramfs

## Loop-Devices

- Ein *Blockgerät* ist im ersten Ansatz nichts anderes als eine Anzahl von Blöcken, die angesprochen (adressiert) werden können.
- Eine Datei ist auch eine Ansammlung von Blöcken.
- Eine Datei kann daher auch als *Blockgerät* betrachtet werden (*Loop-Device*).
- Dateien können als Geräte *eingehängt* (mount) werden.

## Overlay-Filesystem

- Per Overlay-Filesystem werden zwei *Verzeichnisse* transparent übereinander gelegt.
- Das *untere* Filesystem (*lower*) ist typischerweise *readonly*.
- Das *obere* Filesystem (*upper*) ist *writeable*.
- Ermöglichen den Readonly-Zugriff auf Dateisysteme.
- Einsatz:
  - ① Embedded Systems
  - ② Container (Docker)

# Rechtemodell

Embedded  
Systems -  
Einführung (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation

Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübersicht

Rechtliche  
Aspekte

Unix kennt zwei Arten von Benutzerkonten:

- ① Normaler Benutzer (UID!=Null)
- ② Superuser (Admin, Root, UID==0)

## Discretionary Access Control (DAC)

### Zugriffsklassen

- Owner
- Group
- World

### Zugriffsrechte

- lesen
- schreiben
- ausführen

# Rechtemodell (2)

- Für jede Klasse können die Zugriffsrechte angegeben werden.
- Programme erben die Rechte (ID) des startenden Programms.
- Per **S-Bit** können Programme die Rechte des Besitzers bekommen.
- **sudo** ermöglicht einen Kontenwechsel

```
quade@ezs-mobil:/delete/mydir$ sudo su  
root@ezs-mobil:/delete/mydir#
```

# Eltern-Kind Taskhierarchie

- Eltern starten Kindtasks als identische Kopien.
- Tasks können das Codesegment überschreiben und damit ein *anderes* Programm werden.
- Klassischerweise erwarten Elterntasks den Tod der Kindtasks.
- Tasks haben einen *Exitcode*.
- Tasks sind über ihre PID gekennzeichnet.
- Tasks können über das Senden eines Signals (`kill`) beendet werden.

## Section 5

### Kommandoübersicht

# Wichtige Kommandos (1)

## Shell-Kommandos

- cd, ls, pwd
- mkdir, rmdir
- rm
- cp
- mv
- echo
- cat
- ps, kill
- mount, umount
- touch

## System-Kommandos

- tail
  - tail -f /var/log/syslog
- grep
- dd
- cpio
- find
- uptime

# Wichtige Kommandos (2)

*Embedded  
Systems -  
Einführung (1)*

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation

Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübers.

Rechtliche  
Aspekte

## System-Kommandos

- uname
- chmod
- chown

## Netzwerk-Kommandos

- ifconfig
- route, ip
- ping
- netcat
- iptables
- ssh, scp
- rsync

# Verzeichnisstruktur (1)

- **/bin** Elementare Linux-Kommandos
- **/dev** Gerätedateien
- **/etc** Konfigurationsdateien
- **/proc** Virtuelles Dateisystem (Prozess-Infos)
- **/sys** Virtuelles Dateisystem (Hardware-Infos)
- **/sbin** Kommandos für die Systemverwaltung
- **/lib** Bibliotheken
- **/boot** Boot-Dateien (Kernel, Initramfs)

# Verzeichnisstruktur (2)

*Embedded  
Systems -  
Einführung (1)*

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation

Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübers.

Rechtliche  
Aspekte

- /home Daten der User (Homeverzeichnisse)
- /root Homeverzeichnis des Superusers
- /tmp Temporäre Daten
- /usr Anwendungsprogramme, Quellcodes
- /var Logdateien, Spooldateien, . . .

## Werkzeuge

- Editor (`vi`)
- Compiler (`gcc`, ruft Präprozessor, Compiler, Assembler, Linker auf)
- Debugger (`gdb`)
- Make (`make`)
- Source-Code-Verwaltung:
  - `git`

# Make make

- Werkzeug zur Generierung von Quellcode auf Basis von Abhängigkeiten
- Make generiert nur notwendige Komponenten
- Make überwacht zeitliche Abhängigkeiten
- Regeln und Abhängigkeiten werden in einem Makefile spezifiziert
- Make hat mehr als 1200 Regeln eingebaut

# Editor vi (vim)

Embedded  
Systems -  
Einführung (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Organisation

Theoretische  
Grundlagen

Betriebssysteme

Unix Sys-  
temkonzepte

Kommandoübers.

Rechtliche  
Aspekte

## Besonderheiten

- Befehlsmodus zusätzlich zum Einfügemodus
  - Der vi merkt sich den letzten (komplexen) Befehl und kann diesen wiederholen
- 10 Finger Editor
  - Cursor-Funktionen sind über Buchstaben-Tasten erreichbar
- Spaltenweises editieren
- Profi-Werkzeug – nur bedingt für Amateure geeignet
- Extrem leistungsfähig
- Steuerungslogik findet sich in vielen anderen Programmen (Stichwort Firefox vimperator)

## Besonderheiten

- **Dezentrales** Versionskontrollsystem
- Hohe Verbreitung (insbesondere im Open-Source Umfeld)
- Einfaches und kostenfreies Hosting von Open-Source Projekten auf Github (<http://www.github.com>)

## Wichtige Kommandos

- `git clone` – Repository kopieren
- `git pull` – Repository aktualisieren
- `git push` – Änderungen zurückspielen

## Section 6

### Rechtliche Aspekte

# Verzeichnis voller Lizenzen auf einem Sony Camcorder

## Embedded Systems - Einführung (1)

Jürgen Quade,  
Hochschule  
Niederrhein

## Organisation

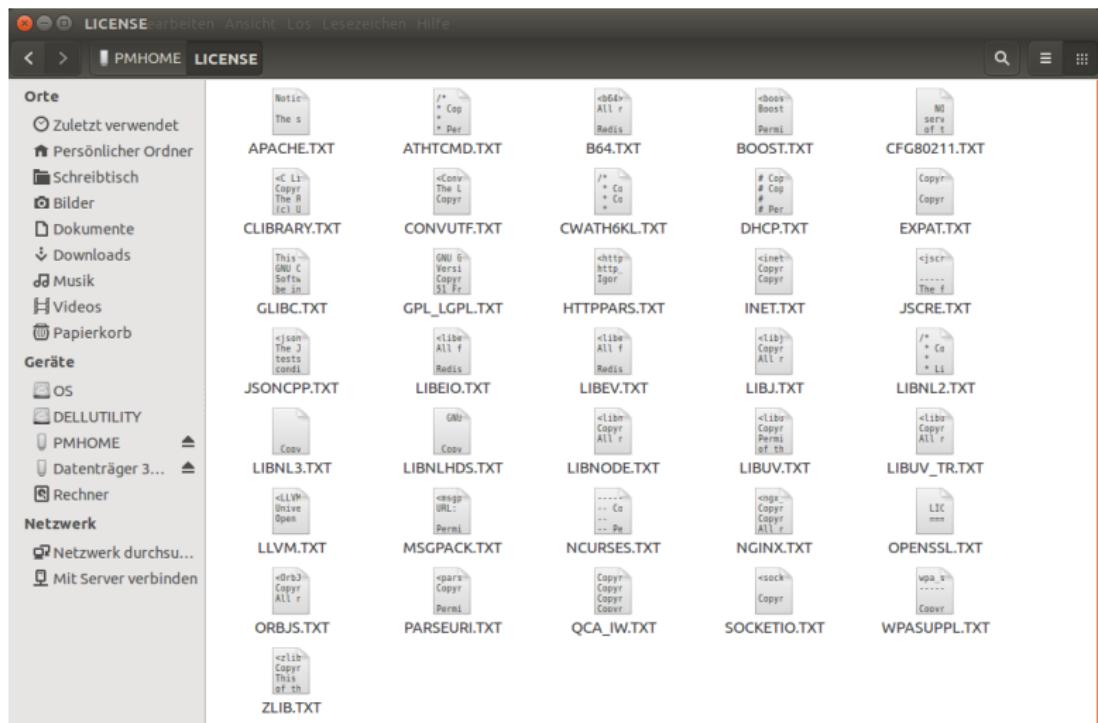
Theoretische  
Grundlagen

## Betriebssysteme

Unix Systemkonzepte

## Kommandoübersicht

## Rechtliche Aspekte



- Verwendete Lizenzen:

- GPL

- Modifikationen müssen öffentlich gemacht werden
    - Modifizierter Quellcode muss (auf Nachfrage) abgegeben werden
    - Kommerzielle Verwendung ist möglich und legal

- BSD

- Autoren müssen genannt werden
    - Modifikationen sind ohne Einschränkungen möglich
    - Kommerzielle Verwendung ist möglich und legal

*Embedded  
Systems -  
Linux  
handmade (1)*

Jürgen Quade,  
Hochschule  
Niederrhein

Linux  
handmade (im  
Emulator)

Kernel

# *Embedded Systems - Linux handmade (1)*

Jürgen Quade, Hochschule Niederrhein

19.03.2023

## 1 Linux handmade (im Emulator)

## 2 Kernel

*Embedded  
Systems -  
Linux  
handmade (1)*

Jürgen Quade,  
Hochschule  
Niederrhein

Linux  
handmade (im  
Emulator)  
Kernel

Prof. Dr.-Ing. Jürgen Quade  
Hochschule Niederrhein

*Embedded  
Systems -  
Linux  
handmade (1)*

Jürgen Quade,  
Hochschule  
Niederrhein

Linux  
handmade (im  
Emulator)

Kernel

## Section 1

Linux handmade (im Emulator)

# Übersicht

Embedded  
Systems -  
Linux  
handmade (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Linux  
handmade (im  
Emulator)

Kernel

## Ziele

- Linux „von der Pike auf“ kennen
- Embedded Linux komplett selbst aufbauen (im Emulator lauffähig)
- Die Konfigurationsoberfläche *menuconfig* kennen lernen
- Kernel konfigurieren und generieren

## Hintergrund

- Wir beginnen einfach: **keine** Cross-Entwicklung...
- Das System soll möglichst *schlank* werden; der Entwickler kennt noch sein gesamtes System!

# Arbeitsschritte

*Embedded  
Systems -  
Linux  
handmade (1)*

Jürgen Quade,  
Hochschule  
Niederrhein

Linux  
handmade (im  
Emulator)

Kernel

- ① Kernel bauen
- ② Userland bauen
- ③ Applikation hinzufügen (Webserver)
- ④ System im Emulator (qemu) testen

*Embedded  
Systems -  
Linux  
handmade (1)*

Jürgen Quade,  
Hochschule  
Niederrhein

Linux  
handmade (im  
Emulator)

Kernel

## Section 2

### Kernel

# Kernel bauen (1)

Embedded  
Systems -  
Linux  
handmade (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Linux  
handmade (im  
Emulator)

Kernel

## Schritte

- ① Vorbereitung (Werkzeuge installieren, Unterschriftenprobe)
- ② Download des Kernel-Quellcodes
- ③ Minimal-Konfiguration (`make allnoconfig`, `make menuconfig`)
- ④ Generieren

## Unterschriftenproben laden

```
user@host:~$ sudo apt install libncurses-dev qemu \
    qemu-system-x86 \
    uml-utilities libssl-dev libelf-dev
user@host:~$ mkdir -p embedded/qemu
user@host:~$ cd embedded/qemu
user@host:~/embedded/qemu$ gpg --locate-keys \
    torvalds@kernel.org gregkh@kernel.org
```

# Kernel bauen (2)

Embedded  
Systems -  
Linux  
handmade (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Linux  
handmade (im  
Emulator)

Kernel

## Download des aktuellen Kernels

<https://www.kernel.org>

```
wget https://www.kernel.org/pub/linux/kernel/v6.x/\\
      linux-6.2.7.tar.xz
```

```
wget https://www.kernel.org/pub/linux/kernel/v5.x/\\
      linux-6.2.7.tar.sign
```

# Kernel bauen (3)

Embedded  
Systems -  
Linux  
handmade (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Linux  
handmade (im  
Emulator)

Kernel

Wir starten von einer minimalen Konfiguration (alles ausgeschaltet, `make allnoconfig`) und konfigurieren nur das Benötigte.

Welche Komponenten werden **unbedingt** benötigt?

# Kernel bauen (4)

Embedded  
Systems -  
Linux  
handmade (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Linux  
handmade (im  
Emulator)

Kernel

- Festplattenzugriff
  - SCSI-Treiber
  - SATA-Treiber
  - Filesystem-Unterstützung (ext2, ext4)
- Treiber für die Konsolen-Ausgabe
- Treiber für Tastatur-Eingabe
- Unterstützung für Netzwerke
  - Protokolle (tcp/ip)
  - Treiber für eine Netzwerkkarte
  - Firewalling
- Ausführen von Programmen (executable file formats)
- Pseudo-Dateisysteme (proc, sys)

# Kernel bauen (5)

Embedded  
Systems -  
Linux  
handmade (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Linux  
handmade (im  
Emulator)

Kernel

## Auspicken und Konfigurieren

```
user@host:~/embedded/qemu$ xz -d linux-6.2.7.tar.xz
user@host:~/embedded/qemu$ gpg --verify linux-6.2.7.tar.sign
...
user@host:~/embedded/qemu$ tar xvf linux-6.2.7.tar
...
user@host:~/embedded/qemu$ cd linux-6.2.7
user@host:~/embedded/qemu/linux-6.2.7$ make allnoconfig
user@host:~/embedded/qemu/linux-6.2.7$ make menuconfig
```

# Hintergrund: make menuconfig (1)

Embedded  
Systems -  
Linux  
handmade (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Linux  
handmade (im  
Emulator)

Kernel

- Konfigurationssystem, liest Datei(en) Kconfig (oder auch Config.in) ein.
  - Konfiguration beschreibt Abhängigkeiten
  - Wahl wird in einer Variablen dokumentiert
- Ergebnis (aktuelle Konfiguration) findet sich in der versteckten Datei .config
- Hilfe zu einzelnen Optionen per “?”
- Auswahl durch Leerzeichen
- Wechsel der Optionen durch Tabulator
- Beim Verlassen wird in .config gespeichert
- Einmal gewählte Optionen bleiben erhalten

# Hintergrund: make menuconfig (2)

Embedded  
Systems -  
Linux  
handmade (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Linux  
handmade (im  
Emulator)

Kernel

.config - Linux/x86 3.18.11 Kernel Configuration  
> Search (ethernet) ——————  
Search Results

Symbol: ETHERNET [=y]  
Type : boolean  
Prompt: Ethernet driver support  
Location:  
    -> Device Drivers  
(1)   -> Network device support (NETDEVICES [=y])  
        Defined at drivers/net/ethernet/Kconfig:5  
        Depends on: NETDEVICES [=y] & NET [=y]  
        Selected by: SCSI\_CXGB3\_ISCSI [=m] && SCSI\_LOWLEVEL [=y] && SCSI [=y]

Symbol: OCTEON\_ETHERNET [=n]  
Type : tristate  
Prompt: Cavium Networks Octeon Ethernet support  
Location:  
    -> Device Drivers  
(2)   -> Staging drivers (STAGING [=y])  
        Defined at drivers/staging/octeon/Kconfig:1  
        Depends on: STAGING [=y] && CAVIUM\_OCTEON\_SOC && NETDEVICES [=y]  
        Selects: PHYLIB [=y] && MDIO\_OCTEON [=n]

Symbol: OCTEON\_MGMT\_ETHERNET [=n]

# Kernel bauen (6)

Embedded  
Systems -  
Linux  
handmade (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Linux  
handmade (im  
Emulator)

Kernel

## Optionen (make menuconfig) (1)

- [64-bit kernel]
- [Device Drivers][PCI support]
- [Executable File Formats][Kernelsupport for Elf-binaries]
- [Executable File Formats][Kernelsupport for scripts starting with #!]
- [Networking Support][Networking Options][Tcp/ip networking]
- [Networking Support][Networking Options][Network packet filtering framework]
- [Networking Support][Networking Options][802.1d Ethernet bridging]

# Kernel bauen (7)

Embedded  
Systems -  
Linux  
handmade (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Linux  
handmade (im  
Emulator)

Kernel

## Optionen (make menuconfig) (2)

- [Device Drivers][Network device Support][Universal TUN/TAP device driver support]
- [Device Drivers][Network device Support][Ethernet Driver Support][National Semiconductor 8390 devices][PCI NE2000 and clones support (see help)]
- [Device Drivers][SCSI device support][SCSI device support]
- [Device Drivers][SCSI device support][SCSI disk support]
- [Device Drivers][SCSI device support][SCSI generic support]
- [Device Drivers][Serial ATA und Parallel ATA drivers (libata)][Intel ESB,ICH,PIIX3,PIIX4 PATA/SATA support]
- [Device Drivers][Input device support][Keyboards]

# Kernel bauen (8)

Embedded  
Systems -  
Linux  
handmade (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Linux  
handmade (im  
Emulator)

Kernel

## Optionen (make menuconfig) (4)

- [File systems][Second extended fs support]
- [File systems][The Extended 4 (ext4) filesystem]
- [File systems][Pseudo Filesystems][tmpfs Virtual memory file system support (former shm fs)]

# Kernel bauen (9)

Embedded  
Systems -  
Linux  
handmade (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Linux  
handmade (im  
Emulator)

Kernel

## Generieren

```
user@host:~/embedded/qemu/linux-6.2.7$ time make -j6 bzImage
```

## Erläuterungen

- `time` -> mal sehen, wie lang das dauert...
- `-j6` -> Alle Prozessorkerne sollen arbeiten...

# Kernel bauen (10)

Embedded  
Systems -  
Linux  
handmade (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Linux  
handmade (im  
Emulator)

Kernel

```
...
LD      arch/x86/boot/compressed/vmlinux
ZOFFSET arch/x86/boot/zoffset.h
OBJCOPY arch/x86/boot/vmlinux.bin
AS      arch/x86/boot/header.o
LD      arch/x86/boot/setup.elf
OBJCOPY arch/x86/boot/setup.bin
BUILD   arch/x86/boot/bzImage
Kernel: arch/x86/boot/bzImage is ready (#1)
```

```
real    3m9,763s
user    19m17,482s
sys    1m35,590s
```

# Kernel bauen (11)

Embedded  
Systems -  
Linux  
handmade (1)

Jürgen Quade,  
Hochschule  
Niederrhein

Linux  
handmade (im  
Emulator)

Kernel

## Erster Test

Im Verzeichnis `~/embedded/qemu/linux-6.2.7/` wird das folgende Kommando aufgerufen:

```
qemu-system-x86_64 -kernel arch/x86/boot/bzImage
```

## Ergebnis

Der Kernel bootet, bleibt aber dann mit einem *Kernel-Panic* hängen:

```
...
---[ end Kernel panic - not syncing: VFS:  
Unable to mount root fs on unknown block(0,0) ]---
```

# *Embedded Systems - Linux handmade (2)*

Jürgen Quade, Hochschule Niederrhein

19.03.2023

## 1 Definitionen

## 2 Applikationen

## 3 Multicall-Binary

## 4 Systemebene

## 5 Realisierungsform

## 6 Praxis

Prof. Dr.-Ing. Jürgen Quade

Hochschule Niederrhein

# Übersicht

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

## Ziele

- Linux „von der Pike auf“ kennen
- Embedded Linux komplett selbst aufbauen (im Emulator lauffähig)
- Mit dem Multicall-Binary Busybox umgehen können
- Userland bauen
- Applikation integrieren

## Hintergrund

- Wir beginnen einfach: **keine** Cross-Entwicklung...
- Das System soll möglichst *schlank* werden; der Entwickler kennt noch sein gesamtes System!

## Section 1

### Definitionen

# Aufgaben

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

Das so genannte Userland wird häufig auch als Rootfilesystem bezeichnet. Es enthält

- alle für den Betrieb notwendigen Dateien
  - Systemdateien
  - Applikationsprogramme
  - Konfigurationsdateien

Im Userland sind zwei Arten von Software zu unterscheiden

- ① Systemprogramme (Dienste)
- ② Applikationen (die die eigentliche Funktion des eingebetteten Systems bestimmen)

## Section 2

### Applikationen

# Einschränkungen

In eingebetteten Systemen gibt es einige Einschränkungen für Applikationen

- eingeschränkte Bibliotheksauswahl
  - weniger Funktionen
  - teilweise nicht threadsafe
- häufig gibt es kein Framework für gemeinsam genutzte Bibliotheken (shared libs, DLLs)
- es stehen nur eingeschränkte Ressourcen (Speicher, Rechenzeit) zur Verfügung

Lösung im Bereich eingebetteter Systeme

Verwendung von Multicall-Binaries (busybox)

## Section 3

### Multicall-Binary

# Busybox

- Einzelne Applikation, die funktional mehrere andere Applikationen ersetzt.
- Das Multicall-Binary findet sich unter verschiedenen Namen (dank Links) auf dem Filesystem
- Spart Hintergrundspeicher und Hauptspeicher
- Bekanntester Vertreter: Busybox

## Funktionalitäten

- Shell (ash)
- Shellkommandos (ls, cd, pwd, ...)
- Editor (vi)
- Programme zur Netzwerkkonfiguration
- Standardprogramme wie Webserver
- ...

# Multicall-Binary (Fortsetzung)

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

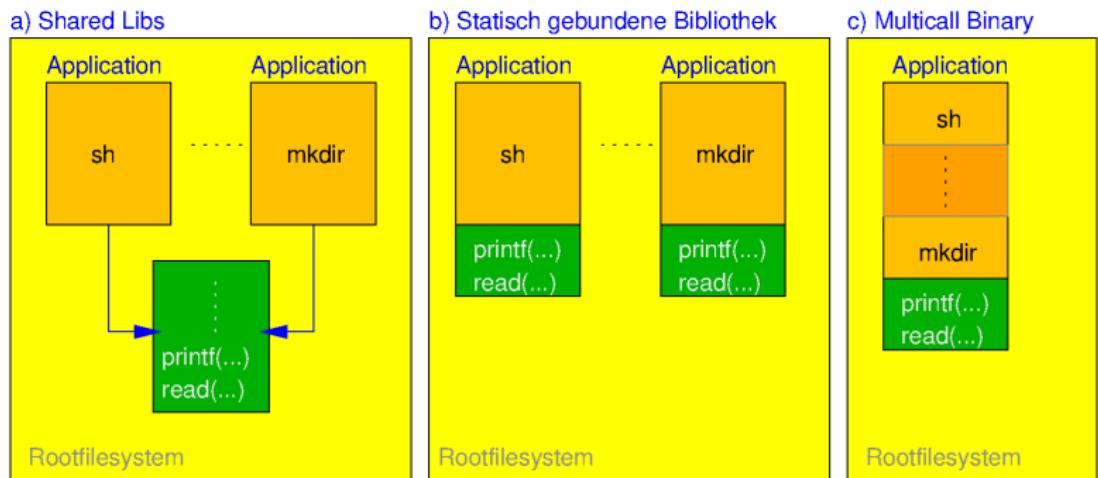
Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration



# Technische Realisierung eines Multicall-Binaries

## ACHTUNG: Prüfungsrelevant!

- Zum Executable (ausführbares Programm) werden symbolische Links unter den Namen der unterschiedlichen Kommandos angelegt
- Damit kann ein einzelnes Busybox mit unterschiedlichen Namen aufgerufen werden
- Innerhalb des Quellcodes kann anhand des Parameters `argv[0]` evaluiert werden, unter welchem Namen Busbox aufgerufen wurde.
- Anhand des Namens wird die jeweilige Funktionalität (in einem Unterprogramm) ausgeführt

# Technische Realisierung eines Multicall-Binaries (Fortsetzung)

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

```
int main( int argc, char **argv, char **envp )
{
    ...
    if (strcmp(argv[0], "ls") == 0)
        do_ls( argv );
    if (strcmp(argv[0], "cp") == 0)
        do_cp( argv );
    if (strcmp(argv[0], "pwd") == 0)
        do_pwd( argv );
    ...
}
```

## Section 4

### Systemebene

# Komponenten

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

## ① Hilfsprogramme

- Setzen des Systemnamens
- Setzen der IP-Adresse
- Shell

## ② Konfigurationsdateien

- Systemname (/etc/hostname)
- IP-Adressen (/etc/networking/interfaces)
- zu startende Applikationen (/etc/inittab)
- Userdaten (Loginname, Passwort, /etc/passwd)

## ③ Gerätedateien

- Verbindung zwischen Programmen und Treibern

## ④ Strukturierung

- Dateibaum, Ordnerstruktur

# Ordnerstruktur Unix

Verzeichnis	Bedeutung
/	Root-Verzeichnis
/bin	User-Kommandos
/sbin	Systemprogramme
/lib	Bibliotheken
/dev	Gerätedateien
/usr	Anwendungsprogramme
/etc	Konfigurationsdateien
/tmp	Temporäre Daten (werden beim Reboot gelöscht)
/var	Log- und Spooldateien (z.B. Druckerjobs)
/proc	Einhängepunkt für das virtuelle Proc-Filesystem
/sys	Einhängepunkt für das virtuelle Sys-Filesystem

# Ordnerstruktur Unix (Fortsetzung)

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

Verzeichnis	Bedeutung
/root	Homeverzeichnis des Superuser
/home	Ausgangsordner für Homeverzeichnisse

# Gerätedateien

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

Stellen die Verbindung zwischen Applikationen und Hardware her. Eine Gerätedatei repräsentiert einen **Gerätetreiber** (der auf die Hardware zugreift).

## Notwendige Gerätedateien

Name	Bedeutung (Peripherie)
/dev/console	Terminalfenster und Tastatur
/dev/zero	Lesender Zugriff gibt Nullen zurück
/dev/null	Schreibender Zugriff verschluckt Daten (Mülleimer)

# Gerätedateien (Fortsetzung)

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

- Applikationen greifen auf Peripherie über (Datei-)Namen zu.
- Treiber besitzen eine (eindeutige) Gerätenummer, bestehend aus
  - Major-Nummer
  - Minor-Nummer
- Gerätummern werden über das Kommando `mknod` mit den Gerätedateien assoziiert.

## Section 5

### Realisierungsform

# Realisierungsformen

*Embedded  
Systems -  
Linux  
handmade (2)*

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsfor

Praxis

Qemu

Webserver-  
Integration

- Normales Dateisystem
  - Inhalte befinden sich auf einem Hintergrundspeicher
- Initramfs
  - Inhalte befinden sich im Hauptspeicher

## Section 6

### Praxis

# Hintergrund: Formatieren

- Um in Partitionen Verzeichnisse anlegen und Dateien ablegen zu können werden Organisations-Datenstrukturen benötigt.
- Hierfür haben sich unterschiedlichste Varianten ausgeprägt:
  - VFAT (Microsoft alt, Fotoapparate etc.)
  - NTFS (Microsoft Windows)
  - ext4 (Linux)
- Das initiale Schreiben der Organisationsdaten auf die Partition wird *formatieren* genannt.
- Zum Formatieren stehen die Programme:
  - mkfs.vfat,
  - mkfs.ext2 und
  - mkfs.ext4 zur Verfügung.

# Rootfilesystem (1)

*Embedded  
Systems -  
Linux  
handmade (2)*

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

Das Rootfilesystem (=Userland) wird in Form eines Image zur Verfügung gestellt. Das Image wird später auf das Flash kopiert oder vom Boot-Loader als Ram-Filesystem in den Hauptspeicher geladen.

# Rootfilesystem (2)

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

## Schritte

- ① Vorbereitungen (Verzeichnisse erstellen)
- ② Imagedatei anlegen (dd)
- ③ Imagedatei formatieren (`mkfs.ext2`)
- ④ Busybox downloaden, Signatur überprüfen, auspacken
- ⑤ Busybox konfigurieren (`make allnoconfig`, `make menuconfig`)
- ⑥ Busybox compilieren und lokal (vor-)installieren
- ⑦ Imagedatei mounten (um **im** Image zu arbeiten)
- ⑧ Kommandos **in** das Image kopieren (`rsync`)
- ⑨ Targetverzeichnisse anlegen (`mkdir`)
- ⑩ Gerätedateien anlegen (`mknod`)
- ⑪ Besitzverhältnisse einstellen (`chown`)
- ⑫ Imagedatei aushängen (`umount`)

# Rootfilesystem (3)

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

## 1. Vorbereitungen

```
user@host:~/embedded$ # Unterschriftenprobe herunterladen
user@host:~/embedded$ gpg --keyserver hkp://keyserver.ubuntu.com
                  --recv-keys ACC9965B
user@host:~/embedded$ # Verzeichnis anlegen
user@host:~/embedded$ mkdir -p qemu/userland
user@host:~/embedded$ # Mount für die Imagedatei
user@host:~/embedded$ mkdir -p qemu/userland/loop
user@host:~/embedded$ cd qemu/userland
```

# Rootfilesystem (4)

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

Größe: 16 MByte

## 2. Imagedatei rootfs.img anlegen

```
user@host:~/embedded$ cd qemu/userland
user@host:~/embedded/qemu/userland$ \
    dd if=/dev/zero of=rootfs.img bs=1k count=16384
```

## 3. Imagedatei formatieren

```
user@host:~/embedded/qemu/userland$ \
    mkfs.ext4 -F rootfs.img
```

# Rootfilesystem (5)

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

## 4. Busybox downloaden, Signatur überprüfen, auspacken

```
user@host:~/embedded/qemu/userland$ # Download
user@host:~/embedded/qemu/userland$ wget \
    https://busybox.net/downloads/busybox-1.34.1.tar.bz2
user@host:~/embedded/qemu/userland$ wget \
    https://busybox.net/downloads/busybox-1.34.1.tar.bz2.sig
user@host:~/embedded/qemu/userland$ # Signatur überprüfen
user@host:~/embedded/qemu/userland$ \
    gpg --verify busybox-1.34.1.tar.bz2.sig
user@host:~/embedded/qemu/userland$ # Auspacken
user@host:~/embedded/qemu/userland$ \
    tar xvf busybox-1.34.1.tar.bz2
```

# Rootfilesystem (6)

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

Wir gehen bei Busybox wieder von einer minimalen Konfiguration aus.

## 5. Busybox konfigurieren (1)

```
user@host:~/embedded/qemu/userland$ cd busybox-1.34.1
user@host:~/embedded/qemu/userland/busybox-1.34.1$ \
    make allnoconfig
user@host:~/embedded/qemu/userland/busybox-1.34.1$ \
    make menuconfig
... # Optionen auf der nächsten Folie
```

# Rootfilesystem (7)

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

## 5. Busybox konfigurieren (2)

Folgende Menüpunkte müssen ausgewählt werden:

- statisch gelinkte Variante
  - [Settings][Build Options][Build BusyBox as a static binary (no shared libs)]
- Kommandozeilen editieren zulassen
  - [Settings][Command line editing]
- Shell ash
  - [Shells][ash]
  - [Shells][Choose which shell is aliased to 'sh' name (ash)]
  - [Shells][Choose which shell is aliased to 'bash' name (ash)]
  - [Shells][User internal glob() implementation]

# Rootfilesystem (8)

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

Konfigurieren Sie nur die Kommandos, die Sie kennen und brauchen.

## 5. Busybox konfigurieren (3)

- Kommandos:

- ls
- mkdir
- uname
- mount
- ps
- kill
- echo
- cat
- pwd
- vi

# Rootfilesystem (9)

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen  
Applikationen  
Multicall-  
Binary  
Systemebene  
Realisierungsform  
**Praxis**  
Qemu  
Webserver-  
Integration

## 6. Busybox compilieren und lokal (vor-)installieren

```
user@host:~/embedded/qemu/userland/busybox-1.34.1$ \
    time make
user@host:~/embedded/qemu/userland/busybox-1.34.1$ \
    make install
```

# Rootfilesystem (10)

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen  
Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

## 7. Imagedatei mounten (um **im** Image zu arbeiten)

```
user@host:~/embedded/qemu/userland/busybox-1.34.1$ cd ../../  
user@host:~/embedded/qemu$ sudo \  
mount -o loop userland/rootfs.img userland/loop
```

## 8. Kommandos **in** das Image kopieren (rsync)

```
user@host:~/embedded/qemu$ sudo \  
rsync -a userland/busybox-1.34.1/_install/ \  
userland/loop
```

# Rootfilesystem (11)

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen  
Applikationen  
Multicall-  
Binary  
Systemebene  
Realisierungsform

Praxis

Qemu

Webserver-  
Integration

## 9. Targetverzeichnisse anlegen (mkdir)

```
user@host:~/embedded/qemu/userland$ mkdir loop/dev
user@host:~/embedded/qemu/userland$ mkdir loop/etc
user@host:~/embedded/qemu/userland$ mkdir loop/proc
user@host:~/embedded/qemu/userland$ mkdir loop/sys
```

# Rootfilesystem (11)

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen  
Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

- Gerätedateien stellen die Verbindung zum Treiber her
- Jeder Treiber hat eine (mehr oder minder eindeutige) Nummer (Majornummer)
- Treiber bekommen beim Zugriff einen numerischen Parameter (Minornummer) übergeben
- Gerätedateien haben einen Typ (c=Character-Devices, b=Block-Devices)

## 10. Gerätedateien anlegen (`mknod`)

```
user@host:~/embedded/qemu/userland$ sudo \  
    mknod loop/dev/null c 1 3  
user@host:~/embedded/qemu/userland$ sudo \  
    mknod loop/dev/tty1 c 4 1  
user@host:~/embedded/qemu/userland$ sudo \  
    mknod loop/dev/console c 5 1
```

# Rootfilesystem (12)

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

## 11. Besitzverhältnisse einstellen (chown)

```
user@host:~/embedded/qemu/userland$ sudo \  
    chown -R root.root loop/*
```

## 12. Imagedatei aushängen (umount)

```
user@host:~/embedded/qemu/userland$ sudo umount loop
```

## Section 7

Qemu

# Allgemeines

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

- Virtuelle Maschine (Emulator), die (der) unterschiedliche Prozessorarchitekturen emulieren kann:
  - x86, AMD64 und x86-64, PowerPC, ARM (32 + 64 Bit), Alpha, CRIS, LatticeMico32, m68k (Coldfire), MicroBlaze, MIPS, Moxie, SH-4, S/390, Sparc32/64, TriCore, OpenRISC, Unicore und Xtensa
- Qemu emuliert komplette Systeme (CPU plus sonstige Hardware, beispielsweise Netzwerkkarten)
- Ist auf unterschiedlichen Plattformen lauffähig:
  - Linux, Windows, FreeBSD, NetBSD, OpenBSD, OpenSolaris, Mac OS X

# Test im Emulator (1)

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen  
Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

*Jetzt wird es spannend...*

```
user@host:~/embedded/qemu$ qemu-system-x86_64 \  
-hda userland/rootfs.img \  
-kernel linux-5.14.14/arch/x86/boot/bzImage \  
-append "root=/dev/sda rw init=/bin/ash"
```

## Parameter

- hda: Hintergrundspeicher
- kernel: Betriebssystemkern
- append: Kernelparameter (bootet hier in eine Shell)

# Test im Emulator (2)

```
QEMU plugin SSH (type=Protokoll) registered.  
Bridge firewalls registered  
input: AT Translated Set 2 keyboard as /devices/platform/i8042/serio0/input/input1  
ata1.00: ATA-7: QEMU HARDDISK, 1.0, max UDMA/100  
ata1.00: 16384 sectors, multi 16: LBA48  
ata2.00: ATAPI: QEMU DVD-ROM, 1.0, max UDMA/100  
ata2.00: configured for MWDMA2  
ata1.00: configured for MWDMA2  
scsi 0:0:0:0: Direct-Access      ATA      QEMU HARDDISK    1.0 PQ: 0 ANSI: 5  
sd 0:0:0:0: Attached scsi generic sg0 type 0  
sd 0:0:0:0: [sda] 16384 512-byte logical blocks: (8.38 MB/8.00 MiB)  
sd 0:0:0:0: [sda] Write Protect is off  
sd 0:0:0:0: [sda] Write cache: disabled, read cache: enabled, doesn't support DP  
0 or FUA  
sda: unknown partition table  
scsi 1:0:0:0: CD-ROM           QEMU      QEMU DVD-ROM    1.0 PQ: 0 ANSI: 5  
scsi 1:0:0:0: Attached scsi generic sg1 type 5  
sd 0:0:0:0: [sda] Attached SCSI disk  
EXT2-fs (sda): warning: mounting unchecked fs, running e2fsck is recommended  
UFS: Mounted root (ext2 filesystem) on device 8:0.  
Freeing unused kernel memory: 604k freed  
/bin/ash: can't access tty; job control turned off  
# Switching to clocksource tsc
```

Embedded Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

# Test im Emulator (3)

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

- Erkunden Sie Ihr System:
  - Versuchen Sie das Kommando `ps`
  - Mounten Sie die Filesysteme `proc` und `sys`
  - Funktioniert der Editor `vi`?
- Gibt es eine Benutzerverwaltung?
- Welche Funktionen fehlen Ihnen?

## Section 8

### Webserver-Integration

# Aufgabe

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

- ① Netzwerkunterstützung in Qemu aktivieren
- ② Webserver in Busybox integrieren
- ③ Konfigurationsdateien erstellen
- ④ Webserver-Dateien (Inhalt) erstellen
- ⑤ Test

# Netzwerkunterstützung in Qemu

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

- Skript `start_el.sh` vom Moodle-Server laden und in das Verzeichnis `~/embedded/qemu` ablegen
- Skript ausführbar machen (`chmod +x start_el.sh`)
- Qemu wird danach über dieses Skript gestartet.

# Image um Webserver erweitern

- Webserver in Busybox auswählen,
- Busybox generieren und installieren
- Image mounten
- Neue Busybox-Version ins Image kopieren

## Kommandos

```
user@host:~/embedded/qemu$ cd userland/busybox-1.34.1
user@host:~/embedded/qemu/userland/busybox-1.34.1$ \
    make menuconfig
# httpd auswählen
user@host:~/embedded/qemu/userland/busybox-1.34.1$ \
    make && make install
user@host:~/embedded/qemu/userland/busybox-1.34.1$ cd ..
user@host:~/embedded/qemu/userland$ sudo \
    mount -o loop rootfs.img loop
user@host:~/embedded/qemu$ sudo \
    rsync -a busybox-1.34.1/_install/ loop
```

# Webserverdateien erstellen (1)

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

- ① /etc/httpd.conf
- ② /var/www/index.html
- ③ /var/www/ps.cgi

```
user@host:~/embedded/qemu/userland$ vi loop/etc/httpd.conf
# Inhalt der Datei siehe nächsten Kasten
```

## httpd.conf

```
H:/var/www
*.cgi:/bin/sh
```

# Webserverdateien erstellen (2)

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

```
user@host:~/embedded/qemu/userland$ mkdir -p loop/var/www
user@host:~/embedded/qemu/userland$ vi loop/var/www/index.html
# Inhalt der Datei siehe nächsten Kasten
```

index.html

```
<html>
<h1>Hi, I am your embedded system!</h1>
</html>
```

# Webserverdateien erstellen (3)

```
user@host:~/embedded/qemu/userland$ vi loop/var/www/ps.cgi
# Inhalt der Datei siehe nächsten Kasten
...
user@host:~/embedded/qemu/userland$ chmod +x loop/var/www/ps.cgi
```

## ps.cgi

```
#!/bin/sh
echo "Content-type: text/html"
echo ""
echo ""
echo "<html>"
echo "<pre>"
ps
echo "</pre>"
echo "</html>"
echo ""
echo ""
```

# Webserver - System starten

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen  
Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

- ① Image aushängen (`umount`)!
- ② System per `./start_el.sh` starten.
- ③ System konfigurieren (Achtung: englische Tastatur)

```
user@host:~/embedded/qemu/userland$ sudo umount loop
user@host:~/embedded/qemu/userland$ cd ..
user@host:~/embedded/qemu$ ./start_el.sh
```

# Webserver - System konfigurieren

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

Auf dem System ist

- ① das Netzwerk zu konfigurieren (IP: 10.69.0.99)
- ② der Webserver zu starten

```
mount -t proc none /proc
ifconfig eth0 10.69.0.99
httpd
```

# Webserver - Test

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

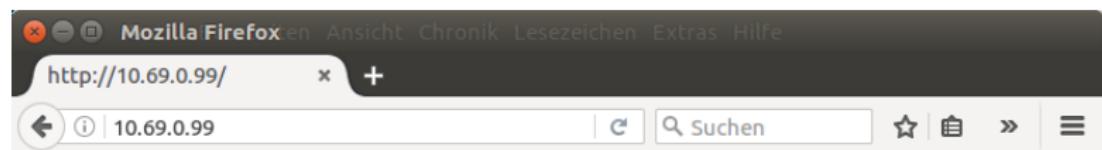
Realisierungsform

Praxis

Qemu

Webserver-  
Integration

Geben Sie auf dem Entwicklungsrechner in einem Browser die URL <http://10.69.0.1> ein.



**Hi, I am your embedded system**

# Webserver - Test

Embedded  
Systems -  
Linux  
handmade (2)

Jürgen Quade,  
Hochschule  
Niederrhein

Definitionen

Applikationen

Multicall-  
Binary

Systemebene

Realisierungsform

Praxis

Qemu

Webserver-  
Integration

PID	USER	VSZ	STAT	COMMAND
1	0	1876	S	/bin/ash
2	0	0	SW	[kthreadd]
3	0	0	SW	[ksoftirqd/0]
4	0	0	SW	[kworker/0:0]
5	0	0	SW<	[kworker/0:0H]
6	0	0	SW	[kworker/u2:0]
7	0	0	SW<	[lru-add-drain]
8	0	0	SW	[oom_reaper]
9	0	0	SW<	[writeback]
10	0	0	SW<	[crypto]
11	0	0	SW	[bioset]
12	0	0	SW<	[kblockd]
13	0	0	SW<	[ata_sff]
14	0	0	SW	[kswapd0]
15	0	0	SW	[kworker/0:1]
33	0	0	SW<	[acpi_thermal_pm]
34	0	0	SW	[scsi_eh 0]
35	0	0	SW	[kworker/u2:1]
36	0	0	SW<	[scsi_tm 0]
37	0	0	SW	[scsi_eh 1]
38	0	0	SW<	[scsi_tm 1]
39	0	0	SW<	[ipv6_addrconf]
40	0	0	SW	[kworker/u2:2]
41	0	0	SW	[kworker/0:2]
42	0	0	SW<	[bioset]
43	0	0	SW	[kworker/u2:3]
44	0	0	SW<	[bioset]
45	0	0	SW<	[kworker/0:1H]
46	0	0	SW<	[ext4-rsv-conver]
51	0	1876	S	httpd
72	0	1876	S	httpd
73	0	1876	S	/bin/sh ps.cgi
79	0	1876	R	ps

# *Embedded Systems - Netzwerkboot*

Jürgen Quade, Hochschule Niederrhein

4.04.2022

## 1 Netzwerkboot

## 2 Hintergrund

## 3 Host-Software

## 4 Kopplung Pi-Host-Ethernet

## 5 Raspberry Pi

## 6 Fehlersuche

7

## Netzwerkkonfig Standalone-Netz

*Embedded  
Systems -  
Netzwerkboot*

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

# Prof. Dr.-Ing. Jürgen Quade

## Hochschule Niederrhein

## Section 1

### Netzwerkboot

# Ziele

*Embedded  
Systems -  
Netzwerkboot*

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

## Optimierung der Entwicklungsumgebung

### Netzwerkboot

## Section 2

### Hintergrund

# Ausgangslage Entwicklung eingebetteter Systeme

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

## Host-Target-Entwicklung:

- Ein Entwicklungsrechner (Host) generiert die Software für das Zielsystem (Target und Targetsoftware).

## Vorteil

- Komfortablere und schnellere Entwicklungsumgebung auf dem Host

## Nachteil

- Häufig Cross-Development notwendig
- (Zeitaufwändiger) Transfer der Targetsoftware auf das Zielsystem

# Grundlagen

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

## Netzwerkboot

- Kurze Turnaround-Zeiten lassen sich durch Netzwerkboot realisieren.
- Der Bootloader holt sich den Kernel über tftp von einem tftp-Server.
- Klassisch wird auch das Userland als Initramfs (Art Ramdisk) vom tftp-Server geladen.
- Falls die Firmware *Netzwerkboot* nicht unterstützt, kann dieses über den Bootloader „Das U-Boot“ realisiert werden.

## Benötigte Komponenten

- ① tftp-Server auf dem Host
- ② Unterstützung auf dem Client

# Grundlagen

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

## Netzwerktechnik (1)

- Das Internet verbindet primär Netze (und erst sekundär Rechner/Hosts).
- Netzwerke haben eine Netzadresse.
- Rechner haben eine Rechneradresse (auch Hostadresse genannt).
- Damit Rechner eindeutig angesprochen werden können, benötigt man eine Netz- und eine Hostadresse.
- Die Kombination aus Netz- und Hostadresse ist die IP-Adresse.
- Klassische IP-Adressen bestehen aus 4 Byte, die dezimal geschrieben, durch Punkt getrennt hintereinander geschrieben werden.
- 192.168.42.42

# Grundlagen

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

## Netzwerktechnik (2)

- Die Grenze zwischen Netz- und Hostadresse ist konfigurierbar und wird über eine so genannte Netzmase angegeben.
- Hosts, die in mehreren Netzen beheimatet sind, können Daten von einem ins andere Netz transferieren (Gateway, Router).
- Hosts schicken Pakete (Nachrichten), die nicht im eigenen Netz sind, zur Weiterleitung an den Router bzw. an das Gateway.

# Grundlagen

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

## Netzwerktechnik (3)

- Zur Konfiguration einer IP-Adresse dient das Programm **ifconfig**
    - Parameter 1: Interface (z.B. eth0 oder wlan0)
    - Parameter 2: IP-Adresse (z.B. 192.168.95.97)
- ```
ifconfig eth0 192.168.95.97
```

## ip

Moderne Systeme verwenden alternativ das Werkzeug ip  
`ip address add 192.168.95.97/24 dev eth0`

# Grundlagen

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

## Netzwerktechnik (4)

- Mit ping lässt sich die Erreichbarkeit eines Hosts überprüfen
  - Parameter 1: IP-Adresse (z.B. 192.168.95.97)

```
quade@felicia:~>ping 192.168.69.1
PING 192.168.69.1 (192.168.69.1) 56(84) bytes of data.
64 bytes from 192.168.69.1: icmp_seq=1 ttl=64 time=0.524 ms
64 bytes from 192.168.69.1: icmp_seq=2 ttl=64 time=0.503 ms
64 bytes from 192.168.69.1: icmp_seq=3 ttl=64 time=0.500 ms
^C
--- 192.168.69.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.500/0.509/0.524/0.010 ms
quade@felicia:~>
```

# Grundlagen

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

## Netzwerktechnik (5)

- Das Kommando `ip route` (bzw. `route`) dient zum Anlegen/Verändern/Anzeigen einer so genannten *Routingtabelle*.
- Die Routingtabelle legt fest, über welches Interface (und Gateways) Nachrichten geschickt werden.

```
quade@rzs:~$ route -n
Kernel-IP-Routentabelle
```

| Ziel          | Router         | Genmask       | Flags | Metric | Ref |
|---------------|----------------|---------------|-------|--------|-----|
| 0.0.0.0       | 194.94.121.254 | 0.0.0.0       | UG    | 0      | 0   |
| 169.254.0.0   | 0.0.0.0        | 255.255.0.0   | U     | 1000   | 0   |
| 192.168.50.0  | 0.0.0.0        | 255.255.255.0 | U     | 0      | 0   |
| 192.168.87.0  | 0.0.0.0        | 255.255.255.0 | U     | 0      | 0   |
| 194.94.121.0  | 0.0.0.0        | 255.255.255.0 | U     | 0      | 0   |
| quade@rzs:~\$ |                |               |       |        |     |

# Grundlagen

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

## Netzwerktechnik (6)

Das Ziel 0.0.0.0 in der Routing-Tabelle stellt das *Default-Gateway* dar. Hierhin werden alle Pakete weiter geleitet, für die es bis keinen anderen Eintrag in der Routing-Tabelle gibt. Im obigen Beispiel werden sämtliche Pakete, die nicht für das Netz 192.169.50.0, 192.168.87.0 oder 194.94.121.0 sind an den Router 194.94.121.254 gesendet. Der leitet die Pakete dann weiter.

# Grundlagen Bootvorgang

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

## Raspberry Pi 4

Die Firmware des Raspberry Pi 4 unterstützt von Haus aus Netzwerkboot. Allerdings muss dazu

- ① die jeweils aktuellste Firmware installiert sein
- ② Netzwerkboot konfiguriert werden
  - Bootreihenfolge
  - Netzwerkparameter (IP-Adresse, Gateway-IP)
  - tftp-Server

## Section 3

### Host-Software

# Host: Installation tftpd-Server

- Als tftp-Server bietet sich tftpd-hpa an. Diesen gibt es als Ubuntu-Paket.
- ACHTUNG: es gibt auch ein Paket namens tftp-hpa, das einen Client realisiert.
- Das Datenverzeichnis des tftp-Servers liegt unter /srv/tftp/.

```
user@host:~# apt-get install tftpd-hpa
```

# Host: Logging vom TFTP-Server aktivieren

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkonfig  
Standalone-  
Netz

Dazu wird die Datei /etc/default/tftpd-hpa editiert:

```
sudo vim /etc/default/tftpd-hpa
TFTP_OPTIONS="--secure --verbose"
```

Zur Beobachtung der Ausgaben geben Sie nachfolgendes Kommando in ein Terminalfenster ein:

```
service tftpd-hpa restart
tail -f /var/log/syslog | grep tftp
```

Anhand der Ausgaben ist zu beobachten, wie der Raspberry Pi beim Booten die einzelnen Dateien anfordert.

# Host: Installation tftpd-Server

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

## Hinweis VirtualBox

- Um den tftpd-Server in einer virtualisierten Umgebung nutzen zu können, muss das Netzwerkinterface als **Netzwerkbrücke** eingerichtet sein, nicht als NAT!
- Separates Netz: **Falls** zum Raspberry ein lokales ethernet-basiertes Netz verwendet wird, muss das Netzwerk in der virtuellen Machine auf dem Ethernet-Interface noch konfiguriert werden. Die Netzwerk-Konfiguration findet sich am Ende des Foliensatzes.

# HOST: Zum Test: Orginale Bootfile installieren

- Die orginalen Bootdateien werden über's Internet geladen und im Verzeichnis /srv/tftp/ ausgepackt.
- Zugriffsrechte müssen noch angepasst werden (sonst kommen ggf. noch "Permission denied or File Not Found" Fehlermeldungen)

```
wget https://github.com/raspberrypi/firmware/archive/master.zip
unzip master.zip
cp -r firmware-master/boot/* /srv/tftp/
chmod -R 755 /srv/tftp/
```

# HOST: Inhalt von /srv/tftp (nach dem Auspacken)

```
root@ezs-cock-a:/srv/tftp# ls
backup                                bcm2711-rpi-400.dtb    kernel71.img
bcm2708-rpi-b.dtb                      bcm2711-rpi-4-b.dtb    kernel8.img
bcm2708-rpi-b-plus.dtb                 bcm2711-rpi-cm4.dtb   kernel.img
bcm2708-rpi-b-rev1.dtb                 bootcode.bin           LICENCE.broadcom
bcm2708-rpi-cm.dtb                     COPYING.linux          overlays
bcm2708-rpi-zero.dtb                   fixup4cd.dat         start4cd.elf
bcm2708-rpi-zero-w.dtb                 fixup4.dat           start4db.elf
bcm2709-rpi-2-b.dtb                   fixup4db.dat        start4.elf
bcm2710-rpi-2-b.dtb                   fixup4x.dat         start4x.elf
bcm2710-rpi-3-b.dtb                   fixup_cd.dat        start_cd.elf
bcm2710-rpi-3-b-plus.dtb              fixup.dat           start_db.elf
bcm2710-rpi-cm3.dtb                   fixup_db.dat        start.elf
bcm2710-rpi-zero-2.dtb                fixup_x.dat         start_x.elf
bcm2710-rpi-zero-2-w.dtb              kernel7.img
```

# HOST: Netzwerkboot

- config.txt vom Lernserver ins tftp-Verzeichnis kopieren

## Relevanter Ausschnitt der Datei config.txt

```
...
[pi4]
# Enable DRM VC4 V3D driver on top of the dispmanx display stack
dtoverlay=vc4-fkms-v3d
max_framebuffers=2

[all]
enable_uart=1
dtoverlay=disable-bt
kernel=kernel7l.img
initramfs rootfs.cpio followkernel
```

# Serielles Interface

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

Zur Kommunikation mit einem headless Raspberry Pi nutzen wir die serielle Schnittstelle.

- Host: Auf dem Host ist ein Terminalprogramm (z.B. minicom) zu installieren.
- Target: Der Raspberry Pi muss für die Nutzung der seriellen Schnittstelle konfiguriert werden.

## Konfiguration Raspberry Pi 4

- Standardmäßig ist das serielle Interface für Bluetooth reserviert.
- Damit der RPi4 über die serielle Schnittstelle mit dem Entwicklungsrechner kommuniziert, muss zum einen die Konfigurationsdatei des Bootloaders config.txt angepasst werden.
  - Option disable-bt (bereits gesetzt)
- Konfiguration der Kernel-Startparameter (kommt später)

# Host: Installation minicom

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

- `apt install minicom`
- Konfiguration
  - Device: `/dev/ttyUSB0`
  - Hardware-Flow-Control: No

# Host: Konfiguration minicom

- Superuser werden: `sudo su`
- Minicom starten: `minicom`
- Per `<CTRL><A>` und dann `<Z>` ins Menu wechseln
- `<O>` eingeben (Options)
- `<Einstellungen zum seriellen Anschluss>`
- **Serieller Anschluss** muss auf `/dev/ttyUSB0` stehen (oder die bei Ihnen verwendete Schnittstelle)
- **Hardware Flow Control** muss auf **Nein** stehen
- Return
- Dann auf “Speichern als dfl” (Return)
- Verlassen auswählen
- `<CTRL><A>`, dann `<Z>`, dann `<X>` um Minicom zu beenden.

# Host: Installation minicom

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

- Am besten starten Sie Minicom direkt unter Angabe der seriellen Schnittstelle: `minicom -D /dev/ttyUSB0`
- Wenn Sie in einer virtuellen Maschine (VirtualBox) arbeiten, müssen Sie die Schnittstelle *durchreichen*. Dazu wählen Sie in VirtualBox (nicht Linux) den Menüpunkt [Geräte] und danach [USB] aus. Die Konfiguration kann und sollte in der Grundkonfiguration permanent gemacht werden.

## Schnittstellenkennung

Prolific Technology Inc. USB-Serial Controller D

## Section 4

### Kopplung Pi-Host-Ethernet

# Kopplung Raspberry Pi mit Host

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

## ACHTUNG

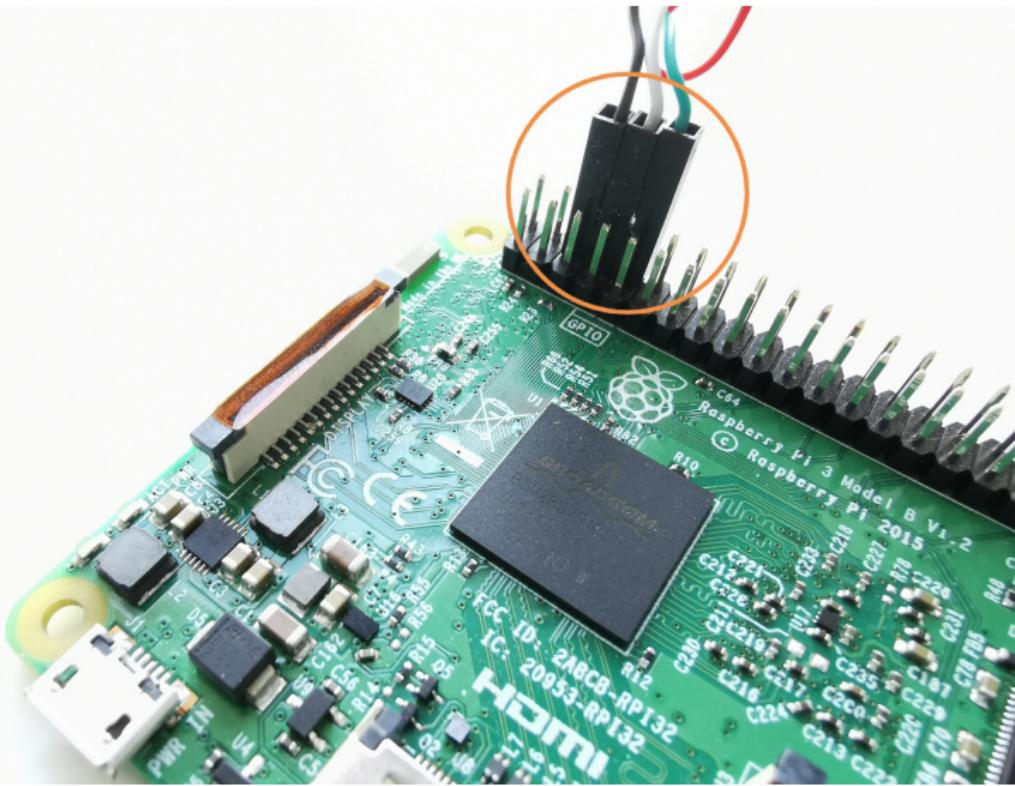
Achten Sie streng darauf das schwarze Kabel (GND) mit Pin 6 und **keinesfalls** mit Pin 4 (5V) oder Pin2 (5V) zu verbinden.  
Das beschädigt die Hardware!

Stellen Sie desweiteren sicher, dass das rote Kabel des USB-Serial Adapters mit keinem anderen Pin in Berührung kommt.

## Kopplung Raspberry Pi zu Host

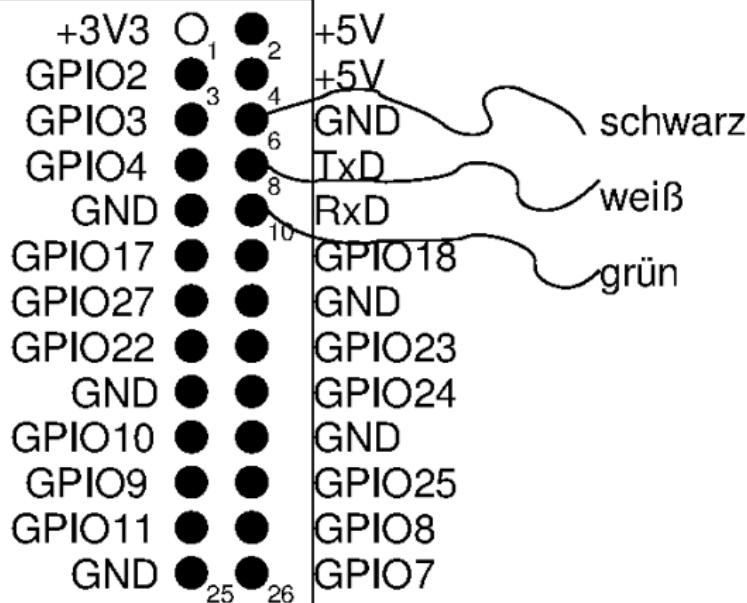
*Embedded  
Systems -  
Netzwerkboot*

Kopplung Pi-  
Host-Ethernet



# Kopplung Raspberry Pi zu Host (Fortsetzung)

SD-Karte



# Kopplung Pi - Ethernet

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

## Generelle Hinweise

- In der Kombination Windows, VirtualBox, Ethernet und Raspberry Pi kann es zu Problemen kommen, bei denen Windows die Netzwerkschnittstelle deaktiviert.
- Eine pragmatische Lösung des Problems besteht darin, den Raspberry Pi über einen Switch anzuschließen.
- Alternativ kann wireshark unter Windows installiert und aktiviert werden.

Verbinden Sie den Raspberry Pi (über einen Switch) mit dem Entwicklungsrechner

# Kopplung Pi - Ethernet

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

## Alternative 1: Separates Netz zw. Raspi und Host

- Raspberry Pi und Entwicklungsrechner werden per Ethernet-Kabel **miteinander** verbunden.
  - Bei Einsatz von VirtualBox ist ein **Switch** zwischenzuschalten.
- Es wird eine freie Netzadresse gewählt (z.B. 192.168.42.0/24)
- Dem Entwicklungsrechner wird eine statische IP-Adresse zugeteilt, beispielsweise 192.168.42.42.
- Dem Target wird eine statische IP-Adresse zugeteilt, z.B. 192.168.42.69.
- Konfiguration des Netzwerks am Ende des Foliensatzes.

# Kopplung Pi - Ethernet (Fortsetzung)

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

## Alternative 2: Raspi und Host im lokalen Netz

- Raspberry Pi und Hostrechner sind per Ethernet mit einem **Netzwerk** verbunden, der beiden Geräten typischerweise per DHCP eine IP-Adresse zuweist.
- Die IP-Adresse des Entwicklungsrechners (Host) ist zu evaluieren.
- Die IP-Adresse des Targets muss ebenfalls evaluiert werden. Alternativ wird eine freie Adresse gewählt und später konfiguriert.

## Section 5

### Raspberry Pi

# Grober Ablauf

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

- ① Raspberry Pi präperieren
  - ① Raspi OS installieren
  - ② Serielle Schnittstelle konfigurieren
  - ③ Raspberry Pi mit SD-Karte booten
  - ④ Einloggen und Bootloadersoftware aktualisieren
  - ⑤ Bootloader konfigurieren
- ② Test

# Raspberry Pi präperieren

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

## Installation Raspi OS

- Installation von rpi-imager
- Start des Imagers
  - `apt-install rpi-imager`
  - `./rpi-imager &`
- Auswahl des Betriebssystems (hier *Raspberry Pi OS (32 Bit)*)
- Auswahl der SD-Karte
  - Falls Sie mit einer virtuellen Maschine arbeiten, reichen Sie die SD-Karte über VirtualBox an das Linux System durch

# Raspberry Pi präperieren

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

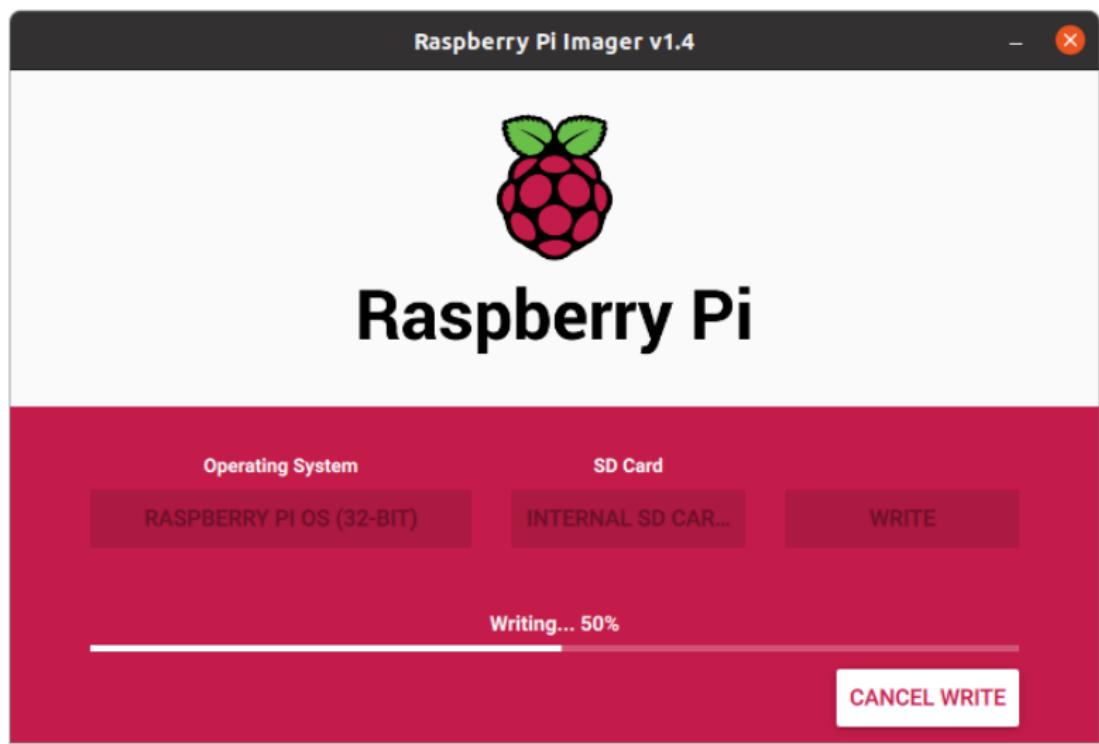
Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz



# Headless Raspberry Pi

- Um den Raspberry Pi 4 über die serielle Schnittstelle betreiben zu können, ist auf der SD-Karte noch die serielle Schnittstelle freizugeben.
- Stecken Sie die SD-Karte in das **Host-System**, so dass die Partitionen (automatisiert) gemountet werden.
- Hängen Sie die Zeile `dtoverlay=disable-bt` an die Datei `boot/config.txt` an.

## Kommando

```
1 echo "dtoverlay=disable-bt" >> /media/`whoami`/boot/config.txt
2 # aushaengen der Partitionen
3 umount /media/`whoami`/boot
4 umount /media/`whoami`/rootfs
5 sync
```

# Raspberry Pi 4 präperieren

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkonfig  
Standalone-  
Netz

## Start Raspi OS

- Nach der Installation: SD-Karte in den Raspberry Pi stecken
- Raspberry Pi booten

## Minicom

Falls noch nicht gemacht:

- Geben Sie in der virtuellen Maschine über das Geräte-Menu den USB-Seriell-Schnittstellenwandler frei.
- Starten Sie das Terminalprogramm `minicom`. Sie sollten in Minicom die Ausgaben von der Himbeere sehen.

```
minicom -D /dev/ttyUSB0
```

# RPi4: Netzwerkboot

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

Loggen Sie sich in das Raspi OS mit den Credentials pi und raspberry ein.

Firmware-Konfigurationssoftware installieren

```
sudo apt install rpi-eeprom
```

Bootloader konfigurieren

Datei bootconf.txt anpassen und aktivieren:

```
sudo -E rpi-eeprom-config --edit
```

# RPi4: Bootloader-Konfiguration

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

IP-Adressen sind anzupassen...

```
BOOT_UART=1
TFTP_IP=192.168.42.42
TFTP_PREFIX=1
TFTP_PREFIX_STR=
BOOT_ORDER=0x12
CLIENT_IP=192.168.42.69
SUBNET=255.255.255.0
```

# Falls es Probleme gibt ...

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

kann die nachfolgende Konfiguration ausprobiert werden  
(IP-Adressen sind anzupassen):

[all]

```
BOOT_UART=1
WAKE_ON_GPIO=1
POWER_OFF_ON_HALT=1
DHCP_TIMEOUT=30000
DHCP_REQ_TIMEOUT=1000
TFTP_FILE_TIMEOUT=20000
TFTP_IP=192.168.42.42
TFTP_PREFIX=1
BOOT_ORDER=0xf21
SD_BOOT_MAX_RETRIES=2
NET_BOOT_MAX_RETRIES=2
CLIENT_IP=192.168.42.69
SUBNET=255.255.255.0
[none]
FREEZE_VERSION=0
```

# Erläuterung Konfig-Parameter

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

## Ausgewählte Parameter

- **BOOT\_UART:** Meldungen werden über die serielle Schnittstelle ausgegeben
- **BOOT\_ORDER:** Legt die Boot-Reihenfolge fest
  - 0x12 priorisiert Netzwerkboot
  - 0x21 priorisiert Booten von SD-Karte
- **TFTP\_IP:** IP-Adresse des TFTP-Servers – in unserem Fall IP des *Hostsystems*
- **CLIENT\_IP:** Die eigene IP-Adresse; kann eventuell auch weggelassen werden und der Raspberry Pi bekommt per DHCP eine IP-Adresse

# Reboot . . .

```
POWER_OFF_ON_HALT=1
DHCP_TIMEOUT=30000
DHCP_REQ_TIMEOUT=1000
TFTP_FILE_TIMEOUT=20000
TFTP_IP=192.168.16.223
TFTP_PREFIX=1
BOOT_ORDER=0xf21
SD_BOOT_MAX_RETRIES=2
NET_BOOT_MAX_RETRIES=2
CLIENT_IP=192.168.16.203
SUBNET=255.255.255.0
[none]
FREEZE_VERSION=0

#####
*** To cancel this update run 'sudo rpi-eeprom-update -r' ***
*** INSTALLING /tmp/tmpbllx4_bt/pieeprom.upd ***
BOOTFS /boot
EEPROM update pending. Please reboot to apply the update.
pi@raspberrypi:~$ sudo su
root@raspberrypi:/home/pi# reboot[]

CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | tyUSB0
```

# Test

*Embedded  
Systems -  
Netzwerkboot*

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

- TFTP-Logging auf dem Host starten
- SD-Karte aus dem Raspberry Pi 4 entfernen
- Raspberry Pi 4 starten

## Section 6

### Fehlersuche

# tftpd

- Kommando `ps axwu | grep tftp` muss anzeigen, dass der Tftp-Server aktiv ist. Alternativ können Sie auch `systemctl status tftpd-hpa` eingeben.

- Protokollieren Sie in einem separaten Terminal die Ausgaben des `tftpd`:

```
tail -f /var/log/syslog | grep tftp
```

- Um den `tftpd` zu testen, booten Sie ein Linux von einer SD-Karte und loggen sich dort ein.

- Pingen Sie den `tftpd`-Server an: `ping IP-des-host-systems.`

- Kann der `tftpd` erreicht werden, installieren Sie den `tftp`-Client durch Eingabe von `apt install tftp-hpa` (ohne `d` wie Daemon). Verfolgen Sie auf dem Hostsystem die Ausgaben des `tftpd`. Laden Sie vom `tftp` eine Datei

# Serielle Schnittstelle des Raspberry Pi

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

- Haben Sie die Schnittstelle unter VirtualBox in das Hostsystem (Ubuntu) durchgereicht?
- Existiert auf dem Hostsystem die Datei /dev/ttyUSB0?
- Haben Sie Minicom mit der Option -D /dev/ttyUSB0 aufgerufen?
- Ist in der Minicom-Konfiguration (<strg><a>, <Z>) unter [Konfiguration][Einstellungen zum seriellen Anschluss] [Hardware Flow Control] und [Software Flow Control] auf Nein eingestellt?

## Section 7

### Netzwerkkonfig Standalone-Netz

# Setup Alternative 1

- Raspberry Pi und Entwicklungsrechner werden per Ethernet-Kabel **miteinander** verbunden.
  - Bei Einsatz von VirtualBox ist ein **Switch** zwischenzuschalten.
- Es wird eine freie Netzadresse gewählt (z.B. 192.168.42.0/24)
- Dem Entwicklungsrechner wird eine statische IP-Adresse zugeteilt, beispielsweise 192.168.42.42.
- Dem Target wird eine statische IP-Adresse zugeteilt, z.B. 192.168.42.69.

# Setup (Fortsetzung)

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

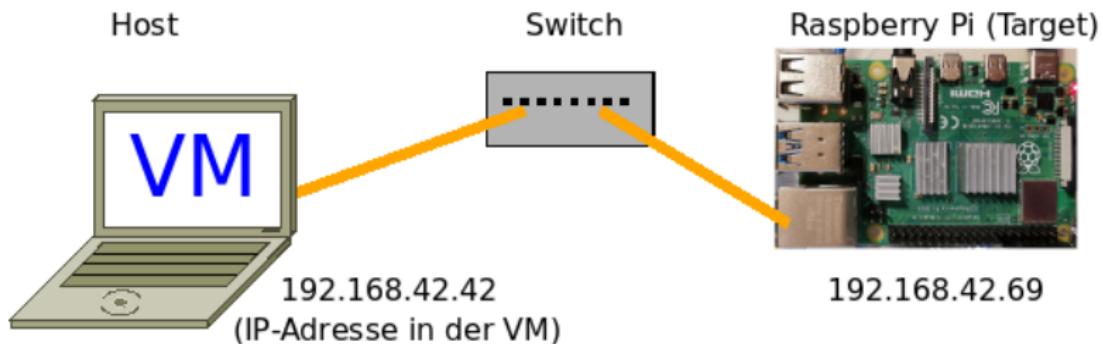
Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz



# Konfiguration NetzwerkManager

Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

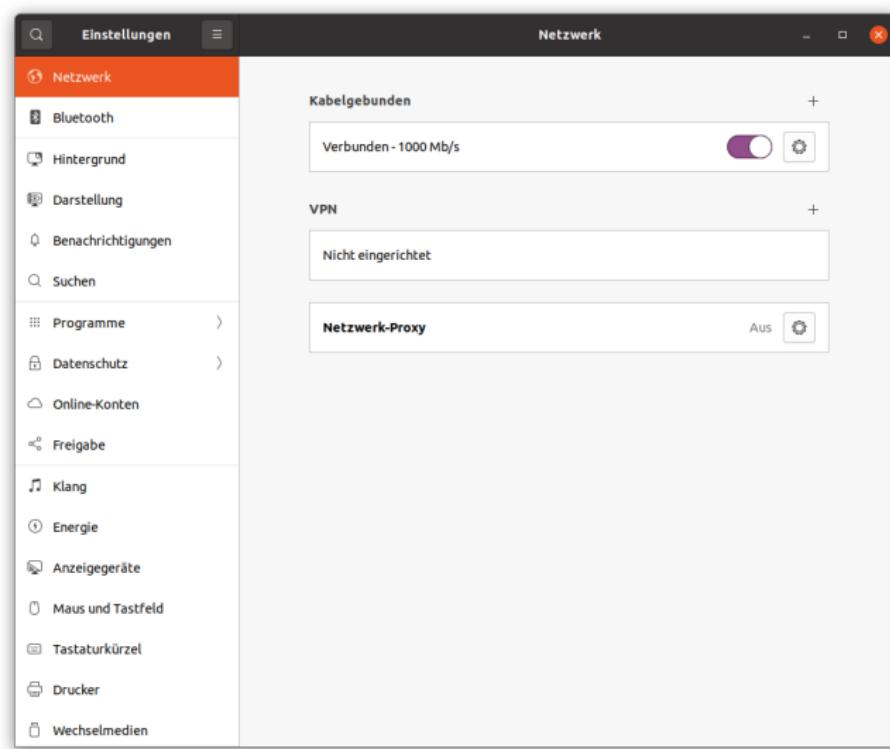
Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

- Ubuntu konfiguriert das Netzwerk über einen so genannten Netzwerkmanager.
- Auf der Oberfläche oben rechts auswählen:
  - [Kabelgebundene Verbindungen][LAN Einstellungen]
- Über das Pluszeichen eine neue Verbindung hinzufügen
- Als Name **tftp-server** eintragen
- IPv4 Parameter einstellen
- Verbindung speichern
- Verbindung aktivieren

# Kabelgebunden - Pluszeichen wählen



Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

# Profil 1 mit *tftp-server* austauschen

Embedded Systems - Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

**Abbrechen** **Neues Profil** **Hinzufügen**

Identität IPv4 IPv6 Sicherheit

Name **Profil 1**

MAC-Adresse

Duplizierte Adresse

MTU **automatisch**

The screenshot shows a software interface for managing network profiles. A modal dialog titled "Neues Profil" (New Profile) is open, with the "Identität" (Identity) tab selected. The profile name is set to "Profil 1". Other tabs include "IPv4", "IPv6", and "Sicherheit" (Security). Below the tabs are input fields for MAC-Adresse, Duplizierte Adresse, and MTU, each with a dropdown arrow icon. At the bottom right of the dialog are buttons for minus and plus.

# Profil 1 mit *tftp-server* austauschen

Embedded Systems - Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot  
Hintergrund  
Host-Software  
Kopplung Pi-  
Host-Ethernet  
Raspberry Pi  
Fehlersuche  
Netzwerkkonfig  
Standalone-  
Netz

Abbrechen      Neues Profil      Hinzufügen

**Identität**    IPv4    IPv6    Sicherheit

|                     |             |
|---------------------|-------------|
| Name                | tftp-server |
| MAC-Adresse         |             |
| Duplizierte Adresse |             |
| MTU                 | automatisch |

# IPv4 Parameter konfigurieren (Manuell, Adresse, Netzmaske)

Embedded Systems - Netzwerkboot

Jürgen Quade, Hochschule Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig Standalone-Netz

**Abbrechen** **Neues Profil** **Hinzufügen**

**Identität** **IPv4** **IPv6** **Sicherheit**

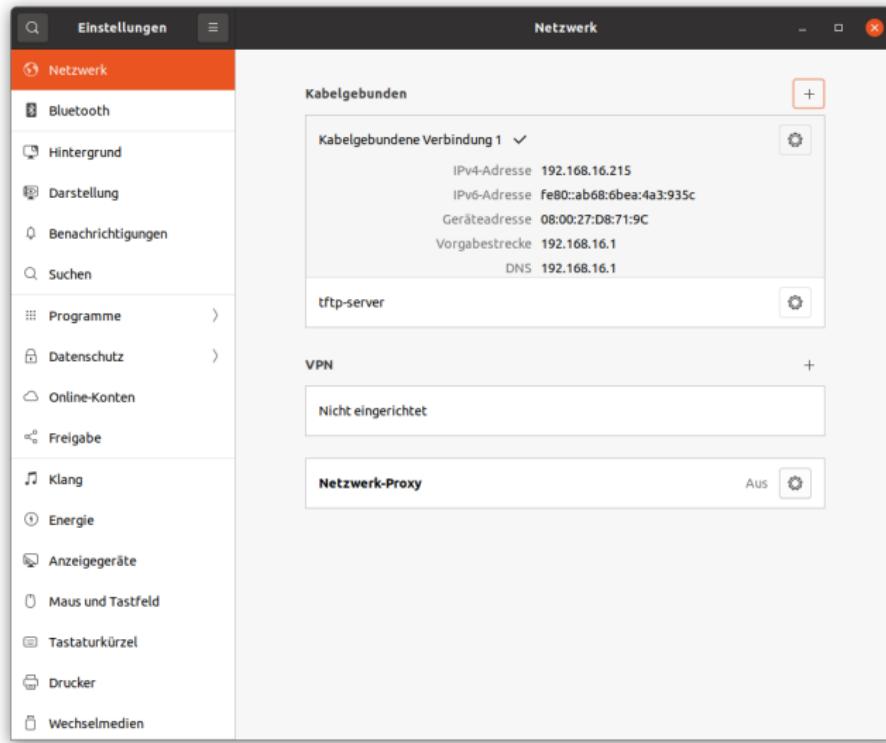
**IPv4-Methode**

Automatisch (DHCP)  Nur Link-Local  
 Manuell  Deaktivieren  
 Für anderen Rechner freigeben

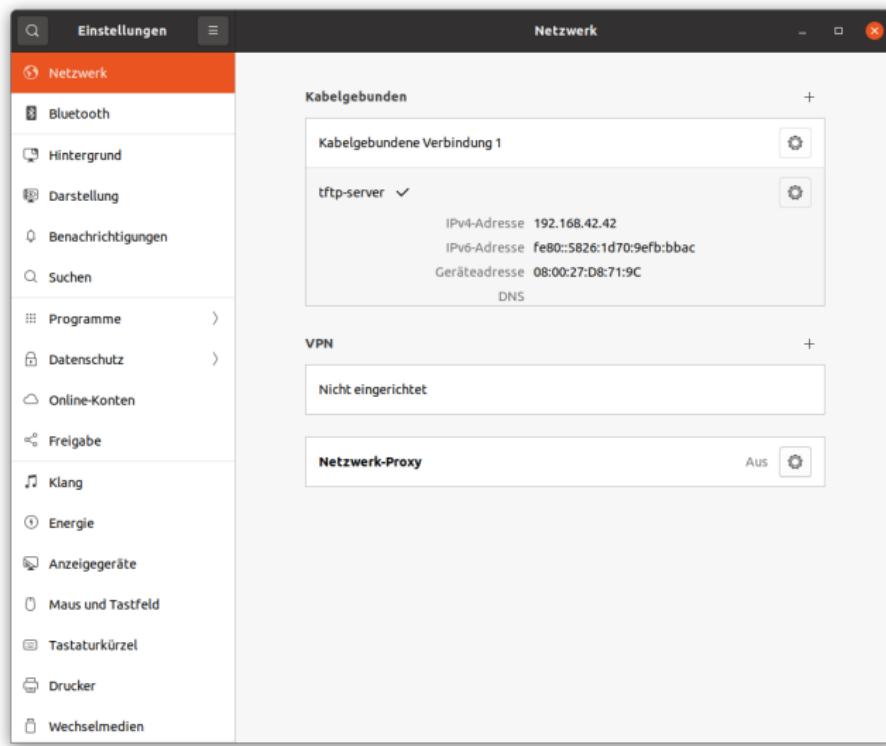
**Adressen**

| Adresse       | Netzmaske     | Gateway |
|---------------|---------------|---------|
| 192.168.42.42 | 255.255.255.0 |         |
|               |               |         |

# Ansicht nach Auswahl von *Hinzufügen*



# Neues Profil *tftp-server* aktivieren



Embedded  
Systems -  
Netzwerkboot

Jürgen Quade,  
Hochschule  
Niederrhein

Netzwerkboot

Hintergrund

Host-Software

Kopplung Pi-  
Host-Ethernet

Raspberry Pi

Fehlersuche

Netzwerkkonfig  
Standalone-  
Netz

# Self Learning 30.04.2024

Simon Krücken, Hochschule Niederrhein

23.04.2024

## Inhaltsverzeichnis

|                                 |          |
|---------------------------------|----------|
| <b>1 Allgemeines</b>            | <b>1</b> |
| 1.1 Lernziele . . . . .         | 1        |
| 1.2 Voraussetzung . . . . .     | 1        |
| 1.2.1 Bekannte Fehler . . . . . | 1        |
| 1.3 Arbeitsschritte . . . . .   | 1        |
| 1.3.1 Anmerkung . . . . .       | 2        |

## 1 Allgemeines

Der Termin am 30.04.2024 fällt aus.

Ich bin vom 28.04. bis zum 04.05. auch nicht erreichbar.

Bei Fragen wenden Sie sich bitte im Forum an Ihre Kommilitonen.

Da Sie 10 Semesterstunden pro Woche für dieses Modul eingeplant haben,  
gebe ich Ihnen hier ein paar Aufgaben, die Sie bitte bis zum 07.05.2024 erledigen.

### 1.1 Lernziele

- Buildroot besser verstehen lernen durch Eingriff in die Toolchain mittels Hook-Skripts (post-install.sh und post-image.sh).
- Kernel für die ARM-Architektur cross-kompilieren.
- Einen WLAN-Zugangspunkt einrichten und in Betrieb nehmen.

### 1.2 Voraussetzung

Ich gehe davon aus, dass bei jedem der Netzwerkboot mit dem eigenen gebauten Userland von Buildroot funktioniert. Sollte dies nicht der Fall sein, wenden Sie sich an Ihre Kommilitonen oder im Forum. Bei vielen läuft dieser Stand.

#### 1.2.1 Bekannte Fehler

- Sollte der Bootprozess dabei hängen bleiben, dass die USB-Geräte vom Raspberry Pi erkannt wurden:
  - config.txt unverändert in /srv/tftp kopieren.
  - cmdline.txt fehlt.

### 1.3 Arbeitsschritte

- 1. Schauen Sie sich die Vorlesung Embedded Systems - Systembuilder (II) an:
  - Implementieren Sie das post-build.sh- und das post-image.sh-Skript.
  - Erstellen Sie die init.d-Skripte zur Konfiguration der Netzwerkschnittstellen.
  - Fügen Sie die Webserver-Anwendung hinzu, mitsamt allen Konfigurationen und init.d-Skripten, die dazu gehören.
- 2. Kernel Cross-Kompilieren (Raspi als AP aufsetzen):
  - Nutzen Sie die Konfiguration aus dem Moodle-Raum als Vorlage.
  - Achten Sie darauf, dass Buildroot bereits fertig ist, da Sie die Compiler verwenden, die Buildroot generiert.

- Sollte Ihr Kernel nicht funktionieren, können Sie auf den im Moodle-Raum bereitgestellten Kernel ausweichen. Für das Projekt am Ende muss jedoch Ihr eigener Kernel laufen.
- 3. WLAN AP (WLAN-Grundlagen und Raspi als AP aufsetzen):
  - Richten Sie den Raspberry Pi als WLAN AP ein.
  - Verbinden Sie sich mit Ihrem Handy und prüfen Sie, welche IP-Adresse Sie erhalten haben.
  - Überprüfen Sie, ob Sie auf Ihren Webserver zugreifen können

#### **1.3.1 Anmerkung**

- Ein WLAN-Passwort MUSS mindestens 8 Zeichen haben.
- Prüfen Sie Ihre Pfade, Sie weichen eventuell von den Vorgaben ab

*Embedded  
Systems -  
Systembuilder*

Jürgen Quade,  
Hochschule  
Niederrhein

Systembuilder

Buildroot

Buildroot-  
Basiskonfig

Frisch  
gebootet

Next steps

# *Embedded Systems - Systembuilder*

Jürgen Quade, Hochschule Niederrhein

24.04.2023

## 1 Systembuilder

## 2 Buildroot

## 3 Buildroot-Basiskonfig

## 4 Frisch gebootet

## 5 Next steps

**Prof. Dr.-Ing. Jürgen Quade**  
**Hochschule Niederrhein**

## Section 1

### Systembuilder

# Systembuilder

Embedded  
Systems -  
Systembuilder

Jürgen Quade,  
Hochschule  
Niederrhein

Systembuilder

Buildroot

Buildroot-  
Basiskonfig

Frisch  
gebootet

Next steps

LINUX FOUNDATION COLLABORATIVE PROJECTS

yocto • PROJECT

SEARCH  Go

ABOUT

ECOSYSTEM

DOWNLOADS

TOOLS + RESOURCES

DOCUMENTATION

New to the Project

Want to learn more, or just kick the tires? Start here.

START HERE TO LEARN MORE ▾



It's not an embedded Linux distribution – it creates a custom one for you

The Yocto Project is an open source collaboration project that provides templates, tools and methods to help you create custom Linux-based systems for embedded products regardless of the hardware architecture. [Read more](#)

# Aufgaben eines Systemgenerators

- Generierung der Cross-Entwicklungsumgebung
- Systemkomponenten über Pakete zur Auswahl stellen
- Konfiguration der Pakete zu ermöglichen
- Das eingebettete System zu generieren
  - Firmware, Kernel, Rootfilesystem, Anwendungen
  - Zusammenbau des Systems
  - Eventuell Installation

## Defizite des *Handmade*-Systems

- Kein User-Management
- Kein Remote-Zugriff per SSH
- Keine Bibliotheken (z.B. Grafik)

# Varianten

Embedded  
Systems -  
Systembuilder

Jürgen Quade,  
Hochschule  
Niederrhein

Systembuilder

Buildroot

Buildroot-  
Basiskonfig

Frisch  
gebootet

Next steps

- Yocto
  - extrem hoher Einarbeitungsaufwand
  - extrem Ressourcenhungig (Speicher, Rechenzeit)
- OpenEmbedded
  - von OpenEmbedded abgeleitet
- Buildroot
  - schlank
  - übersichtlich
  - leicht zu verstehen

## Section 2

### Buildroot

# Allgemeines

*Embedded  
Systems -  
Systembuilder*

Jürgen Quade,  
Hochschule  
Niederrhein

Systembuilder

Buildroot

Buildroot-  
Basiskonfig

Frisch  
gebootet

Next steps

- Open-Source Systembuilder
- Sammlung von Skripten zum Bauen von Systemsoftware (Bootloader, Kernel, Userland)
- Unterstützt mehr als 50 verschiedene Plattformen
- kleiner Footprint, effizient
- Konfigurierbar per `make menuconfig`

# Grundstruktur

Embedded  
Systems -  
Systembuilder

Jürgen Quade,  
Hochschule  
Niederrhein

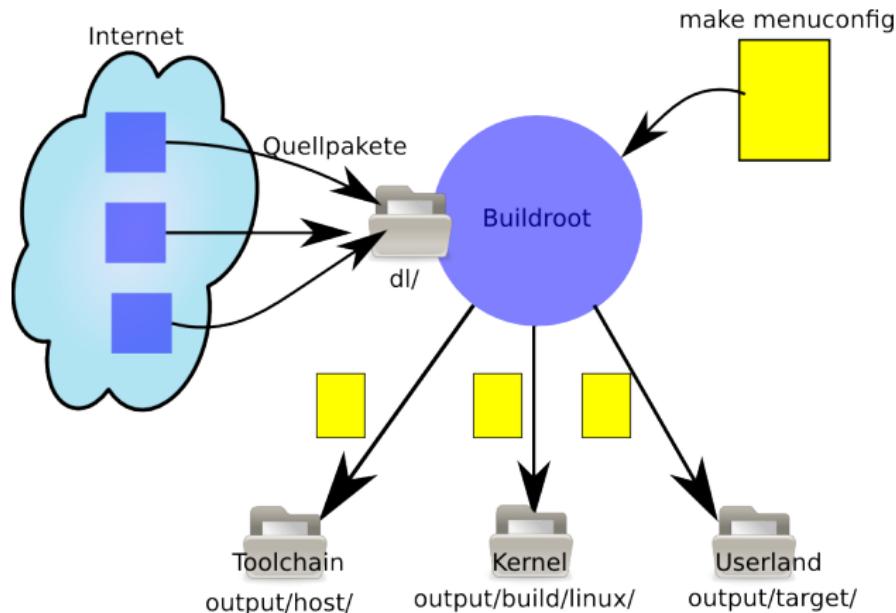
Systembuilder

Buildroot

Buildroot-  
Basiskonfig

Frisch  
gebootet

Next steps



# Konfiguration

Embedded  
Systems -  
Systembuilder

Jürgen Quade,  
Hochschule  
Niederrhein

Systembuilder

Buildroot

Buildroot-  
Basiskonfig

Frisch  
gebootet

Next steps

- Konfiguration per `make menuconfig`
- Targets (Generierungsziele):
  - `make menuconfig`
  - `make busybox-menuconfig`
  - `make uclibc-menuconfig`
  - `make linux-menuconfig`
  - `make toolchain`
  - `make dl` (Download)
  - `make clean`
  - `make <paketname>-reconfigure`
  - `make <paketname>-rebuild`

# Verzeichnisstruktur

Embedded  
Systems -  
Systembuilder

Jürgen Quade,  
Hochschule  
Niederrhein

Systembuilder

Buildroot

Buildroot-  
Basiskonfig

Frisch  
gebootet

Next steps

| Verzeichnis         | Bedeutung                                                                 |
|---------------------|---------------------------------------------------------------------------|
| configs/            | Konfigurationsdateien für die unterstützten Plattformen                   |
| output/images/      | Enthält die generierten Imagedateien (Kernel, Bootloader, Rootfilesystem) |
| output/build/       | Zur Generierung notwendige Werkzeuge                                      |
| output/host/usr/dl/ | Generierte Cross-Entwicklungs Umgebung Ablage für die Download-Dateien    |
| docs/               | Dokumentation                                                             |
| package/            | Für die Pakete notwendige Patches/Startskripts                            |
| toolchain/          | Konfigurationen/Patches für die Toolchain                                 |

# Wo finde ich was?

- Änderungen am **erzeugten** Filesystem können über das Verzeichnis 'system/skeleton/' vorgenommen werden.
- Zugriffsrechte lassen sich über 'system/device\_table.txt' einstellen
- Ein post-build-Skript wird **nach** der Generierung der Pakete und **vor** dem Zusammenbau des Rootfilesystems aufgerufen.
- Ein Postimage-Skript wird **nach** dem Zusammenbau des Systems aufgerufen (wird von uns zur Installation genutzt werden).

## Section 3

### Buildroot-Basiskonfig

# Überblick

Embedded  
Systems -  
Systembuilder

Jürgen Quade,  
Hochschule  
Niederrhein

Systembuilder

Buildroot

Buildroot-  
Basiskonfig

Frisch  
gebootet

Next steps

Um mit Buildroot ein Embedded Linux für den Raspberry Pi zu bauen, sind die folgenden Schritte notwendig:

## Schritte

- ① Download und Installation von Buildroot (bereits erledigt)
- ② Defaultkonfiguration (Auswahl Raspberry Pi) (bereits erledigt)
- ③ Konfiguration des Embedded Linux (Auswahl und Konfig der Pakete)
- ④ Generierung der Programme für das Hostsystem und Generierung der Image- und Archivedateien für das Target
- ⑤ Installation des Embedded Linux

# Download und Installation

Embedded  
Systems -  
Systembuilder

Jürgen Quade,  
Hochschule  
Niederrhein

Systembuilder

Buildroot

Buildroot-  
Basiskonfig

Frisch  
gebootet

Next steps

Die nachfolgenden Schritte haben wir teilweise bereits  
durchgeführt.

## Buildroot herunterladen

```
user@host:~/embedded/raspi$ sudo apt install
user@host:~/embedded/raspi$ git clone \
    https://github.com/buildroot/buildroot.git
user@host:~/embedded/raspi$ cd buildroot
user@host:~/embedded/raspi/buildroot$ \
    make raspberrypi4_defconfig
```

# Konfiguration (1)

Embedded  
Systems -  
Systembuilder

Jürgen Quade,  
Hochschule  
Niederrhein

Systembuilder

Buildroot

Buildroot-  
Basiskonfig

Frisch  
gebootet

Next steps

## Konfiguration

```
user@host:~/embedded/raspi/buildroot$ make menuconfig
# Toolchain: uclibc-ng, ipv6, uchar, nptl, c++, largfile support
[Toolchain] [C Library uClibc-ng]
[Toolchain] [Enable WCHAR support]
[Toolchain] [Enable toolchain locale/i18n support]
[Toolchain] [Thread library implementation
           \Native Posix Threading (NPTL)]
[Toolchain] [Enable C++ support]
[Toolchain] [Build cross gdb for the host]
[Toolchain] [Thread library debugging]
```

# Konfiguration (2)

Embedded  
Systems -  
Systembuilder

Jürgen Quade,  
Hochschule  
Niederrhein

Systembuilder

Buildroot

Buildroot-  
Basiskonfig

Frisch  
gebootet

Next steps

Die nachfolgenden Pakete **neu** auswählen:

```
[System configuration] [(Ihr Name hier) System hostname]
# Vergeben Sie hier ein Passwort, beispielsweise "root"
[System configuration] [(root) Root password]
[Filesystem images] [cpio the root filesystem (for
                     use an an initial RAM filesystem)]
[Filesystem images] [Compression method (gzip) --]

# SSH-Server
[Target packages] [Networking applications] [dropbear]
```

# Konfiguration (4)

## Fortsetzung Konfiguration Buildroot

# Löschen des Eintrags

[System configuration] [Custom scripts to run  
after creating filesystem images]

# Abwählen des Paketes \*Linux Kernel\*

[Kernel] [Linux Kernel]

# Auswählen des Paketes dropbear

# Hier selber suchen

```
user@host:~/embedded/raspi/buildroot$ time make
```

# Installation Userland

Embedded  
Systems -  
Systembuilder

Jürgen Quade,  
Hochschule  
Niederrhein

Systembuilder

Buildroot

Buildroot-  
Basiskonfig

Frisch  
gebootet

Next steps

Buildroot ist so konfiguriert worden, dass ein CPIO-Archiv als Initramfs erzeugt wurde. Dieses wird auf das tftp-Verzeichnis kopiert. Sind Kernel und Userland kopiert, kann der Raspberry Pi gestartet werden.

## rootfs.cpio kopieren

```
user@host:~/embedded/raspi/buildroot-2023.02$  
user@host:~/embedded/raspi/buildroot-2023.02$ cp \  
    output/images/rootfs.cpio /srv/tftp/rootfs.cpio
```

## Section 4

Frisch gebootet

# Login über die serielle Verbindung

Embedded  
Systems -  
Systembuilder

Jürgen Quade,  
Hochschule  
Niederrhein

Systembuilder

Buildroot

Buildroot-  
Basiskonfig

Frisch  
gebootet

Next steps

```
[ 1.954396] mmc0: host does not support reading read-only switch. assuming w.  
[ 1.967161] Indeed it is in host mode hprt0 = 00021501  
[ 1.977225] mmc0: new high speed SD card at address b368  
[ 1.997000] mmcblk0: mmc0:b368 00000 1.83 GiB  
[ 2.004312] mmcblk0: p1 p2  
[ 2.055929] EXT4-fs (mmcblk0p2): mounted filesystem without journal. Opts: ()  
[ 2.066659] VFS: Mounted root (ext4 filesystem) on device 179:2.  
[ 2.079921] devtmpfs: mounted  
[ 2.085153] Freeing unused kernel memory: 132K (c0574000 - c0595000)  
[ 2.167765] usb 1-1: new high-speed USB device number 2 using dwc_otg  
[ 2.176031] Indeed it is in host mode hprt0 = 00001101  
[ 2.285650] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)  
[ 2.387545] usb 1-1: New USB device found, idVendor=0424, idProduct=9512  
[ 2.423007] usb 1-1: New USB device strings: Mfr=0, Product=0, SerialNumber=0  
[ 2.466494] hub 1-1:1.0: USB hub found  
[ 2.498235] hub 1-1:1.0: 3 ports detected  
♦  
Welcome to Buildroot  
buildroot login: [ 2.797099] usb 1-1.1: new high-speed USB device number 3 ug  
[ 2.917356] usb 1-1.1: New USB device found, idVendor=0424, idProduct=ec00  
[ 2.925952] usb 1-1.1: New USB device strings: Mfr=0, Product=0, SerialNumber=0  
[ 2.938384] smsc95xx v1.0.4  
[ 3.000873] smsc95xx 1-1.1:1.0 eth0: register 'smsc95xx' at usb-bcm2708_usb-a
```

# SSH-Zugriff

- ① Loggen Sie sich auf dem System ein.
- ② Geben Sie dem Superuser ein Passwort.
- ③ Eventuell müssen Sie dem Ethernet-Interface eine IP-Adresse zuteilen
- ④ Einloggen auf dem Target per SSH

## Kommandos Target

```
Welcome to Buildroot  
buildroot login: root  
...  
passwd  
ifconfig eth0 192.168.42.99
```

## Kommandos Host

```
ssh root@192.168.42.09
```

## Section 5

Next steps

# Aufgaben

Embedded  
Systems -  
Systembuilder

Jürgen Quade,  
Hochschule  
Niederrhein

Systembuilder

Buildroot

Buildroot-  
Basiskonfig

Frisch  
gebootet

Next steps

- Ändern Sie den Systemnamen auf „master-device“
- Ändern Sie den Anmeldestring (System banner) auf „Welcome Master“.
- Generieren Sie das System neu.
- Kopieren Sie das generierte System in das Download-Verzeichnis des tftp-Servers.
- Überprüfen Sie den Erfolg ihrer Änderungen.

# *Embedded Systems - Systembuilder (II)*

Jürgen Quade, Hochschule Niederrhein

24.04.2023

## 1 Motivation

## 2 Systemdienste

## 3 Hintergrund

## 4 Optimierung Entwicklungs-umgebung

## 5 Netzwerk-Konfiguration

## 6 Test

## 7

# Applikation: Webserver

*Embedded  
Systems -  
Systembuilder  
(II)*

Jürgen Quade,  
Hochschule  
Niederrhein

Motivation

Systemdienste

Hintergrund

Optimierung  
Entwicklungs-  
umgebung

Netzwerk-  
Konfiguration

Test

Applikation:  
Webserver

*Embedded  
Systems -  
Systembuilder  
(II)*

Jürgen Quade,  
Hochschule  
Niederrhein

Motivation

Systemdienste

Hintergrund

Optimierung  
Entwicklungs-  
umgebung

Netzwerk-  
Konfiguration

Test

Applikation:  
Webserver

Prof. Dr.-Ing. Jürgen Quade  
Hochschule Niederrhein

## Section 1

### Motivation

# Defizite

Nachdem wir uns bisher mit der Erstellung von Kernel und Userland beschäftigt haben, soll in dieser Lerneinheit die Konfiguration und Konfektionierung des Systems behandelt werden.

Das bisherige System hat beispielsweise noch keinerlei funktionale Komponente (Webserver). Außerdem gibt es die folgenden Punkte zu kritisieren:

- Passwort als Klartextpasswort abgespeichert
- Kein User-Login
- Keine aktuelle Zeit
- Kein Zugriff auf die SD-Karte
- Kein WLAN
- ...

# Lösungen

Um diese Schwächen auszubügeln sind einige Systemkenntnisse notwendig, die im folgenden stichwortartig adressiert werden sollen:

- Syslog
- Timekeeping
- Zugriff auf Datenspeicher
- SSH-Server (dropbear)
- WLAN (wpa\_supplicant)

# Startvorgang

Embedded  
Systems -  
Systembuilder  
(II)

Jürgen Quade,  
Hochschule  
Niederrhein

Motivation

Systemdienste

Hintergrund

Optimierung  
Entwicklungs-  
umgebung

Netzwerk-  
Konfiguration

Test

Applikation:  
Webserver

Wie bereits mehrfach thematisiert ist der erste Prozess, der vom Kernel gestartet wird, der Prozess `init`. In aktuellen Debian-Versionen ist `init` zwischenzeitlich durch das modernere aber auch komplexere `systemd` ersetzt worden. Im Bereich eingebetteter Systeme findet man daher weiterhin `init` vor.

`Init` wird über die `inittab` konfiguriert. Diese Konfiguration kommt aber schnell an seine Grenzen, so dass als flexiblere zweite Stufe `rcS` durch `init` aktiviert wird.

- Das Skript rcS sucht im Verzeichnis /etc/init.d nach Skripten.
- Skripte sind gemäß folgender Konvention benamt:  
**SxxName**  
S=Start, xx=Nummer zwischen 1 und 99, Name=Name des Dienstes
- Init ruft die Skripte in numerischer Reihenfolge auf

# rcS (Fortsetzung)

- Skripte, die mit S beginnen sind Startskripte und werden beim Hochfahren aufgerufen
- Skripte, die mit K beginnen, werden beim Herunterfahren des Systems aufgerufen
- Skripte werden mit einem Parameter aufgerufen:
  - start
  - stop
  - restart
  - reload
- Da die Auswertung der standardisierten Parameter für alle gleich ist, kann für eigene Entwicklungen ein Template als Basis verwendet werden.

# rcS (Fortsetzung)

## Template eines rcS-Skripts

```
#!/bin/sh

case "$1" in
    start)
        echo "Starting Service ..."
        /path/to/service/to/start
        ;;
    stop)
        ;;
    restart|reload)
        ;;
    *)
        echo "Usage: $0 {start|stop|restart}"
        exit 1
esac

exit $?
```

## Section 2

### Systemdienste

# Syslog

- Für das Loggen von Systeminformationen ist der Job `syslogd` zuständig
- Syslogd kann über `/etc/syslogd.conf` konfiguriert werden
- Jede zu loggende Nachricht kann einer Ebene (Debug, Info, Notice, Warning, Error, Critical, Alert, Emergency) zugeordnet werden
- Nachrichten werden klassisch in `/var/log/syslog` oder in `/var/log/messages` abgespeichert
- Der Kernel legt seine Nachrichten (zusätzlich) unter `/var/log/kern.log` ab
- Damit das Verzeichnis `/var/log/` nicht irgendwann überläuft, kann `logrotate` Logdateien komprimieren und löschen.

# Syslog (Fortsetzung)

Um einen Einblick in `syslogd` zu bekommen, können Sie folgendermaßen vorgehen:

- Öffnen Sie eine separate Konsole (Terminal)
- Geben Sie das Kommando `tail -f /var/log/syslog` ein
- Lassen Sie das Kommando ständig (nebenbei) laufen

# Timekeeping

- Der Raspberry Pi besitzt keinen Uhrenbaustein, in dem sich die aktuelle Uhrzeit (UTC) befindet
- Die aktuelle Zeit wird über Internet per *Network Time Protocol* (ntp) geladen
- NTP holt sich die Zeit von einem Zeitserver (z.B. 0.pool.ntp.org), Laufzeiten werden ausgerechnet
- NTP passt die Uhrzeit ständig der Uhr des NTP-Servers an
- !Anmerkung! Funktioniert nur wenn der Raspberry Pi auch eine Verbindung zum Internet hat also nicht bei einer direkten Computer zu Pi Verbindung.

# Zugriff auf Datenspeicher

- Bisheriges Userland befindet sich im RAM

## Vorteile Initramfs

- Keine Abnutzung von Flash-Speicher
- Schneller Zugriff
- Kürzere Bootzeiten (kein Filesystemcheck)

## Nachteile Initramfs

- Weniger Hauptspeicher
- Modifikationen sind nicht persistent

## Konsequenzen

- Persistente Daten müssen auf einem Hintergrundspeicher abgelegt werden
- Jede (formatierte) Partition ist über eine Gerätedatei identifiziert (`/dev/mmcblk0p1`).
- Kommando `mount` ordnet eine Partition einem Verzeichnis zu
- Zuordnung kann in eine Datei abgelegt werden (`/etc/fstab`)
- Kommando `mount -a` hängt automatisiert die in `fstab` konfigurierten Partitionen ein

# SSH-Server

*Embedded  
Systems -  
Systembuilder  
(II)*

Jürgen Quade,  
Hochschule  
Niederrhein

Motivation

Systemdienste

Hintergrund

Optimierung  
Entwicklungs-  
umgebung

Netzwerk-  
Konfiguration

Test

Applikation:  
Webserver

- Sicherer Remote-Login per SSH
- Im Embedded-Bereich hat sich dropbear als SSH-Server etabliert
- Netzwerk muss konfiguriert sein

# DHCP

- Ein DHCP-Server teilt anfragenden Rechnern (DHCP-Client) IP-Adressen zu
- Als DHCP-Client wird häufig udhcp eingesetzt
- Als DHCP-Server wird häufig dnsmasq eingesetzt
- Für jedes Netzwerkinterface (eth0, wlan0) muss festgesetzt werden, ob die IP-Adresse
  - statisch oder
  - dynamisch per DHCP zugewiesen wird.
- Ein Router (ebenso Accesspoint) fungiert typischerweise als DHCP-Server

# Kommando install

- Das Kommando `install` vereint drei Kommandos:
  - kopieren (`cp`)
  - Besitzrechte setzen (`chown`, Optionen `-o` für *owner* und `-g` für *group*) und
  - Zugriffsrechte konfigurieren (`chmod`)
- Es ist einem normalen `cp` damit vorzuziehen

## Beispiel

```
install -o root -g root -m 0755 S40network etc/  
install -m 644 -o root -g root target/httpd.conf /etc/
```

## Section 3

### Hintergrund

# Buildroot-Konfiguration

- Buildroot speichert die Konfiguration in einer (versteckten) Datei `.config`.
- Manchmal ist es einfacher Konfigurationen mithilfe eines Editors direkt in der Datei `.config` vorzunehmen.
- Ein **Backup** der Konfiguration ist durch Kopieren der Datei schnell gemacht.

# Ausschnitt .config

```
# Automatically generated file; DO NOT EDIT.  
# Buildroot 2023.02 Configuration  
  
#  
BR2_HAVE_DOT_CONFIG=y  
BR2_HOST_GCC_AT_LEAST_4_9=y  
BR2_HOST_GCC_AT_LEAST_5=y  
BR2_NEEDS_HOST_UTF8_LOCALE=y  
  
# Target options  
#  
BR2_ARCH_HAS_MMU_OPTIONAL=y  
# BR2_arcle is not set  
# BR2_arceb is not set  
BR2_arm=y
```

# buildroot-Dokumentation

- Orginal-Dokumentation ist unter <https://buildroot.org/downloads/manual/manual.html> zu finden
- Die Dokumentation gliedert sich in
  - User-Dokumentation (Nutzer von Buildroot)
  - Entwickler-Dokumentation (zur Weiterentwicklung von Buildroot selbst)
- Buildroot bietet ausgeklügeltere Möglichkeiten als im Folgenden vorgestellt, projektspezifische Aspekte zu implementieren bzw. zu verwalten (Stichwort **br2-external tree**)

## Section 4

### Optimierung Entwicklungs-umgebung

# Anforderungen und Ziele

Nicht alle notwendigen Systemanpassungen lassen sich per `make menuconfig` konfigurieren.

## Forderung

- Änderungen an Buildroot selbst sollten möglichst wenig invasiv sein um leichteres Update auf neuere Versionen zu ermöglichen.
- Build-, Test- und Install-Prozess sollte weitestgehend automatisiert erfolgen.

# Idee

Embedded  
Systems -  
Systembuilder  
(II)

Jürgen Quade,  
Hochschule  
Niederrhein

Motivation

Systemdienste

Hintergrund

Optimierung  
Entwicklungs-  
umgebung

Netzwerk-  
Konfiguration

Test

Applikation:  
Webserver

- Buildroot nur über `make menuconfig (.config)` konfigurieren.
- Zusätzliche Aktionen/Modifikationen in einem separaten Entwicklungszweig vornehmen (`raspi/target`).
- Per `post-build.sh` und `post-image.sh` beide Teile zusammenführen.

# Verzeichnisstruktur

- raspi/linux – Kernel
- raspi/buildroot – Target-Userland
- raspi/target – spezifische Anpassungen für das Target
- raspi/scripts – notwendige Skripte auf dem Host

Neue Verzeichnisse scripts und target anlegen:

## Host-Kommandos

```
cd ~/embedded/raspi/  
mkdir scripts  
mkdir target
```

# Post-build

- Buildroot ruft nach dem Generieren sämtlicher Komponenten ein *Post-build-Skript* auf.
- Pfad zum Post-build-Skript wird per `make menuconfig` konfiguriert.
- Für den Raspberry-Pi gibt es im Buildroot-Verzeichnis `boards/` bereits ein Post-build-Skript das aufgerufen wird.

## ToDo

- ➊ Eigenes Post-build-Skript erstellen
- ➋ Im Post-build-Skript das orginale Skript aufrufen
- ➌ Buildroot-Konfiguration anpassen.

# Post-build-Skript (1)

- Dient zur Anpassung des generierten Images (Userland)
- Wird auf dem Entwicklungsrechner (Host) abgearbeitet

## WICHTIG

Das Skript bekommt als Parameter den Pfad zum generierten Target-System übergeben (im Skript \$1)

# Post-build-Skript (2)

## Vorbereitung

- Template für das Skript (`post-build.sh`) im Verzeichnis ('`~/embedded/raspi/scripts/`') anlegen.
- `post-build.sh` ausführbar machen (`chmod +x post-build.sh`).
- Pfad und Name (`../scripts/post-build.sh`) des post-build-Skripts in Buildroot eintragen (`make menuconfig`)

## Buildroot Auswahlpunkte für das post-build-Skript

[System configuration][Custom scripts to run before creating filesystem images]

## Alternative

Konfiguration per Editor in `.config` eintragen.

# Post-build-Skript (3)

## Template

```
#!/bin/sh

# MARK A: Orginalskript starten
#
# MARK B: Netzwerk
#
# MARK C: Zeit setzen
#
# MARK D: SD-Karte einhaengen
#
# MARK E: Webserver
#
# MARK F: Firewall
#
# MARK G: Usermanagement
```

# MARK A: Orginalskript starten

## Ergänzungen in post-build.sh

```
echo "${TARGET_DIR}../../../../board/raspberrypi4/post-build.sh $*"  
${TARGET_DIR}../../../../board/raspberrypi4/post-build.sh $*
```

# Postimage-Skript (1)

Das Postimage-Skript `scripts/post-image.sh` bietet sich an, um damit das generierte Buildroot-System automatisch in das tftp-Verzeichnis zu kopieren.

# Postimage-Skript (2)

## Vorgehen

Skript `~/embedded/raspi/scripts/post-image.sh` erstellen (siehe nächste Folie), das den Kernel und das Initramfs in das Verzeichnis `/srv/tftp/` kopiert.

## Buildroot: `make menuconfig`

[System configuration][`(..)/scripts/post-image.sh`]  
Custom scripts to run after creating filesystem images]

## Alternative

Konfiguration per Editor in `.config` eintragen.

# Postimage-Skript (3)

## scripts/post-image.sh

```
#!/bin/bash

# Orginalscript starten
echo "${TARGET_DIR}/../../..	board/raspberrypi4/post-image.sh $*"
echo
${TARGET_DIR}/../../..	board/raspberrypi4/post-image.sh $*

cp output/images/rootfs.cpio /srv/tftp/
echo "Done"
```

## Section 5

### Netzwerk-Konfiguration

# Netzwerk Mechanik

- Unix-Systeme konfigurieren Netzwerkinterfaces über die Kommandos
  - `S40network`
  - `ifup`
  - `ifdown`
- Konfigurationsdateien befinden sich unter
  - `/etc/network/interfaces` und
  - `/etc/resolv.conf`
- Aufgrund von Timing-Problemen muss das Kommando `ifup` eventuell durch Einbau eines `sleep` verlangsamt werden

# Konfiguration

- Rufen Sie auf dem Host-System `man interfaces` auf, um die Syntax der Konfigurationsdatei kennen zu lernen.
- Erstellen Sie die Datei `interfaces` im Unterverzeichnis `~/embedded/raspi/target/`
- Erstellen Sie die Datei `resolv.conf` im Unterverzeichnis `~/embedded/raspi/target/` (evtl. vorher Man-Page studieren)
- Klicken Sie das Kopieren der Datei über das Skript `post-build.sh` in den Build-Prozess ein.

## Section 6

Test

# Test

- Zum Test im Buildroot-Verzeichnis `make` aufrufen.
- Per `ls -lrt /srv/tftp/` kann das Datum der Datei `rootfs.cpio` überprüft werden.
- Nach Reboot des Raspberry Pi muss das neu generierte System booten.

# Fehlersuche

- Schauen Sie sich die Ausgaben von *buildroot* genau an.
- Haben die Skripte *post-build.sh* und *post-image.sh* Ausführ-Rechte?
- Hat Buildroot Schreibrechte auf das Verzeichnis */srv/tftp/*?
- Stimmen die in der Buildroot-Konfiguration abgelegten Pfade zu den eigenen Skripten?
  - eventuell über die Datei *.config* überprüfen

## Section 7

### Applikation: Webserver

# Übersicht

- Unser Embedded System soll wieder statische und dynamische Webinhalte zur Verfügung stellen.
- Buildroot bietet unterschiedliche Webserver an:
  - boa
  - lighttpd
  - mongoose
  - thttpd
  - tinyhttpd
- Wir nehmen weiterhin den von Busybox zur Verfügung gestellten Webserver.

# Vorgehen

## Schritte

- ① Benötigte Dateien für das Target (aus dem Qemu-System) kopieren
  - index.html
  - ps.cgi
  - httpd.conf
  - (Neu) S51httpd
- ② In Busybox den Webserver auswählen (`make busybox-menuconfig`)
- ③ `post-build.sh` ergänzen
- ④ Test

# 1. Webserverinhalte

## Kommandos auf dem Entwicklungssystem

```
cp ~/embedded/qemu/userland/target/index.html \
~/embedded/raspi/userland/target/
cp ~/embedded/qemu/userland/target/ps.cgi \
~/embedded/raspi/userland/target/
cp ~/embedded/qemu/userland/target/httpd.conf \
~/embedded/raspi/userland/target/
```

## 2. In Busybox den Webserver auswählen

### Buildroot-Konfiguration

```
user@host:~/embedded/raspi/buildroot$ \
    make busybox-menuconfig
[Networking Utilities] [httpd]
```

### 3. Generierung anpassen

#### Ergänzung zu post-build.sh

```
# MARK D: Webserver
echo "Webserver..."
mkdir -p $1/var/www/cgi-bin
install -m 755 ./userland/target/S51httpd $1/etc/init.d/
install ./userland/target/httpd.conf $1/etc/
install -D ./userland/target/index.html $1/var/www/
install -D -m 755 ./userland/target/ps.cgi $1/var/www/cgi-bin/
```

# 4. Test und Arbeiten mit dem System (1)

## Funktionstest Applikation

- Nach dem Booten: Zugriff über  
<http://192.168.42.69/index.html>

# *Eingebettete Systeme - Systembuilder WLAN-AP*

Jürgen Quade, Hochschule Niederrhein

9.05.2021

## Inhaltsverzeichnis

|                                       |          |
|---------------------------------------|----------|
| <b>Einführung</b>                     | <b>1</b> |
| Netzwerk-Architektur . . . . .        | 1        |
| Router . . . . .                      | 1        |
| Firewall . . . . .                    | 2        |
| IP-Forwarding . . . . .               | 2        |
| Beispiel . . . . .                    | 2        |
| Software . . . . .                    | 2        |
| WLAN . . . . .                        | 2        |
| DHCP-Server . . . . .                 | 3        |
| Firewall . . . . .                    | 3        |
| Begriffe dieser Lerneinheit . . . . . | 3        |

## Einführung

Bei WLAN gibt es zwei Betriebsmodi:

1. Client
2. Access Point (AP)

Als Access Point konfiguriert, können sich Clients (beispielsweise Smartphones) beim eingebetteten System anmelden, vorausgesetzt sie kennen das notwendige Passwort.

**Hinweis:** Die Hardware des Raspberry Pi limitiert die Anzahl der gleichzeitig angemeldeten Clients. Bei einem Raspberry Pi 4 sind das (meiner Kenntnis nach) sieben.

Das eingebettete System kann als Insel-System betrieben werden oder – falls eine Verbindung ins Internet existiert – als Router den angemeldeten Clients einen Datenaustausch mit dem Internet ermöglichen.

## Netzwerk-Architektur

Falls das eingebettete System ein AP ist, spannt dieser in den meisten Fällen ein eigenes Netz auf, benötigt also eine Netzzadresse. Damit ist das Embedded System auch für die Verteilung der Hostadressen (innerhalb des Netzes) zuständig und benötigt hierfür einen Dienst, den so genannten **DHCP-Server**. Der Systemarchitekt entscheidet darüber, welche Hostadressen über den DHCP-Server **dynamisch** vergeben werden. Darüber hinaus bekommt das eingebettete System selbst eine **statische IP-Adresse**, beispielsweise die Hostadresse 1, also 192.168.42.1, wenn das Netz die Adresse 192.168.42.0 hat.

Statische respektive dynamische Hostadressen sind in einer Datei konfiguriert, beispielsweise der Datei `/etc/dnsmasq.conf`, wenn als DHCP-Server `dnsmasq` eingesetzt wird.

## Router

Dient der Access Point gleichzeitig als Router, stellt er also eine Verbindung zum Internet her, muss er auch mindestens zwei Netzwerkschnittstellen haben, die unabhängig voneinander betrachtet werden. Unser Raspberry Pi hat beispielsweise ein Ethernet-Interface und ein WLAN-Interface. Denkbar sind aber durchaus auch zwei WLAN-Interfaces.

Soll Routing ermöglicht werden, muss das zunächst im Linux-Kernel erlaubt werden. Außerdem sind die Routing-Tabellen zu erstellen. Klassischerweise wird eine Default-Route auf den Rechner im Netz konfiguriert. Dabei werden alle Pakete, die keinen Rechner im eigenen Netz (im Beispiel 192.168.42.0)

adressieren, an einen Rechner im zweiten Netz weitergereicht, der dort eine Gateway-Funktionalität übernimmt (das so genannte Default-Gateway).

## Firewall

Da typischerweise eine private Netzadresse für das WLAN verwendet wird, im Internet aber nur öffentliche Netzadressen geroutet werden, muss NAT (Network Address Translation) aktiviert werden. Und wenn man schon mal an der Firewall *rumschraubt*, können zur Steigerung der Sicherheit weitere Regeln hinzugefügt werden.

- Die Firewall hat hier die (eingeschränkte) Aufgabe, die privaten Netzadressen auf die öffentliche Netzadresse umzusetzen (Masquerading, NAT=Network Address Translation).
- Das Programm zum Setzen von Firewall-Regeln lautet bei Linux `iptables`.
- Zur Firewall-Konfiguration wird ein Init-Skript (`S99firewall`) erstellt, das beim Booten ausgeführt wird.
- Das Skript wird in das Verzeichnis `/etc/init.d` abgelegt.

```
init.d/S99firewall
#!/bin/sh

if [ "$1" == "start" ]; then
    iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
fi
echo "Firewall activated"
```

## IP-Forwarding

- Damit der Linux-Kernel überhaupt routet, muss IP-Forwarding aktiviert werden.
- Zur Aktivierung wird in die (virtuelle) Datei `/proc/sys/net/ipv4/ip_forward` eine “1” geschrieben werden.
- Dazu reicht es aus, eine Datei `/etc/sysctl.conf` mit dem untenstehenden Inhalt anzulegen.

```
sysctl.conf
# Enable IP forwarding.
net.ipv4.ip_forward = 1
```

## Beispiel

- Netzadresse WLAN: 192.168.42.0 (privates Netz)
- Hostadresse WLAN-Interface im embedded system (Accesspoint): 192.168.42.1 (private IP-Adresse)
- Hostadresse im per Ethernet angeschlossenen Netz: 194.94.121.156 (öffentliche Netzadresse)
- Gateway im öffentlichen Netz: 194.94.121.254
- Routing-Tabelle auf dem embedded System

| Netzadresse  | Gateway        | Interface |
|--------------|----------------|-----------|
| default      | 194.94.121.254 | eth0      |
| 194.94.121.0 | •              | eth0      |
| 192.168.42.0 | •              | wlan0     |

## Software

### WLAN

Für die Konfiguration und Verbindungsaufnahme mit einem WLAN wird in Linux das Werkzeug `wpa_supplicant` eingesetzt. Die Software unterstützt sowohl den Client- als auch den AP-Betriebsmode.

Alternativ kann für den AP-Betriebsmode auch `hostapd` verwendet werden.

## **DHCP-Server**

Als DHCP-Server wird im embedded Umfeld gerne `dnsmasq` eingesetzt.

## **Firewall**

`iptables`

## **Begriffe dieser Lerneinheit**

- Netzadresse
- Hostadresse
- DHCP-Server
- DHCP-Client
- Interfaces
- Access Point
- Router

*Embedded  
Systems -  
Cross-Compile  
Kernel and  
Userland*

Jürgen Quade,  
Hochschule  
Niederrhein

Cross-  
Entwicklung  
allgemein

Cross-  
Entwicklung  
Kernel

Cross-  
Entwicklung  
Userland

# *Embedded Systems - Cross-Compile Kernel and Userland*

Jürgen Quade, Hochschule Niederrhein

25.04.2022

## 1 Cross-Entwicklung allgemein

## 2 Cross-Entwicklung Kernel

## 3 Cross-Entwicklung Userland

*Embedded  
Systems -  
Cross-Compile  
Kernel and  
Userland*

Jürgen Quade,  
Hochschule  
Niederrhein

Cross-  
Entwicklung  
allgemein

Cross-  
Entwicklung  
Kernel

Cross-  
Entwicklung  
Userland

Prof. Dr.-Ing. Jürgen Quade  
Hochschule Niederrhein

## Section 1

Cross-Entwicklung allgemein

# Generelles

- Bei der Cross-Entwicklung kommen die Cross-Entwicklungswerkzeuge zum Einsatz.
- Die Entwicklungsumgebung ist/wird so gestrickt, dass die Wahl von Kompiler, Headerdateien etc. über Environmentvariablen erfolgt.
- Die Cross-Entwicklungswerkzeuge müssen von den Werkzeugen *gefunden* werden.
  - Anpassung der Variable PATH

# Generelles

Embedded  
Systems -  
Cross-Compile  
Kernel and  
Userland

Jürgen Quade,  
Hochschule  
Niederrhein

Cross-  
Entwicklung  
allgemein

Cross-  
Entwicklung  
Kernel

Cross-  
Entwicklung  
Userland

## Environmentvariablen

**ARCH** Name der Architektur, z.B. `ARCH=arm`

**CROSS\_COMPILE** Namenspräfix für die Cross-Werkzeuge, z.B.  
`arm-linux-`

**KERNEL** Kernelvariante für den Raspberry Pi (z.B.  
`kernel7` oder `kernel7l`)

**SYSROOT** Verzeichnis (Ort), wo der Compiler  
Target-Headerdateien und der Linker die  
Target-Bibliotheken findet.

**PATH** Damit die Cross-Werkzeuge gefunden werden muss  
die Environment-Variable PATH um den Pfad zu  
den Binaries ergänzt werden

# Tipp

Embedded  
Systems -  
Cross-Compile  
Kernel and  
Userland

Jürgen Quade,  
Hochschule  
Niederrhein

Cross-  
Entwicklung  
allgemein

Cross-  
Entwicklung  
Kernel

Cross-  
Entwicklung  
Userland

- Realisieren Sie das Setzen der Variablen über ein Skript, dass Sie in jeder beteiligten Shell aufrufen.
- Applikationen lassen sich per scp auf das Target kopieren und dort testen, bevor diese ins Rootfs-Image übernommen werden.

## Section 2

### Cross-Entwicklung Kernel

# Motivation

Embedded  
Systems -  
Cross-Compile  
Kernel and  
Userland

Jürgen Quade,  
Hochschule  
Niederrhein

Cross-  
Entwicklung  
allgemein

Cross-  
Entwicklung  
Kernel

Cross-  
Entwicklung  
Userland

Wir benötigen den Treiber für einen `brcmfmac` WLAN-Adapter.

## Realisierung

- In eingebetteten Systemen werden Treiber fest in den Kernel einkompiliert.
- Aufgrund der Abhängigkeiten müssen die zugehörigen Subsysteme ebenfalls als feste Kernelbestandteile einkompiliert werden.
- Feste Bestandteile des Kernels sind in der Konfiguration anhand eines „\*“ anstelle eines „M“ zu erkennen.

# Problem

Embedded  
Systems -  
Cross-Compile  
Kernel and  
Userland

Jürgen Quade,  
Hochschule  
Niederrhein

Cross-  
Entwicklung  
allgemein

Cross-  
Entwicklung  
Kernel

Cross-  
Entwicklung  
Userland

- Identifikation der benötigten Konfigurationsoptionen.

## Tipp zur Lösung des Problems

- Generieren Sie im ersten Schritt ein System, dass Module zur Laufzeit bedarfsabhängig lädt.
  - System benötigt erheblich mehr Speicherplatz
- Lassen Sie sich dann auf dem laufenden und funktionierendem System per `lsmod` die Liste der geladenen Kernelmodule ausgeben.
- Konfigurieren Sie die gelisteten Module fest in den Kernel und generieren den Kernel neu.

# Kernel (1)

- Environment-Variablen ARCH, CROSS\_COMPILE, KERNEL und PATH setzen
- KERNEL wird nur für die Generierung des Kernels benötigt.
  - kernel7 für einen Raspberry Pi 3
  - kernel7l für einen Raspberry Pi 4

# Automatisierung

Embedded  
Systems -  
Cross-Compile  
Kernel and  
Userland

Jürgen Quade,  
Hochschule  
Niederrhein

Cross-  
Entwicklung  
allgemein

Cross-  
Entwicklung  
Kernel

Cross-  
Entwicklung  
Userland

## Shellskript exports.sh

```
echo "preparing environment for this shell"
HOMEDIR=~`echo ~`  
  
export ARCH=arm
export CROSS_COMPILE=arm-linux-
export PATH="${PATH}: ${HOMEDIR}/embedded/raspi/\
    buildroot-2021.02.1/output/host/usr/bin/"
export KERNEL=kernel71
```

## Aufruf in der Shell

```
source exports.sh
```

# Kernel (2)

Für den Raspberry Pi nehmen wir einen vorgepatchten Kernel:

```
1 user@host:~/embedded/raspi$ git clone \
2     https://github.com/raspberrypi/linux.git
3 user@host:~/embedded/raspi$ source ./export.sh
4 user@host:~/embedded/raspi$ cd linux
5 user@host:~/embedded/raspi/linux$ git branch -a
6 user@host:~/embedded/raspi/linux$ git checkout rpi-5.3.y
7 user@host:~/embedded/raspi/linux$ make bcm2709_defconfig
8 user@host:~/embedded/raspi/linux$ time make -j6 zImage dtbs \
9     modules
```

# Kernel (3)

- Kernelkonfiguration `make . . .`

| Variante                              | KERNEL   | make                           |
|---------------------------------------|----------|--------------------------------|
| Pi 1, Zero, Zero W,<br>Compute Module | kernel   | <code>bcmrpidefconfig</code>   |
| Pi 2, 3, 3+, Compute<br>Module 3      | kernel7  | <code>bcm2709_defconfig</code> |
| Pi 4                                  | kernel7l | <code>bcm2711_defconfig</code> |

- Konfiguration über `make menuconfig` (im Linux-Quellcodeverzeichnis)

# Kernel (4)

Embedded  
Systems -  
Cross-Compile  
Kernel and  
Userland

Jürgen Quade,  
Hochschule  
Niederrhein

Cross-  
Entwicklung  
allgemein

Cross-  
Entwicklung  
Kernel

Cross-  
Entwicklung  
Userland

RPi4: Kernel ins tftp-Verzeichnis kopieren

```
cp arch/arm/boot/zImage /srvr/tftp/kernel71.img
```

Module ins Target-Verzeichnis kopieren

- Falls die Variante über Kernelmodule gewählt wird, müssen die Kernelmodule ins Rootfilesystem kopiert werden.

```
make INSTALL_MOD_PATH=~/embedded/raspi/target/ modules_install
```

# Kernel (5)

Embedded  
Systems -  
Cross-Compile  
Kernel and  
Userland

Jürgen Quade,  
Hochschule  
Niederrhein

Cross-  
Entwicklung  
allgemein

Cross-  
Entwicklung  
Kernel

Cross-  
Entwicklung  
Userland

## Integration

- Module müssen noch auf das Target kopiert werden
- Dazu wird auf post-build.sh des Buildroot-Generierungsprozesses zurückgegriffen.
- Bereits erstelltes Skript wird ergänzt.

### target/post-build.sh

```
# additions
echo "copying modules..."
cp -r ../../target/lib/modules/5.3.3-v7+/ ${TARGET_DIR}/lib/modules/
```

# Kernel (5)

Embedded  
Systems -  
Cross-Compile  
Kernel and  
Userland

Jürgen Quade,  
Hochschule  
Niederrhein

Cross-  
Entwicklung  
allgemein

Cross-  
Entwicklung  
Kernel

Cross-  
Entwicklung  
Userland

Buildroot-Konfiguration anpassen (wegen vieler, überflüssiger  
Module)

- Size 150M

`make menuconfig`

`[Filesystem images] [exact size] [150M]`

Neues Rootfilesystem erzeugen

`make`

## Section 3

### Cross-Entwicklung Userland

# Ziele

Embedded  
Systems -  
Cross-Compile  
Kernel and  
Userland

Jürgen Quade,  
Hochschule  
Niederrhein

Cross-  
Entwicklung  
allgemein

Cross-  
Entwicklung  
Kernel

Cross-  
Entwicklung  
Userland

- Einschränkungen bei der Entwicklung von Software für eingebettete Systeme kennen
  - programmtechnische Aspekte
  - entwicklungstechnische Aspekte
- Zentrale Systemfunktion (Systemcalls) kennen
- Programme für den Raspberry Pi erstellen können
- Kennenlernen der Cross-Generierung von Applikationen

## Praktisches Ziel

Cross-Generierung von „Hello World“ (C-Programm)

# Allgemeines (1)

*Embedded  
Systems -  
Cross-Compile  
Kernel and  
Userland*

Jürgen Quade,  
Hochschule  
Niederrhein

Cross-  
Entwicklung  
allgemein

Cross-  
Entwicklung  
Kernel

Cross-  
Entwicklung  
Userland

## Entwicklungstechnische Aspekte

- Für Programme in C oder C++ wird eine Cross-Entwicklungsumgebung benötigt.
- Programme müssen zum Target transferiert werden.

# Allgemeines (2)

Embedded  
Systems -  
Cross-Compile  
Kernel and  
Userland

Jürgen Quade,  
Hochschule  
Niederrhein

Cross-  
Entwicklung  
allgemein

Cross-  
Entwicklung  
Kernel

Cross-  
Entwicklung  
Userland

## Programmtechnische Aspekte

- Die Ausstattung mit Systemkomponenten (Bibliotheken, Middleware) ist eingeschränkt.
- Im Vergleich zum Entwicklungssystem stehen weniger Ressourcen (Hauptspeicher, Hintergrundspeicher, Rechenleistung) zur Verfügung.
- Stromverbrauch ist relevant!
- Zeitverhalten (Realzeitsystem)
- Direkte Hardwarezugriffe werden benötigt (Treiberprogrammierung)
- Safety und Security

# Entwicklungstechnische Konsequenzen

- Modularisierung
  - Lässt die Anforderung an Least Privilege (Entwurfsmuster für IT-Sicherheit) leichter erfüllen
  - Reduziert (auf Multicore) den Stromverbrauch
  - Bessere Kontrolle des Realzeitverhaltens durch Priorisierung
    - Reduktion der Latenzzzeiten
- Dateien statt Datenbanken
  - KISS
- Hardwarezugriffe im Treiber
  - Erhöht signifikant die Betriebssicherheit (Safety)
  - Zugriffe sind erheblich performanter
  - Reduziert den Stromverbrauch (z.B. durch Interruptbetrieb)

# Programmierschnittstelle

Embedded  
Systems -  
Cross-Compile  
Kernel and  
Userland

Jürgen Quade,  
Hochschule  
Niederrhein

Cross-  
Entwicklung  
allgemein

Cross-  
Entwicklung  
Kernel

Cross-  
Entwicklung  
Userland

## Wichtige Systemcalls

- `clock_nanosleep()` (`sleep()` und Freunde sind veraltet)
- `clock_gettime()`
- `sched_setscheduler()` (Priorität und Schedulingverfahren wählen)
- `pthread_create()`

# Hardwarezugriffe

Embedded  
Systems -  
Cross-Compile  
Kernel and  
Userland

Jürgen Quade,  
Hochschule  
Niederrhein

Cross-  
Entwicklung  
allgemein

Cross-  
Entwicklung  
Kernel

Cross-  
Entwicklung  
Userland

- Klassisch: open, close, read und write

## Hardwarezugriffe auf dem Raspberry Pi

- Zugriff erfolgt häufig über die Bibliothek WiringPi.
- Die Verwendung der WiringPi ist *unprofessionell*.
- Die Bibliothek mappt die Hardwareregister in den Adressraum der Applikation (Systemcall mmap)
  - Applikation greift direkt auf Hardware zu
  - Kein Schutz vor parallelen Zugriffen
- Professioneller Zugriff: **Gerätetreiber**

# Klassische Entwicklung im Linux-Umfeld

Embedded  
Systems -  
Cross-Compile  
Kernel and  
Userland

Jürgen Quade,  
Hochschule  
Niederrhein

Cross-  
Entwicklung  
allgemein

Cross-  
Entwicklung  
Kernel

Cross-  
Entwicklung  
Userland

- Entwickelt wird auf der Konsole
  - Editor
  - Compiler
  - Linker
  - Make
  - Debugger
- IDEs stehen zur Verfügung, bieten aber nicht alle Möglichkeiten

# Cross-Entwicklung (1)

- Ein Makefile sollte unabhängig von der Plattform gestaltet werden:
  - Applikation lässt sich für unterschiedliche Plattformen verwenden
- Das Makefile muss die Werkzeuge für die Zielplattform verwenden
  - Steuerung über Environmentvariablen
- Die Cross-Werkzeuge müssen die Target-Headerdateien und Target-Bibliotheken (Sysroot) verwenden
  - Pfad zur Sysroot ist typischerweise in die Werkzeuge eincompiliert
  - Pfad zur Sysroot kann über die Option --sysroot mitgegeben werden

# Cross-Entwicklung (2)

Embedded  
Systems -  
Cross-Compile  
Kernel and  
Userland

Jürgen Quade,  
Hochschule  
Niederrhein

Cross-  
Entwicklung  
allgemein

Cross-  
Entwicklung  
Kernel

Cross-  
Entwicklung  
Userland

## Environmentvariablen

**ARCH** Name der Architektur, z.B. `ARCH=arm`

**CROSS\_COMPILE** Namenspräfix für die Cross-Werkzeuge, z.B.  
`arm-linux-`

**SYSROOT** Verzeichnis (Ort), wo der Compiler  
Target-Headerdateien und der Linker die  
Target-Bibliotheken findet.

**PATH** Damit die Cross-Werkzeuge gefunden werden muss  
die Environment-Variable PATH um den Pfad zu  
den Binaries ergänzt werden

# Hello World (1)

Embedded  
Systems -  
Cross-Compile  
Kernel and  
Userland

Jürgen Quade,  
Hochschule  
Niederrhein

Cross-  
Entwicklung  
allgemein

Cross-  
Entwicklung  
Kernel

Cross-  
Entwicklung  
Userland

Ziel: Hello World für den Raspberry Pi cross-generieren

## Vorgehen

- ① Verzeichnis für Quellcode und Makefile  
`mkdir -p ~/embedded/application/hello-20210510`
- ② Quellcode erstellen
- ③ Makefile erstellen
- ④ Environmentvariablen setzen
- ⑤ Programm generieren
- ⑥ Programm auf das Target kopieren und testen

# Hello World (2)

## Makefile

```
1 CFLAGS=-sysroot $(SYSROOT) -g -Wall -static
2 LDLIBS=-lrt
3
4 CXX = $(CROSS_COMPILE)g++
5 CC = $(CROSS_COMPILE)gcc
6 AS = $(CROSS_COMPILE)as
7 AR = $(CROSS_COMPILE)ar
8 NM = $(CROSS_COMPILE)nm
9 LD = $(CROSS_COMPILE)ld
10 OJDUMP = $(CROSS_COMPILE)objdump
11 OBJCOPY = $(CROSS_COMPILE)objcopy
12 RANLIB = $(CROSS_COMPILE)ranlib
13 STRIP = $(CROSS_COMPILE)strip
14
15 all: hello
16
17 clean:
18     rm -rf *.o hello
```

# Hello World (3)

Embedded  
Systems -  
Cross-Compile  
Kernel and  
Userland

Jürgen Quade,  
Hochschule  
Niederrhein

Cross-  
Entwicklung  
allgemein

Cross-  
Entwicklung  
Kernel

Cross-  
Entwicklung  
Userland

```
1 #include <stdio.h>
2 #include <time.h>
3
4 int main( int argc, char **argv, char **envp )
5 {
6     struct timespec sleeptime;
7
8     sleeptime.tv_sec = 1;
9     sleeptime.tv_nsec= 0;
10    while ( 1 ) {
11        printf("Hello World\n");
12        clock_nanosleep( CLOCK_MONOTONIC, 0, &sleeptime, NULL );
13    }
14    return 0;
15 }
```

# Hello World (4)

- ① Environmentvariablen setzen
- ② Generieren

## Datei export.sh erstellen

```
export ARCH=arm
export CROSS_COMPILE=arm-linux-
export SYSROOT=~/embedded/raspi/buildroot-2021.02.1/\
               output/host/usr/arm-buildroot-linux-uclibcgnue
export PATH=$PATH:~/embedded/raspi/buildroot-2021.02.1/\
               output/host/usr/bin/
```

# Hello World (5)

Embedded  
Systems -  
Cross-Compile  
Kernel and  
Userland

Jürgen Quade,  
Hochschule  
Niederrhein

Cross-  
Entwicklung  
allgemein

Cross-  
Entwicklung  
Kernel

Cross-  
Entwicklung  
Userland

## Generierung

```
user@host:~/embedded/application/hello-20210510$ \
    source ./export.sh

quade@ezs-mobil:~/embedded/application/20191211hello$ make
    arm-linux-gcc --sysroot /home/quade/embedded/raspi/\
    buildroot-2021.02.1/output/host/usr/\
    arm-buildroot-linux-uclibcgnueabihf/sysroot/ \
    -g -Wall -static    hello.c -lrt -o hello
user@host:~/embedded/application/hello-20210510$ file hello
hello: ELF 32-bit LSB executable, ARM, EABI5 version \
    1 (SYSV), statically linked, with debug_info, not stripped

user@host:~/embedded/application/hello-20210510$ scp hello \
    root@192.168.11.19:
```

# Hello World (6)

Embedded  
Systems -  
Cross-Compile  
Kernel and  
Userland

Jürgen Quade,  
Hochschule  
Niederrhein

Cross-  
Entwicklung  
allgemein

Cross-  
Entwicklung  
Kernel

Cross-  
Entwicklung  
Userland

## Target

```
# cd
# pwd
/root
# ls -lrt
total 8
-rwx-----    1 root      root          6701 May  1 09:07 hello
# ./hello
Hello World
Hello World
Hello World
^C
#
```

# Linux handmade (3)

Jürgen Quade, Hochschule Niederrhein

5.04.2021

## Inhaltsverzeichnis

|                                                     |          |
|-----------------------------------------------------|----------|
| <b>Ziel der Übungseinheit</b>                       | <b>1</b> |
| <b>Hintergrund</b>                                  | <b>1</b> |
| Arbeitsschritte wiederholt (nur Userland) . . . . . | 1        |
| Vorbereitung . . . . .                              | 1        |
| Doing . . . . .                                     | 1        |
| Template <code>mkrootfs.sh</code> . . . . .         | 2        |
| <b>Aufgaben</b>                                     | <b>3</b> |
| (c) 2021, Alle Rechte vorbehalten                   |          |

## Ziel der Übungseinheit

Das Ziel dieser Übungseinheit besteht darin, die Erstellung des Userlands durch ein Skript zu automatisieren und damit Änderungen schneller durchführen zu können. Durch die Automatisierung ist die Generierung des Userlands weniger fehleranfällig.

### Literaturhinweis

Hintergrundinfos und Hinweise zu dieser Übungseinheit finden Sie im Buch *Embedded Linux lernen mit dem Raspberry Pi* auf Seite 51 ff.

## Hintergrund

Ein eingebettetes System besteht bekanntermaßen aus

1. dem Kernel
2. dem Userland
3. der Applikation (als Teil des Userlands)

Insbesondere der Bau des Userlands besteht aus vielen Arbeitsschritten. Einige Schritte, wie beispielsweise das Generieren der Busybox sind bei Änderungen am System immer wieder gleich, während andere Arbeitsschritte, zum Beispiel die Konfiguration der Busybox, unterschiedlich sind.

### Arbeitsschritte wiederholt (nur Userland)

#### Vorbereitung

1. Busybox downloaden
2. Dateistruktur anlegen
3. Applikationsdateien erstellen und ablegen

#### Doing

1. Imagedatei anlegen (`dd`)
2. Imagedatei formatieren (`mkfs.ext2`)
3. Busybox konfigurieren (`make menuconfig`)
4. Busybox compilieren und lokal (vor-)installieren

5. Imagedatei mounten (um **im** Image zu arbeiten)
6. Kommandos **in** das Image kopieren (**rsync**)
7. Targetverzeichnisse anlegen (**mkdir**)
8. Gerätedateien anlegen (**mknod**)
9. Applikationsdateien kopieren (**install**)
10. Applikation ins System einbinden
11. Besitzverhältnisse einstellen (**chown**)
12. Imagedatei aushängen (**umount**)

Von diesen Schritten sind die folgenden individuell:

- Konfiguration Kernel
- Konfiguration System
- Einbindung der Applikation ins System

Damit man nicht bei jeder kleinen Änderung am System jeden Schritt erneut durchführen muss, bietet sich die Automatisierung der Systemgenerierung durch ein einfaches Shellskript an.

Dabei müssen in so einem Build-Skript im Wesentlichen nur die Kommandos, die sonst auf der Konsole (im Terminal) eingegeben werden, abgelegt werden.

Damit ergibt sich der geänderte Arbeitsablauf eines Entwicklungszyklus (also einer Änderung am System):

1. Evtl. Änderungen in Busybox konfigurieren
2. Evtl. Applikation anpassen
3. Evtl. Skript anpassen
4. Build-Skript aufrufen
5. System testen

### Template **mkrootfs.sh**

Im Rahmen dieser Übung soll das nachfolgende Template ergänzt werden, so dass damit automatisiert ein Userland erstellt wird. Das Userland soll dann mit dem bereits erzeugten Kernel getestet werden.

```
#!/bin/bash

MAINDIR=~/embedded

cd $MAINDIR/qemu/userland

# Dateisystem anlegen

# Dateisystem formatieren

# Dateisystem einhängen (mounten)

# Verzeichnisstruktur im eingehängten Dateisystem (Image) anlegen

# Busybox generieren und vorab installieren

# Busyboxdateien ins Image installieren (kopieren)

# Gerätedateien im Image anlegen

# Applikationsdateien kopieren (Webserver)

# Zugriffsrechte anpassen

# MARK A

# MARK B

# MARK C
```

# MARD D

# *Image (Dateisystem) wieder aushaengen*

## Aufgaben

1. Laden Sie sich vom Lernserver das Gerüst des Generierungsskripts `mkrootfs.sh` zur automatisierten Erstellung des Userlands herunter. Das Skript geht davon aus, dass
  - das Basisverzeichnis in einer Shellvariablen `MAINDIR` abgelegt wird
  - es ein Verzeichnis `$MAINDIR/qemu/userland` gibt
  - die generelle Dateistruktur existiert, wie wir diese in der vorhergehenden Übung angelegt haben
  - Busybox im Verzeichnis `$MAINDIR/qemu/userland/busybox-1.32.1` (beziehungsweise entsprechend der verwendeten Busybox-Version) existiert.
2. Ergänzen Sie die einzelnen, kommentierten Schritte mit Ausnahme der mit MARK X bezeichneten Blöcke.
3. Bauen Sie ausreichend `echo`-Ausgaben ein, damit Sie den Fortschritt debuggen können.
4. Ändern Sie die Zugriffsrechte, so dass das Skript `mkrootfs.sh` ausgeführt werden kann.
5. Generieren Sie das System durch Aufruf von `./mkrootfs.sh`. Achten Sie auf Fehlerausgaben und korrigieren Sie eventuelle Fehler.
6. Testen Sie das generierte System mit Hilfe von Qemu.

# *Embedded Systems - Kernelprogrammierung*

Jürgen Quade, Hochschule Niederrhein

1.05.2022

1 API

2 Kontrollfluss

3 Kernel erweitern

4 Gerätetreiber

5 Datentransfer User-/Kernel-Space

6 Zusammenfassung

Prof. Dr.-Ing. Jürgen Quade

Hochschule Niederrhein

## Section 1

API

# Applikationsinterface für den Gerätezugriff

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

Zusammenfassung

- Interfaces für Gerät- und Dateizugriff sind identisch
- Pro Gerät mindestens eine Datei im Dateisystem vorhanden  
(so genannte *Gerätedatei*)
  - Blockdevice
  - Characterdevice

## Systemfunktionen

- |                                                                                                       |                                                                                                                             |
|-------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"><li>• open</li><li>• close</li><li>• read</li><li>• write</li></ul> | <ul style="list-style-type: none"><li>• ioctl</li><li>• select/poll</li><li>• fcntl</li><li>• seek</li><li>• nmap</li></ul> |
|-------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|

# Open/Close

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

Zusammenfassung

## Abläufe beim *Öffnen* des Gerätes

- Check: Ist die Datei vorhanden respektive ist der benötigte Treiber geladen?
- Check: Stimmen die Zugriffsrechte?
- Check: Welcher Zugriff wird verlangt?
- Rückgabe: Filedeskriptor als Zugriffsschlüssel

## Aufräumen

- Freigeben der bei `open` allozierten Ressourcen
- Nach dem `close` befindet sich das Gerät wieder in einen definierten Zustand

# Read

## Prototyp

```
ssize_t read( int fd, char *buffer,  
              size_t max_bytes_to_read );
```

## Aufgabe

- Transfer von Daten aus einer Datei oder aus der Hardware in den Speicher der Applikation

## Returnwert

- Anzahl der gelesenen Bytes (Minimum 1 Byte)
- Fehlercode (-EINTR, -EAGAIN, ...)

# Write

## Prototyp

```
ssize_t write (int fd, char *buffer,  
              size_t max_bytes_to_write );
```

## Aufgabe

- Transfer von Daten aus der Applikation in eine Datei oder auf die Hardware

## Returnwert

- Fehlerfreier Fall
  - Anzahl der geschriebenen Bytes
- Fehlerfall
  - Fehlercode (z.B. -EINTR ,Unterbrechung per Signal)

## Hinweis

- Soll eine definierte Anzahl Bytes geschrieben werden, muss das in einer Schleife programmiert werden

## Prototyp

```
int ioctl( int fd, int command, .... );
```

## Aufgabe

- **Universalfunktion** zur Kommunikation mit dem Treiber
- Unterstützt frei definierbare Kommandos mit frei definierbaren Parametern
  - Beispiel: ioctl zum *atomic write-read*
- Kommandos und Parameter werden durch den Treiber festgelegt.

# Mmap / Munmap

## Prototype

```
void * mmap(void *start, size_t length, int prot,  
           int flags, int fd, off_t offset);  
int munmap(void *start, size_t length);
```

## Aufgabe

- Funktion, um Speicherbereiche eines Geräte (Hardware) in den Adressraum einer Applikation einzublenden.
- Damit kann die Applikation performant (ohne Übergang in den Kernel-Mode) auf Hardware (-Register) zugreifen. (Beispiel: xorg).

# Select/poll

## Prototyp

```
int select( int n, fd_set *read_fds, fd_set *write_fds,  
            fd_set *exception_fds,  
            struct timeval *timeout );
```

## Aufgabe

- Funktion um festzustellen, ob an einem oder an mehreren Geräten (Dateien oder Netzverbindungen) Daten zum Lesen bereit liegen bzw. Daten geschrieben werden können.
- Der Aufruf erfolgt zeitüberwacht.
- Mit dieser Systemfunktion kann eine Applikation ohne zu pollen mehrere Kanäle (Dateien) auf Ein- oder Ausgaben überwachen.

## Hinweise

Über die Makros FD\_SET, FD\_ZERO und FD\_ISSET können die Datenstrukturen „read\_fds“, „write\_fds“ und „exception\_fds“

# Select/Poll (Beispiel)

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

Zusammenfassung

```
#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

int main(void)
{
    fd_set rfds;
    struct timeval tv;
    int retval;

    FD_ZERO(&rfds);
    FD_SET(0, &rfds);
    /* Wait up to five seconds. */
    tv.tv_sec = 5;
    tv.tv_usec = 0;
    retval = select(1, &rfds, NULL, NULL, &tv);
    /* Don't rely on the value of tv now! */
    ... // next slide
```

# Select/Poll (Beispiel)

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

Zusammenfassung

```
    ...
    if (retval)
        printf("Data is available now.\n");
        /* FD_ISSET(0, &rfds) will be true. */
    else
        printf("No data within five seconds.\n");
    exit(0);
}
```

# Lseek

- Mit dieser Funktion kann in einer Datei (bzw. bei einem Blockdevice) gezielt ein bestimmtes Byte gelesen bzw. geschrieben werden.
- Die Positionierung kann relativ
  - zum Dateianfang
  - zum Dateiende
  - zur aktuellen Position erfolgen.

# Beispielapplikation: Auslesen des Bootsektors

```
int main( int argc, char **argv, char **envp )
{
    int i, j, fd;
    unsigned char buffer[512];
    if ((fd=open("/dev/sda", O_RDONLY))<0) {
        perror( "/dev/sda" );
        exit( -1 );
    }
    if (read( fd, buffer, sizeof(buffer)) != sizeof(buffer)) {
        perror( "read" );
        exit( -2 );
    }
    for (i=0; i<sizeof(buffer)/16; i++) {
        for( j=0; j<16; j++ )
            printf( "%2.2x ",buffer[i*16+j] );
        printf("\n");
    }
    close( fd );
    return 0;
}
```

# Beispielapplikation: Druckerausgabe

```
int main( int argc, char **argv, char **envp )
{
    int fd;
    char *ptr="aaaaaa";

    if ((fd=open(„/dev/lp0“,0_WRONLY))<0) {
        perror(„/dev/lp0“);
        exit(-1);
    }
    if (write(fd,ptr,strlen(ptr))<strlen(ptr)) {
        perror(„write“);
    }
    close( fd );
    return 0;
}
```

## Section 2

### Kontrollfluss

# Blocking

- Für den Zugriff auf ein Gerät gibt es zwei Zugriffsarten:
  - blocking mode
  - non blocking mode
- Beim „blocking mode“ wird eine Applikation in den Zustand „schlafend“ versetzt, falls beim read-Aufruf angeforderte Daten noch nicht zur Verfügung stehen.
- Beim „blocking mode“ wird eine Applikation in den Zustand „schlafend“ versetzt, falls bei einem write-Aufruf auszugebende Daten noch nicht geschrieben werden können.

# Beispiel Blocking

```
#include <stdio.h>

int main( int argc, char **argv, char **envp )
{
    int fd=0; // stdin, standardmaessig im blocking mode
    char buffer[512];

    printf("Blocking-Mode: WARTEN AUF EINGABE ... \n");
    read( fd, buffer, sizeof(buffer) ); // blockierendes Warten
    printf("Eingabe getaetigt... \n");
    return 0;
}
```

# Non-Blocking

- Beim „non-blocking mode“ kommt der Aufruf (Systemcall) direkt zurück, auch wenn keine Daten gelesen bzw. geschrieben werden konnten.
- Dieser Fall ist am Fehlercode `-EAGAIN` erkennlich.
- Der Zugriffsmode wird beim Zugriff auf Dateien entweder
  - beim Öffnen angegeben oder
  - nach dem Öffnen mit der Systemfunktion `fcntl`

# Beispiel Non-Blocking

```
#include <stdio.h>
#include <fcntl.h>

int main( int argc, char **argv, char **envp )
{
    int fd=0; // stdin, standardmaessig im blocking mode
    int ret;
    char buffer[512];

    fcntl( fd, F_SETFL, O_NONBLOCK );
    printf("Nonblocking-Mode: KEIN WARTEN\n");
    ret=read( fd, buffer, sizeof(buffer) );
    printf("Returnwert: %d\n", ret);
    return 0;
}
```

# Zusammenfassung

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

Zusammenfassung

- Das einheitliche Interface für den Zugriff auf Dateien oder auf Geräte entlastet den Programmierer.
- Der Datenfluss ist relativ simpel programmiert.
- Der Kontrollfluss (blocking/non blocking) ist wesentlich komplexer.
- Iocontrols bieten dem Treiberentwickler sehr viel Freiheit, sind aber fehlerträchtig und sollten vermieden werden.

## Section 3

### Kernel erweitern

## Literatur

Quade/Kunst: Linux-Treiber entwickeln. 3. Auflage,  
dpunkt-Verlag, 2011, S. 76-80.

# Bevor es losgeht ...

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

Zusammenfassung

## Voraussetzungen für die Kernelprogrammierung

- Kernelquellen (normal unter `/usr/src/linux/`)
- passende Kernelkonfiguration
- Programme `insmod`, `rmmod`

# Kernel-Quellen

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

Zusammenfassung

- Download über <https://www.kernel.org>
- Alternativ per git (aktuellen Linus-Kernel)

```
git clone git://git.kernel.org/pub/scm/linux/kernel/\n    git/torvalds/linux.git
```

# Aufbau Kernel-Quellen

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

Zusammenfassung

- arch: architekturspezifischer Code
- block: Blockgeräte-Subsystem
- certs: Zertifikate für die digitale Signatur
- crypto: Verschlüsselungsalgorithmen
- Documentation: Dokumentation
- drivers: Gerätetreiber
- fs: Dateisysteme
- include: Include-Dateien
- init: Initialisierung des Kernels
- ipc: Inter Process Communication

# Aufbau Kernel-Quellen (Fortsetzung)

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

Zusammenfassung

- kernel: Kern des Kerns
- lib: oft benötigte Hilfsfunktionen
- mm: Speicherverwaltung
- net: Netzwerk-Code
- scripts: Hilfsskripte zur Generierung
- security: Sicherheitsmodule
- sound: Audio Unterstützung
- usr: User-Software des Init-Filesystems
- virt: Virtualisierung

# Kernel-Module

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

Zusammenfassung

- Treiber sind (meist) Module
- Module werden *dynamische* (zur Laufzeit) geladen
  - geringer Ressourcenverbraucht im Kernel
  - einfache Treiberentwicklung ohne ständige Neustarts

## Kommandos

- `insmod`: Laden eines Kernel-Moduls
- `rmmmod`: Entladen
- `lsmod`: Anzeigen der geladenen Module

# Modul-Code

```
#include <linux/version.h>
#include <linux/module.h>

int init_module(void)
{
    printk("init_module called\n");
    return 0;
}

void cleanup_module(void)
{
    printk("cleanup_module called\n");
}
MODULE_LICENSE("GPL");
```

# Aufgabe der Funktion init\_module

- Modul bei Kernel-Subsysteme anmelden
  - IO-Subsystem (Treiber)
  - PCI
  - Sys-Filesystem
  - proc-Filesystem
  - ...
- Initialisierung von Objekten (z.B. Waitqueues)
- Reservieren von Ressourcen

# Aufgabe der Funktion cleanup\_module

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

Zusammenfassung

- Freigeben der allozierten Systemressourcen
- Abmelden des Moduls beim System

# Lizenzen

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

Zusammenfassung

- Module müssen angeben, welcher Lizenz sie unterliegen.
- Dazu dient das Makro `MODULE_LICENSE()`
- Folgende Lizenzen stehen zur Auswahl:
  - „GPL“
  - „GPL and additional rights“
  - „Dual BSD/GPL“
  - „Dual MPL/GPL“
  - „Proprietary“

## Konsequenz

Die Wahl der Lizenz entscheidet darüber, auf welche Funktionen des Kernels ein Modul zugreifen darf

# Lizenzen (Fortsetzung)

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

Zusammenfassung

- Besitzt ein Modul zwei Lizenzen, zählt die GPL.
- Das Kommando `modinfo` zeigt die Lizenz an.
- Fehler, die aus einem proprietären Modul herrühren, werden nicht verfolgt.
- Hersteller können sich auf Fehler ihrer Module beschränken.

## Beispiel

```
modinfo -l modulename.ko
```

# Exkurs: Debugausgaben

- Im Kernel gibt es die zu `printf()` korrespondierende Funktion `printk()`
- Ausgaben von `printk()` werden im Syslog protokolliert
- Die verschiedenen Syslog-Level ( `LOG_ALERT`, `LOG_INFO`, `LOG_NOTICE`, `LOG_DEBUG`,... ) werden über die drei ersten Zeichen des Formatstrings gesteuert

# Exkurs: Debugausgaben

| Symbol       | Wert | Bedeutung                               |
|--------------|------|-----------------------------------------|
| KERN_EMERG   | <0>  | Das System ist nicht mehr zu gebrauchen |
| KERN_ALERT   | <1>  | Sofortige Maßnahmen sind erforderlich   |
| KERN_CRIT    | <2>  | Der Systemzustand ist kritisch          |
| KERN_ERR     | <3>  | Fehlerzustände sind aufgetreten         |
| KERN_WARNING | <4>  | Warnung                                 |
| KERN_NOTICE  | <5>  | Wichtige Nachricht, kein Fehler         |
| KERN_INFO    | <6>  | Zur Information                         |
| KERN_DEBUG   | <7>  | Debug-Information                       |

# Exkurs: Debugausgaben

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

Zusammenfassung

## Beispiel

```
printf("<7>user will %d bytes kopieren.\n", count );
printf(KERN_DEBUG "user will %d bytes kopieren.\n",count);
```

# Exkurs: Debugausgaben (Fortsetzung)

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

Zusammenfassung

- Über `pr_debug()` und `pr_info()` sind übersichtliche Ausgaben (bei bedingter Compilierung) möglich:

## Der Code

```
pr_debug( "Lesefifo=%d\n", fifoindex );
```

entspricht

```
#ifdef DEBUG
printf( KERN_DEBUG "Lesefifo=%d\n", fifoindex );
#endif
```

# Exkurs: Debugausgaben (Fortsetzung)

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

Zusammenfassung

## Der Code

```
pr_info( "Treiber erfolgreich geladen.\n");
```

## entspricht

```
printf( KERN_INFO "Treiber erfolgreich geladen.\n");
```

# Debugausgaben (Fortsetzung)

Zusätzlich gibt es noch Funktionen, die einen Stacktrace ausgeben:

- `WARN()` ; -> Code wird weiterbearbeitet.
- `WARN_on( bedingung )` ; -> Code wird weiterbearbeitet.
- `WARN_on_once(bedingung)` ;-> Code wird weiterbearbeitet.
- `BUG()` ; -> Code wird abgebrochen.
- `BUG_ON( bedingung )` ; -> Code wird abgebrochen.

# Makefile

```
ifneq ($(KERNELRELEASE),)  
obj-m := mod1.o  
  
else  
KDIR := /lib/modules/$(shell uname -r)/build  
PWD := $(shell pwd)  
  
default:  
        $(MAKE) -C $(KDIR) M=$(PWD) modules  
endif
```

# Modul Test

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

Zusammenfassung

The screenshot shows a terminal window titled "root@ezs-mobil: /tmp/treiber - Befehlsfenster - Konsole". The terminal displays the following command sequence:

```
root@ezs-mobil:/tmp/treiber # make
make  -C /lib/modules/2.6.13-PM/build M=/tmp/treiber modules
make[1]: Entering directory `/usr/src/linux-2.6.13'
  CC [M]  /tmp/treiber/mod1.o
  Building modules, stage 2.
  MODPOST
  LD [M]  /tmp/treiber/mod1.ko
make[1]: Leaving directory `/usr/src/linux-2.6.13'
root@ezs-mobil:/tmp/treiber # insmod mod1.ko
root@ezs-mobil:/tmp/treiber # lsmod | head -n 5
Module           Size  Used by
mod1            1216   0
af_packet       23176   0
i915            20288   2
drm             68692  3 i915
root@ezs-mobil:/tmp/treiber # tail -f -n 5 /var/log/messages
Sep 14 20:26:16 localhost kernel: init_module called
Sep 14 20:27:31 localhost kernel: cleanup_module called
Sep 14 20:27:52 localhost kernel: init_module called
Sep 14 20:28:10 localhost kernel: cleanup_module called
Sep 14 20:28:26 localhost kernel: init_module called
root@ezs-mobil:/tmp/treiber # rmmod mod1
root@ezs-mobil:/tmp/treiber #
```

Annotations with blue arrows point to specific lines of text:

- An arrow points to the line "root@ezs-mobil:/tmp/treiber # make" with the label "Modul generieren".
- An arrow points to the line "root@ezs-mobil:/tmp/treiber # insmod mod1.ko" with the label "Modul laden (Extension „.ko“)".
- An arrow points to the line ".ko=kernel object" with the label ".ko=kernel object".
- An arrow points to the line "Sep 14 20:26:16 localhost kernel: init\_module called" with the label "Aktivität im Syslog verifizieren".
- An arrow points to the line "root@ezs-mobil:/tmp/treiber # rmmod mod1" with the label "Modul entladen".

# Linux Kernelmodul-Template

```
#include <linux/module.h>
#include <linux/init.h>

static int __init mod_init(void)
{
    printk("mod_init called\n");
    return 0;
}

static void __exit mod_exit(void)
{
    printk("mod_exit called\n");
}

module_init( mod_init );
module_exit( mod_exit );

...
```

# Linux Kernemodul-Template (Fortsetzung)

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

Zusammenfassung

```
// Metainformation
MODULE_AUTHOR("Juergen Quade");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Just a Modul-Template, without specific funct
MODULE_SUPPORTED_DEVICE("none");
```

## Section 4

### Gerätetreiber

# Der Treiber ...

- ist im Kernel als ein „Objekt“ repräsentiert:
- Instantiierung der `struct file_operations`
- Registrieren des Objekts beim Chardev-Subsystem.

## Grundprinzip

Für jede Applikationsfunktion gibt es eine Funktion (Methode) im Kernel-Treiber-Objekt:

- `open - driver_open`
- `close - driver_close`
- `read - driver_read`
- `write - driver_write`
- ...

# Zugriff auf den Treiber durch die Applikation

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

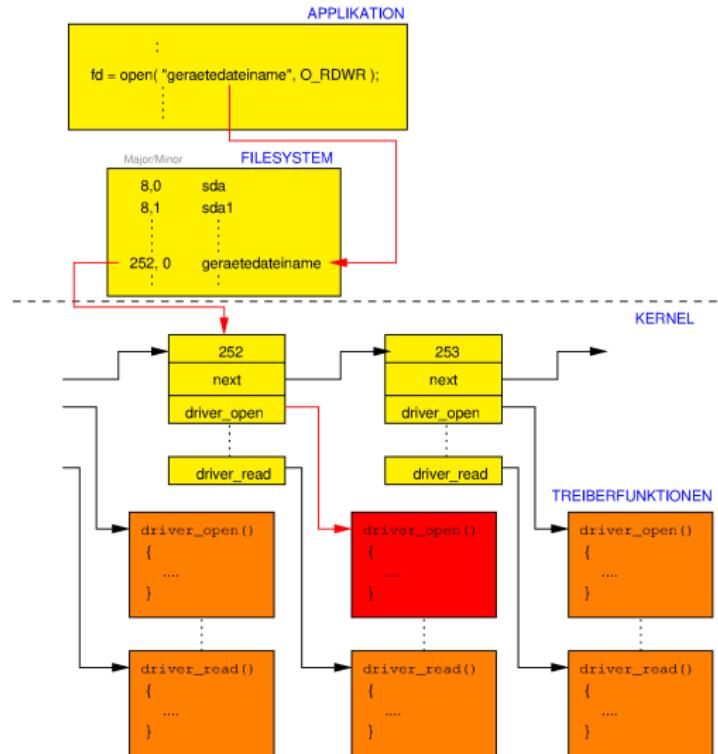
Zusammenfassung

- Ruft die Applikation `read()` auf, wird im Kernel die zugehörige `driver_read()`-Funktion aufgerufen.
- `driver_read()` führt den Datentransfer durch:
  - Lesen der Daten von der Hardware.
  - Kopieren der gelesenen Daten in den Speicherbereich der Applikation

## PROBLEMATIK

- Viele Treiber – viele Funktionen `driver_read()`
- Die *richtige* Funktion muss ausgewählt werden...

# Auswahl über Gerätedatei



# Treiber-Identifikation

- Der Treiber meldet sich beim IO-Subsystem unter einer eindeutigen Kennung (Major-Nummer) an.
- Auf der User-Ebene gibt es eine Zuordnung zwischen dieser Major-Nummer und einem (Geräte-) Dateinamen (Attribut der Datei).
- Im Kernel sind Major- und Minornummern durch eine 32-Bit breite Gerätenummer ersetzt worden.

# Majornumber

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

Zusammenfassung

- Gerätedateien werden „attributiert“ mit
  - Art der Gerätedatei (Character- oder Blockdevice)
  - Majornumber (zur Identifikation des Treibers)
  - Minornumber (zur Unterscheidung in einzelne logische Geräte)

## Kommando

```
mknod MyDeviceName c Major Minor
```

# Majornumber

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

Zusammenfassung

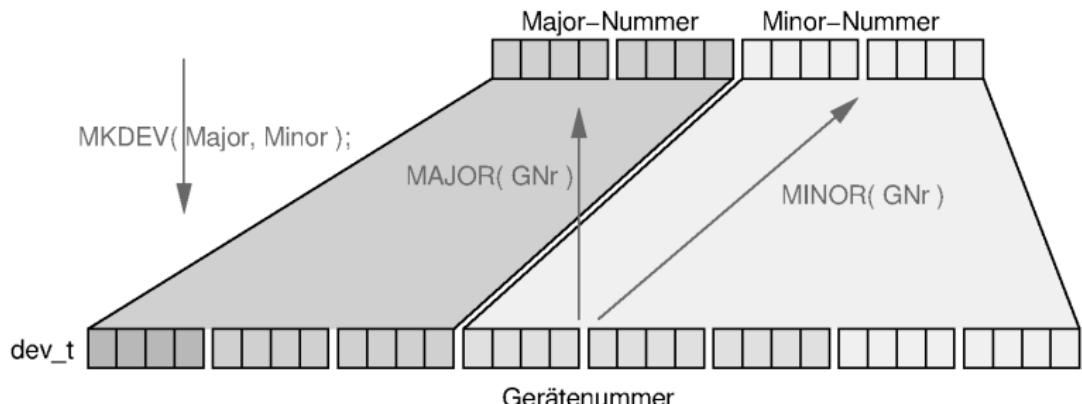
## Problematik

Für die Major-Nummer stehen nur 8 Bit zur Verfügung, also gibt es maximal 256 unterschiedliche Geräte (zu wenig).

## Abhilfe

- Gerätenummern
- dynamische Vergabe der Major-Nummer und dynamische Generierung eines Gerätedateieintrags.

# Abbildung auf Gerätenummern



# Minornummer

- Kodierung zusätzlicher Informationen
  - Funktionalitäten (z.B. die Art des Handshaking bei der seriellen Schnittstelle)
  - Zugriffsbereiche (z.B. die einzelnen Partitionen einer Festplatte)
  - Auswahl einer spezifischen Hardware (z.B. ob die 1. oder die 2. serielle Schnittstelle verwendet werden soll)
  - Ein Treiber für mehrere (gleiche respektive ähnliche) Geräte.
- Logische Sicht auf reale oder virtuelle Geräte.
- Belegung/Bedeutung bei Character-Devices frei.
- Belegung/Bedeutung bei Block-Devices vorgegeben.

# Minornummer (Fortsetzung)

- Zugriff auf die Minor-Nummern per Makro:
  - MINOR() : über struct device
  - iminor() : über „struct inode“
- struct inode wird driver\_open() und driver\_close() als Parameter übergeben
  - minor=iminor( geraetedatei );
- driver\_read(), driver\_write() bekommen nur struct file übergeben:
  - struct file enthält einen Verweis auf struct inode
  - minor=iminor( instance->f\_dentry->d\_inode );

# Treiber anmelden

- Gerätenummern-Bereich reservieren.
  - `alloc_chrdev_region()`
  - `register_chrdev_region()`
- Treiber beim Kernel anmelden.
  - `cdev_alloc()` – Objekt reservieren
  - `cdev_add()` – Anmeldung (Objekt instanzieren)
- Sysfs-Verzeichnis erstellen, damit automatisiert durch Udev Gerätedateien angelegt werden.

# Vom Modul zum Treiber ...

```
if( alloc_chrdev_region(&template_dev_number, 0, 1, TEMPLATE) < 0
    return -EIO;
driver_object = cdev_alloc(); /* Anmeldeobjekt reservieren */
if( driver_object==NULL )
    goto free_device_number;
driver_object->owner = THIS_MODULE;
driver_object->ops = &fops;
if( cdev_add(driver_object,template_dev_number, 1) )
    goto free_cdev;
/* Eintrag im Sysfs, damit Udev den Gerätetypeneintrag erzeugt */
template_class = class_create( THIS_MODULE, MY_CLASS_NAME );
device_create( template_class, NULL, template_dev_number,
              NULL, "%s", MY_DEVICE_FILE_NAME );
return 0;
free_cdev:
kobject_put( &driver_object->kobj );
free_device_number:
unregister_chrdev_region( template_dev_number, 1 );
return -EIO;
```

# Vom Modul zum Treiber ...

```
static void __exit template_exit( void )
{
    device_destroy( template_class, template_dev_number );
    class_destroy( template_class );
    /* Abmelden des Treibers */
    cdev_del( driver_object );
    unregister_chrdev_region( template_dev_number, 1 );
    return;
}
```

# Gerätedatei automatisiert anlegen

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

Zusammenfassung

- Die Gerätedatei kann über Udev automatisiert angelegt werden.
  - Diese Technik erleichtert den Umgang mit vom Kernel vergebenen Majornummern.
- Dazu muss sich der Treiber bei einer Gerätekasse über das Gerätmodell (Sys-Filesystem) anmelden.
  - Das Gerätmodell erwartet eine Gerätenummer.
- Beim Abmelden vom Gerätmodell wird die Gerätedatei wieder entfernt.

# Treiberarchitektur

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

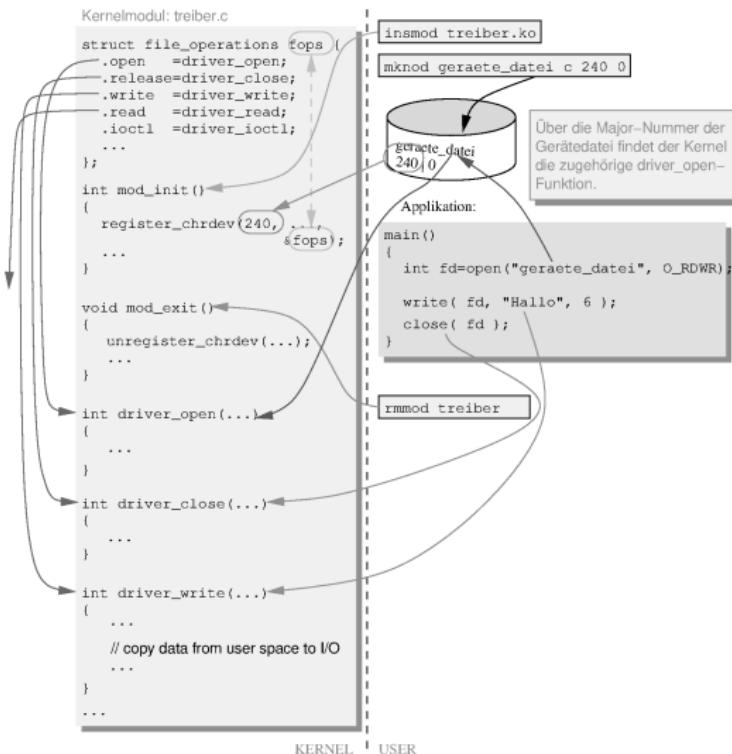
Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

Zusammenfassung



# Open-Funktion im Treiber

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

Zusammenfassung

- Zugriffsüberwachung
- Initialisierung

## Rückgabewert

- 0: Zugriff gestattet
- -EIO, -EBUSY, ...: Zugriff verweigert

# Open-Funktion im Treiber

- Überprüfen von dedizierten Zugriffsrechten (z.B. Sicherstellen, dass zu einem Zeitpunkt nicht mehr als ein Prozess bzw. eine Treiberinstanz auf den Treiber zugreifen).
- Parameter: `struct inode` und `struct file`
- Allokation und Initialisierung von (Hardware-) Ressourcen.
- Funktion gibt 0 zurück, wenn das Gerät geöffnet werden darf, ansonsten einen Fehlercode ( `<asm/errno.h>` ).

# driver\_open()

## Einfaches Beispiel

```
static int driver_open( struct inode *geraetedatei,
    struct file *instanz )
{
    return 0;
}
```

# driver\_open()

```
static int write_count=0;  
...  
static int driver_open( struct inode *geraedatei,  
                      struct file *instanz )  
{  
    if( instanz->f_flags&O_RDWR || instanz->f_flags&O_WRONLY ) {  
        if( write_count > 0 ) {  
            return -EBUSY;  
        }  
        write_count++;  
    }  
    return 0;  
}
```

## Spoiler

Obige Funktion ist **unprofessionell** und besitzt eine  
*Race-Condition!*

# driver\_close()

## Aufgabe

- Ressourcenfreigabe
- Überführen der Hardware in den sicheren Zustand.

## Rückgabewert

0

## Beispiel

```
static int driver_close( struct inode *geraetedatei,
    struct file *instanz )
{
    if( instanz->f_flags&0_RDWR || instanz->f_flags&0_WRONLY ) {
        write_count--;
    }
    return 0;
}
```

# driver\_read()

## Aufgabe

- Datentransfer zwischen Kernel- und User-Space.
- Zugriff auf (eventuell vorhandene) Hardware.

## Rückgabewert

- Anzahl der transferierten Bytes.
- Alternativ: Fehlercode.

## Funktion zum Kopieren von Daten

```
copy_to_user( char *to, char *from, int count )
```

# driver\_read()

## Einfaches Code-Beispiel

```
static ssize_t driver_read( struct file *instanz,
    char *user, size_t count, loff_t *offset )
{
    int not_copied, to_copy;

    to_copy = strlen(hello_world)+1;
    to_copy = min( to_copy, count );
    not_copied=copy_to_user(user,hello_world,to_copy);
    *offset = *offset + (to_copy-not_copied);
    return to_copy-not_copied;
}
```

## Section 5

### Datentransfer User-/Kernel-Space

# Datentransfer zwischen User- und Kernel-Space

- Zugriff auf den User-Space ist vom Kernel aus nicht direkt möglich.

## Funktionen zum Datentransfer

```
copy_to_user( to, from, len );  
copy_from_user( to, from, len );  
get_user(variable, source );  
put_user( variable, source );
```

Zusätzlich gibt es Funktionen für den Zugriff ohne  
Zugriffsschutz... ( \_\_copy\_to\_user() ... )

# Treiber-Write / Treiber-Read

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

Zusammenfassung

## Beschreibung der Parameter

- `struct file *instanz`: Info über die Treiberinstanz
- `char *user`: Speicherbereich im User-Space
- `size_t count`: Größe des Buffers im User-Space
- `loff_t offs`: Offset

## Returnwert

Anzahl der kopierten Bytes oder Fehlercode (negativer Wert)

# driver\_write()

## Aufgabe

- Datentransfer zwischen User- und Kernel-Space.
- Zugriff auf (eventuell vorhandene) Hardware.

## Rückgabewert

- Anzahl der transferierten Bytes.
- Alternativ: Fehlercode.

## Datenkopier-Funktion

```
copy_from_user( char *to, char *from, int count )
```

# driver\_write() (Fortsetzung)

## Einfaches Codebeispiel

```
static ssize_t driver_write( struct file *instanz,
    char __user *buffer, size_t count, loff_t *offset )
{
    return count;
}

static struct file_operations Fops = {
    .owner = THIS_MODULE,
    .write = driver_write,
};
```

# driver\_write() (Fortsetzung)

## Codebeispiel (a littlebit more sophisticated)

```
static ssize_t driver_write( struct file *instanz, const char *us
                           size count, loff_t *offs )
{
    int to_copy;
    int not_copied;

    printk("DriverWrite called\n");
    to_copy = min( count, sizeof(fifo_buf) );
    not_copied = copy_from_user( fifo_buf, user, to_copy );
    *offset = *offset + (to_copy-not_copied);
    printk("%s", fifo_buf );
    return to_copy-not_copied;
}
```

# Treiber Test

Embedded  
Systems -  
Kernelpro-  
grammierung

Jürgen Quade,  
Hochschule  
Niederrhein

API

Kontrollfluss

Kernel  
erweitern

Gerätetreiber

Datentransfer  
User-/Kernel-  
Space

Zusammenfassung

```
$ make
$ cat /etc/motd > /dev/mydevice
$ tail -f /var/log/kern.log
```

## Section 6

### Zusammenfassung

# Zusammenfassung

- Die Systemcalls `read()` und `write()` triggern im Treiber die Funktionen `driver_read()` und `driver_write()`.
- Die Funktionen geben die Anzahl der transferierten Bytes zurück.
- Zum Datentransfer zwischen Kernel- und User-Space werden die Funktionen `copy_to_user()` und `copy_from_user()` eingesetzt.
- Logische Geräte werden innerhalb der Lese- oder Schreibfunktion über die Minornummer identifiziert.

*Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi*

Jürgen Quade,  
Hochschule  
Niederrhein

Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen

# *Embedded Systems - Kernelprogrammierung - Zugriffsmodi*

Jürgen Quade, Hochschule Niederrhein

2.05.2022

## 1 Zugriffsmodi

## 2 Hardware Anbindung

## 3 Ressourcen Verwaltung

## 4 GPIOs

## 5 Treiberinstanzen

*Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi*

Jürgen Quade,  
Hochschule  
Niederrhein

Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen

**Prof. Dr.-Ing. Jürgen Quade**

**Hochschule Niederrhein**

## Section 1

### Zugriffsmodi

*Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi*

Jürgen Quade,  
Hochschule  
Niederrhein

Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen

## Literatur

Quade/Kunst: Linux-Treiber entwickeln. 3. Auflage,  
dpunkt-Verlag, 2011, S. 106.

# Zugriffsmodi im Treiber

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi

Jürgen Quade,  
Hochschule  
Niederrhein

Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen

## Blockierend

Wenn per `read` angeforderte Daten (zur Zeit) nicht verfügbar sind, **schläft** die Applikation, bis die Daten vorliegen.

Wenn per `write` zu schreibende Daten (zur Zeit) nicht geschrieben werden können, **schläft** die Applikation, bis die Daten geschrieben werden können.

## Nicht blockierend

Wenn per `read` angeforderte Daten (zur Zeit) nicht verfügbar sind, wird die Anwendung darüber informiert (Fehlercode `-EAGAIN`).

Wenn per `write` zu schreibende Daten (zur Zeit) nicht geschrieben werden können, wird die Anwendung darüber informiert (Fehlercode `-EAGAIN`).

# Realisierung im Treiber

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi

Jürgen Quade,  
Hochschule  
Niederrhein

Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen

## Non-blocking

```
// no data available
// non blocking mode return
if( TreiberInstanz->f_flags & O_NONBLOCK )
    return -EAGAIN;
```

## Blocking

```
// no data available
// blocking mode -> sleep
if( !(TreiberInstanz->f_flags & O_NONBLOCK) )
    wait_event_interruptible( &wq_read, condition ); // sleep
// data available
...
```

# Taskzustand ändern

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi

Jürgen Quade,  
Hochschule  
Niederrhein

Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen

Der Taskzustand kann über die folgenden Befehle beeinflusst werden:

- `wait_event()`
- `wait_event_interruptible()`
- `wait_event_killable()`
- `wait_event_interruptible_timeout()`
- `wait_event_killable_timeout()`
- `wait_event_for_completion()`

Die Interruptible-Varianten sind durch Signals unterbrechbar.

# Taskzustand ändern (Fortsetzung)

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi

Jürgen Quade,  
Hochschule  
Niederrhein

Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen

- Killable-Varianten sind nur durch Signals unterbrechbar, die zum Ende des zugehörigen Jobs führen.
  - SIGKILL (9)
  - Alle Signals, die nicht explizit abgefangen werden.
- Weitere Funktionen um den Taskzustand zu beeinflussen:
  - `wake_up()`
  - `wake_up_interruptible()`

# driver\_read()

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi

Jürgen Quade,  
Hochschule  
Niederrhein

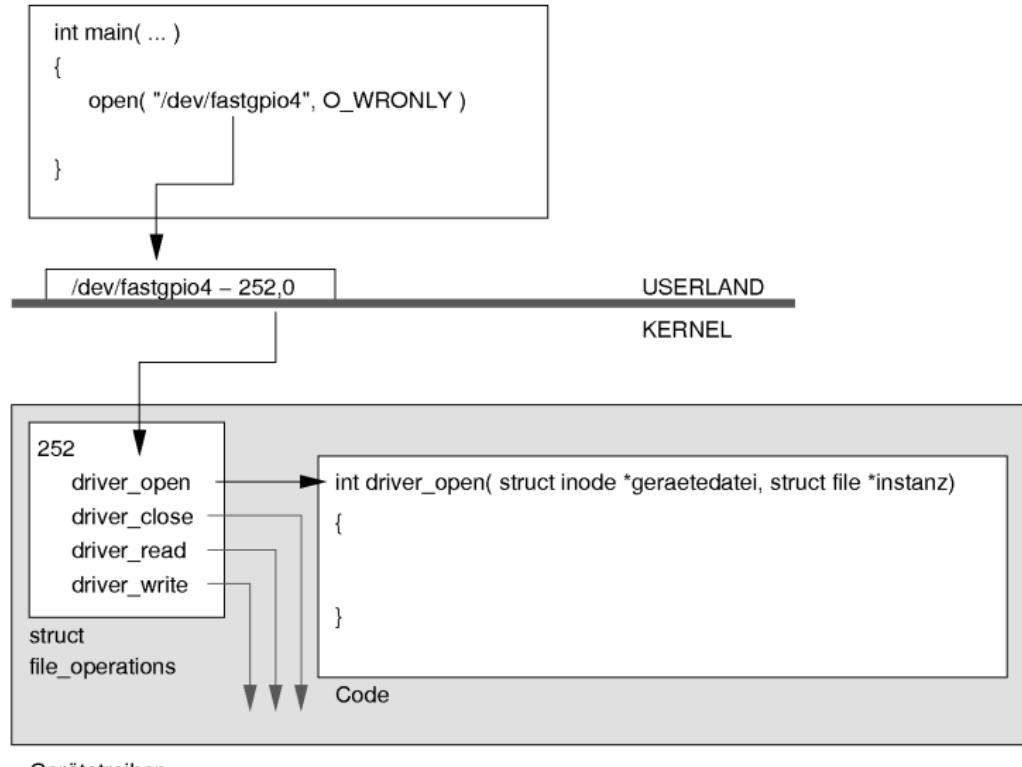
Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen



Gerätetreiber

# Funktion `wait_event_interruptible()`

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi

Jürgen Quade,  
Hochschule  
Niederrhein

Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen

## Aufgabe

- Rechenprozess in den Zustand *schlafen* (`TASK_INTERRUPTIBLE`) überführen:

## Vorgehen

- Objekt vom Typ `wait_queue_head_t` instanzieren.
- Auf einer `wait_queue` können beliebig viele Rechenprozesse schlafen.

## Codebeispiel

```
wait_event_interruptible( my_wait_queue, Schlafensbedingung )
```

# Hintergrund wait\_event\_interruptible()

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi

Jürgen Quade,  
Hochschule  
Niederrhein

Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen

- Funktion ist ein **Makro**.
  - Sie wird direkt mit der `wait_queue` parametert, nicht mit deren Adresse.
- Die Bedingung ist erforderlich, um eine Race-Condition zu verhindern: Das `wake_up()` (durch die ISR) kommt vor dem Schlafen-legen.

# Hintergrund wait\_event\_interruptible()

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi

Jürgen Quade,  
Hochschule  
Niederrhein

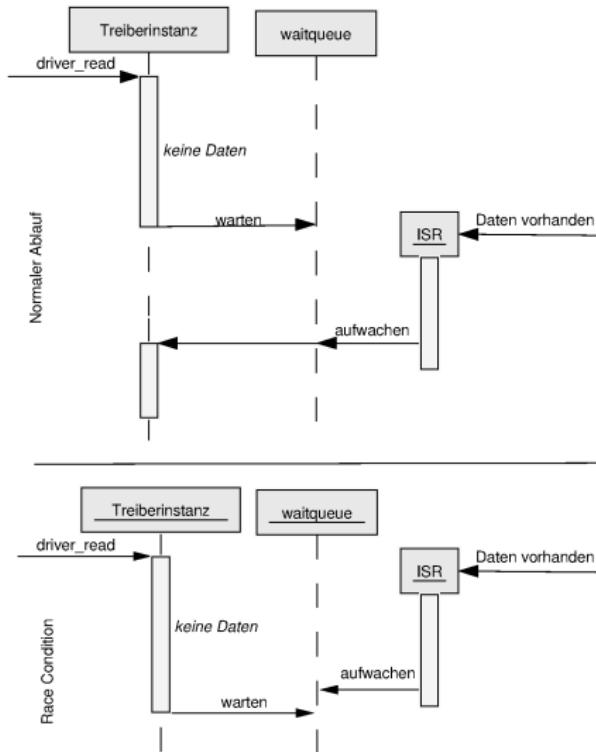
Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen



# Funktion wake\_up\_interruptible()

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi

Jürgen Quade,  
Hochschule  
Niederrhein

Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen

## Aufgabe

- Schlafende Jobs aufwecken

## Codebeispiel

```
wake_up_interruptible(&my_wait_queue);
```

# Codefragment Zugriffsmodus

```
static wait_queue_head_t wait_queue_for_read, wait_queue_for_write;
...
    init_waitqueue_head( &wait_queue_for_read );
    init_waitqueue_head( &wait_queue_for_write );
...
// keine Daten vorhanden
if( instanz->f_flags & O_BLOCKING ) { // blocking mode
    retval = wait_event_interruptible( wait_queue_for_read,
        data_available != 0 );
// nach dem Aufwecken arbeitet der Prozess hier weiter
...
int InterruptServiceRoutine( ... )
{
...
// jetzt sind Daten vorhanden
data_available = 1;
wake_up_interruptible( &wait_queue_for_read );
...
}
```

# Signals

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi

Jürgen Quade,  
Hochschule  
Niederrhein

Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen

- `wait_event()` schläft genau so lange, bis die Schlafensbedingung erfüllt ist.
- `wait_event_interruptible()` schläft so lange, bis entweder die Schlafensbedingung erfüllt ist oder aber ein Signal die Task aufweckt.
- Die Variante `wait_event_interruptible_timeout()` berücksichtigt zusätzlich ein Timeout (Zeitbedingung).
- Auftreten eines Signals während des Schlafens muss im Treiber berücksichtigt werden!

# Signals (Fortsetzung)

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi

Jürgen Quade,  
Hochschule  
Niederrhein

Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen

- Jedem Rechenprozess sind 64 Signals zugeordnet.
  - 32 Signals werden jedoch nur genutzt.
- Jedes Signal ist durch ein Bit repräsentiert.
- Die Zuordnung von Bit im Bitfeld ist in `<asm/signal.h>` definiert.

## Returnwert

- Der Returnwert der Funktion `wait_event_interruptible()` gibt an, ob das Schlafen aufgrund eines Signals beendet wurde:
  - `-ERESTARTSYS`

# Signals (Fortsetzung)

- In den Funktionen `driver_read()` und `driver_write()` wird das Schlafen durch ein Signal beendet:
  - Es werden typischerweise keine Daten in den User-Space kopiert.
  - Die Funktionen geben ihrerseits den Wert `-ERESTARTSYS` zurück.

## Code

```
...
/* keine Daten vorhanden */
    if( instanz->f_flags & O_BLOCKING ) { /* blocking mode */
        retval = wait_event_interruptible( wait_queue_for_read,
            data_available );
        if( retval == -ERESTARTSYS )
            return -ERESTARTSYS;
    }
/* Daten kopieren */
...
```

# Codefragment driver\_read()

```
#define READ_POSSIBLE (atomic_read(&bytes_available)!=0)

...
ssize_t driver_read( struct file *instance, char __user *buffer,
                     size_t max_bytes_to_read, loff_t *offset)
{
    size_t to_copy, not_copied;
    char kernelmem[128]; /* anpassen */
    ...
    if ( !READ_POSSIBLE && (instance->f_flags&O_NONBLOCK) )
        return -EAGAIN;
    if ( wait_event_interruptible( wq_read, READ_POSSIBLE ) )
        return -ERESTARTSYS;
    to_copy = min((size_t)atomic_read(&bytes_available),
                  max_bytes_to_read);
    not_copied = copy_to_user( buffer, kernelmem, to_copy );
    atomic_sub( to_copy-not_copied, &bytes_available );
    *offset = *offset + (to_copy-not_copied);
    return to_copy-not_copied;
}
```

# Codefragment driver\_write()

```
#define WRITE_POSSIBLE (atomic_read(&bytes_that_can_be_written) != 0)

...
ssize_t driver_write( struct file *instance, const char __user *buffer,
                      size_t max_bytes_to_write, loff_t *offset)
{
    size_t to_copy, not_copied;
    char kernelmem[128]; /* Groesse anpassen */

    if ( !WRITE_POSSIBLE && (instance->f_flags&O_NONBLOCK) )
        return -EAGAIN;
    if ( wait_event_interruptible(wq_write,WRITE_POSSIBLE) )
        return -ERESTARTSYS;

    to_copy = min((size_t) atomic_read(&bytes_that_can_be_written),
                  max_bytes_to_write);
    not_copied = copy_from_user( kernelmem, buffer, to_copy );
    write_data_to_hardware();
    atomic_sub( to_copy-not_copied, &bytes_that_can_be_written );
    *offset = *offset + (to_copy-not_copied);
    return to_copy-not_copied;
}
```

# Zusammenfassung

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi

Jürgen Quade,  
Hochschule  
Niederrhein

Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen

- Für die Realisierung eines Treibers wird benötigt:
  - `struct file_operations`
  - Methoden (`driver_open`, `driver_close`, `driver_read`, `driver_write`)
- Zugriffsmodi werden im Treiber realisiert.
- Dazu kann der Treiber die Applikation schlafen legen, falls Daten nicht vorhanden sind.

*Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi*

Jürgen Quade,  
Hochschule  
Niederrhein

Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen

## Section 2

### Hardware Anbindung

# Datentransfer User/Kernel/Hardware

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi

Jürgen Quade,  
Hochschule  
Niederrhein

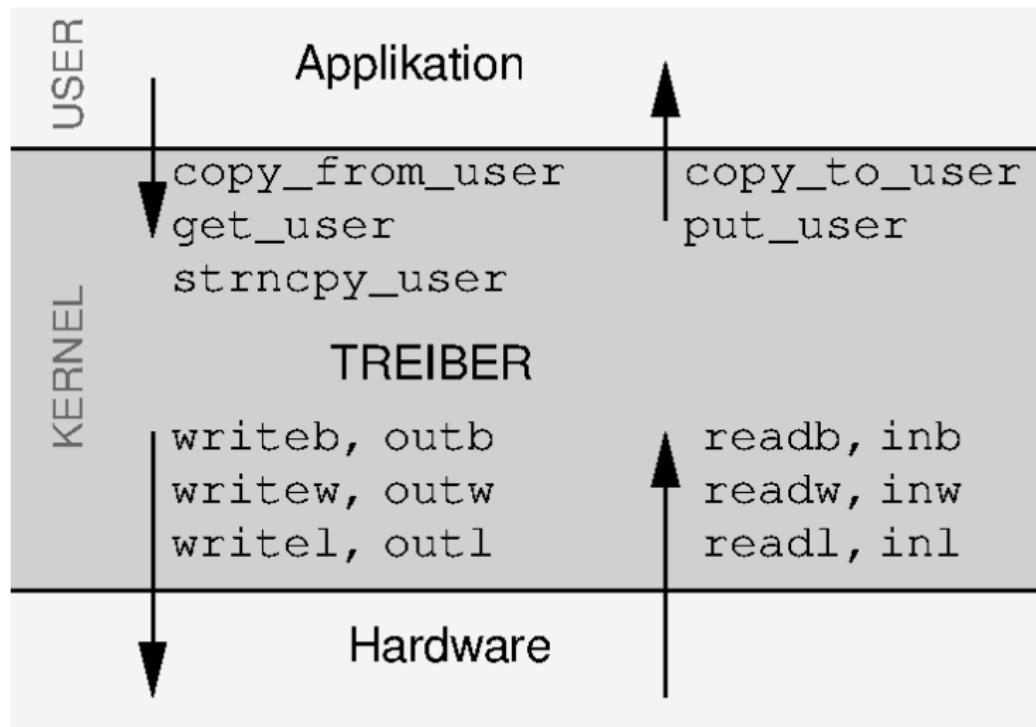
Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen



# Datentransfer Kernel-/User-Space

## Problematik

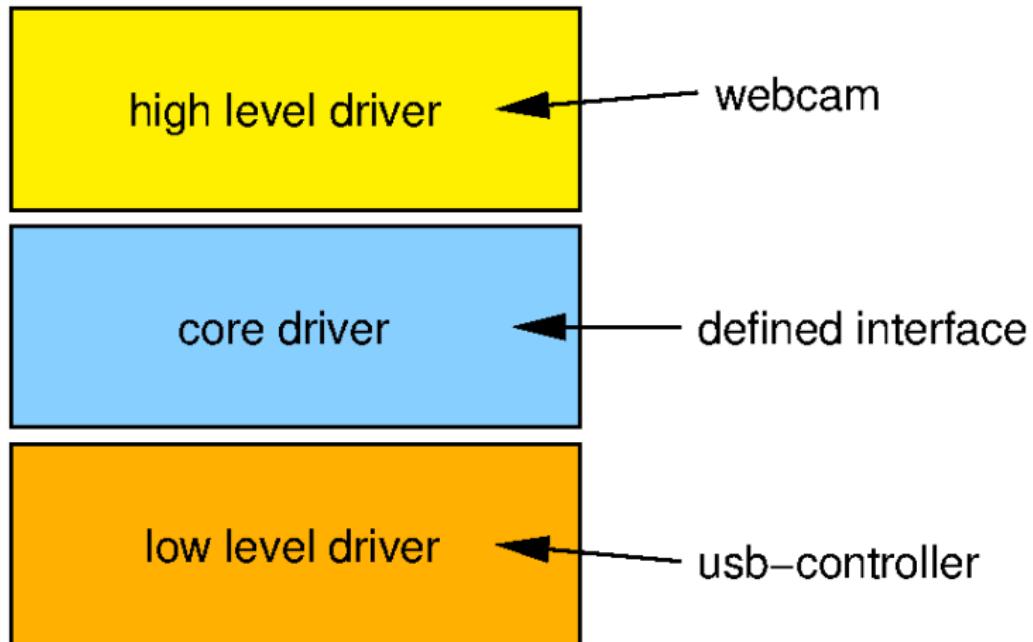
- Bedingt durch die Memory Management Unit (MMU) kann auch der Kernel nicht direkt auf den Speicher einer Applikation zugreifen.
- Der Kernel stellt Funktionen zur Verfügung, die den Datentransfer realisieren.

## Funktionen

- `copy_to_user`: Kopieren eines Speicherbereichs (wie `memcpy`)
- `copy_from_user`: Kopieren eines Speicherbereichs (wie `memcpy`)
- `put_user`: Kopieren eines *originären* Datentyps (`char`, `short`, `int`, `long`)
- `get_user`: Kopieren eines *originären* Datentyps (`char`, `short`, `int`, `long`)

# Abstraktionsebenen für Gerätetreiber

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi  
  
Jürgen Quade,  
Hochschule  
Niederrhein  
  
Zugriffsmodi  
  
Hardware  
Anbindung  
  
Ressourcen  
Verwaltung  
  
GPIOs  
  
Treiberinstanzen



# Abstraktionsebenen für Gerätetreiber

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi

Jürgen Quade,  
Hochschule  
Niederrhein

Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen

## Low Level Driver

- Implementieren die **direkten** Hardwarezugriffe (bei memory mapped IO direkte Speicherzugriffe)
- Beispiel: Ansteuerung eines USB-Controllers

## High Level Driver

- Nutzen Funktionen, die vom *core driver* zur Verfügung gestellt werden.
- Beispiel: Funktion zum Transfer eines Datenblocks zum USB-Gerät.

# Datentransfer Kernel/Hardware (low level)

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi

Jürgen Quade,  
Hochschule  
Niederrhein

Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen

Zugriff auf Hardware ist abhängig von der Ankopplungs-Art.

## Memory Mapped IO

- Abstraktion der (direkten) Speicherzugriffe über Funktionen beziehungsweise Makros
  - `write[bwl]`
  - `read[bwl]`

## IO-Ports (separater Adressbus für Peripheriezugriffe)

- `out[bwl], out[bwl]_p, # outs[bwl]`
- `in[bwl], in[bwl]_p, # in[bwl]`

# Datentypen

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi

Jürgen Quade,  
Hochschule  
Niederrhein

Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen

- Datentypen signalisieren in der Typbezeichnung die Bitbreite.
- Datentypen drücken das zugrundeliegende Datenablageformat aus.
- Das ist notwendig, wenn Systeme unterschiedlicher Bitbreite miteinander Datenaustauschen (z.B. ESP32 mit Linux 64 Bit).

| Datentyp               | Beschreibung                   |
|------------------------|--------------------------------|
| u8, u16, u32, u64      | vorzeichenlose Datentypen      |
| s8, s16, s32, s64      | vorzeichenbehaftete Datentypen |
| __le16, __le32, __le64 | little Endian                  |
| __be16, __be32, __be64 | little Endian                  |

# Konvertierungsfunktionen für Datenablageformate

| Funktionsname              | Bedeutung                   |
|----------------------------|-----------------------------|
| <code>cpu_to_le16()</code> | cpu 16 bit to little endian |
| <code>cpu_to_le32()</code> | cpu 32 bit to little endian |
| <code>cpu_to_le64()</code> | cpu 64 bit to little endian |
| <code>cpu_to_be16()</code> | cpu 16 bit to big endian    |
| <code>cpu_to_be32()</code> | cpu 32 bit to big endian    |
| <code>cpu_to_be64()</code> | cpu 64 bit to big endian    |

# Datenablage: aligned und packed

| Adresse     | Variable |   |
|-------------|----------|---|
| 0xbfffffa8c | u8       | a |
| 0xbfffffa8d | s8       | b |
| 0xbfffffa8e | frei     |   |
| 0xbfffffa8f |          |   |
| 0xbfffffa90 |          | c |
| 0xbfffffa91 |          |   |
| 0xbfffffa92 | s32      |   |
| 0xbfffffa93 |          | d |
| 0xbfffffa94 | s8       |   |
| 0xbfffffa95 | frei     |   |
| 0xbfffffa96 |          | e |
| 0xbfffffa97 | u16      |   |

Normale Datenablage  
aligned (ausgerichtet)

| Adresse     | Variable |     |
|-------------|----------|-----|
| 0xbfffffa8c | u8       | a   |
| 0xbfffffa8d | s8       | b   |
| 0xbfffffa8e |          |     |
| 0xbfffffa8f |          | s32 |
| 0xbfffffa90 |          |     |
| 0xbfffffa91 |          |     |
| 0xbfffffa92 | s8       |     |
| 0xbfffffa93 |          |     |
| 0xbfffffa94 |          | u16 |
| 0xbfffffa95 |          |     |
| 0xbfffffa96 |          |     |
| 0xbfffffa97 |          |     |

Gepackte Datenablage

```
struct _LetterVar {  
    __u8 a;  
    __s8 b;  
    __s32 c;  
    __s8 d;  
    __u16 e;  
} __attribute__ ((packed));
```

*Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi*

Jürgen Quade,  
Hochschule  
Niederrhein

Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen

## Section 3

### Ressourcen Verwaltung

# Reservierung

- Vor dem Zugriff müssen Hardware-Ressourcen beim Kernel reserviert werden.

## Ressourcen

| Ressource                 | Funktion                                                                 |
|---------------------------|--------------------------------------------------------------------------|
| IO-Ports                  | <code>request_region()</code> ,<br><code>release_region()</code>         |
| Memory (memory mapped io) | <code>request_mem_region()</code> ,<br><code>release_mem_region()</code> |
| Interrupts                | <code>request_irq()</code> , <code>free_irq()</code>                     |
| DMA-Kanäle                | <code>request_dma()</code> ,<br><code>release_dma()</code>               |

## Section 4

### GPIOs

# GPIO-Zugriff

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi

Jürgen Quade,  
Hochschule  
Niederrhein

Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen

- Viele Controller stellen *General Purpose Input Outputs* (GPIO) für die Ankopplung von Hardware zur Verfügung.
  - Raspberry Pi: 54 GPIOs

## Adressierungsmöglichkeiten

- Controller-Bezeichnung
- Pin-Nummer auf der Steckerleiste

Kernel verwendet die **Controller-Bezeichnung**

# GPIO-Zugriff

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi

Jürgen Quade,  
Hochschule  
Niederrhein

Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen

- GPIOs lassen sich zur Eingabe oder zur Ausgabe programmieren.

## Zugriffsfunktionen

- Gpio reservieren
- Gpio initialisieren
- Gpio verwenden
- Gpio freigeben

# Zugriff über Sys-Filsystem (deprecated)

## Unprofessioneller Zugriff

```
#!/bin/bash

trap "echo \"4\" >/sys/class/gpio/unexport" EXIT

echo "4" >/sys/class/gpio/export
echo "out" >/sys/class/gpio/gpio4/direction

while true
do
    echo "1" >/sys/class/gpio/gpio4/value
    sleep 1
    echo "0" >/sys/class/gpio/gpio4/value
    sleep 1
done
```

# GPIO reservieren

- GPIOs werden über eine „struct gpio\_desc“ repräsentiert

## Funktionsprototypen

```
struct gpio_desc *gpiod_get( struct device *,
                            const char * con_id, enum gpiod_flags flags )
struct gpio_desc *gpio_to_desc (unsigned gpio_number )
```

## Codebeispiel

```
static struct gpio_des *gd_echo;

gd_echo = gpio_to_desc( 17 );
if (IS_ERR(gd_echo)) {
    dev_err(gpiod_dev, "can't acquire GPIO\n");
    return -EIO;
}
```

# GPIO konfigurieren (direction)

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi

Jürgen Quade,  
Hochschule  
Niederrhein

Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen

## Funktionsprototypen

```
int gpiod_direction_output( struct gpio_desc * gpio,
                            int initval)
int gpiod_direction_input( struct gpio_desc * )
int gpiod_get_direction( const struct gpio_desc * )
int gpiod_to_irq( const struct gpio_desc * )
```

## Codebeispiel Output

```
err = gpiod_direction_output( my_gpio, 0 );
if (err) {
    dev_err(gpiod_dev, "gpio_dir_output\n", err);
    gpio_put( my_gpio );
    return -EIO;
}
```

# GPIO konfigurieren (direction)

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi

Jürgen Quade,  
Hochschule  
Niederrhein

Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen

## Prototyp

```
int gpiod_direction_input( struct gpio_desc * )
```

## Codebeispiel

```
err = gpiod_direction_input( gd_echo );
if (err) {
    printk("gpiod_direction_input failed\n");
    gpio_put( gd_echo );
    return -EIO;
}
```

## GPIO Input

## Prototypen

```
int gpiod_get_value( struct gpio_desc * )
int gpiod_cansleep( const struct gpio_desc * )
int gpiod_get_value_cansleep(const struct gpio_desc *)
```

# GPIO Output

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi

Jürgen Quade,  
Hochschule  
Niederrhein

Zugriffsmodi  
Hardware  
Anbindung  
Ressourcen  
Verwaltung  
GPIOs  
Treiberinstanzen

## Prototypen

```
int gpiod_set_value( struct gpio_desc *, int value )
int gpiod_set_value_can_sleep( struct gpio_desc *, int
                               value )
```

## Codebeispiel Rechteckfrequenz ausgeben

```
not_copied=copy_from_user(&value,user,to_copy);
if (value==1)
    gpiod_set_value( my_gpio, 1);
else
    gpiod_set_value( my_gpio, 0);
// Kurzversion:
// gpiod_set_value( my_gpio, value?0:1 );
```

# GPIO Freigabe

## Prototyp

```
int gpiod_put( struct gpio_desc * )
```

## Codebeispiel

```
...
if (my_gpio) {
    gpiod_put( my_gpio );
    my_gpio = NULL;
}
if (gd_echo) {
    gpiod_put( gd_echo );
    gd_echo = NULL;
}
...
```

## Section 5

### Treiberinstanzen

# Treiberinstanzen

- Mit jedem Aufruf von `open()` legt der Kernel eine Struktur vom Typ `struct file` an. Diese `struct file` repräsentiert eine *Treiberinstanz*.
- Treiber müssen sehr häufig **instanzenspezifische Daten** abspeichern.
- Deklarieren Sie für instanzenspezifische Parameter eine eigene Datenstruktur.
- Reservieren Sie beim `driver_open()` Speicher für ein solches Objekt.

# Treiberinstanzen (Fortsetzung)

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi

Jürgen Quade,  
Hochschule  
Niederrhein

Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen

- Speichern Sie die Adresse des Objektes in dem übergebenen Objekt vom Typ `struct file` ab.
- Bei jeder weiteren Treiberfunktion kann auf dieses Objekt zugegriffen werden.

## Codebeispiel

```
...
int driver_open( struct inode *devfile, struct file *instance )
{
    instance->private_data = kmalloc( sizeof(struct inst_data) );
    ...
}
```

# Treiberinstanzen (Fortsetzung)

Beim `driver_close()` wird der per `kmalloc()` reservierte Speicher wieder freigegeben.

## Codebeispiel

```
...
int driver_close( struct inode *devfile, struct file *instance )
{
    if (instance->private_data) {
        kfree( instance->private_data );
    }
    ...
}
```

# Quintessenz

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Zugriffsmodi

Jürgen Quade,  
Hochschule  
Niederrhein

Zugriffsmodi

Hardware  
Anbindung

Ressourcen  
Verwaltung

GPIOs

Treiberinstanzen

- Viele Datenstrukturen des Kernels bieten Hooks an, um dort eigene (modul- oder treiberspezifische) Datenstrukturen anzukoppeln.
- Eine Treiberinstanz ist im Kernel durch die Struktur `struct file` repräsentiert.
- Diese Struktur hält mit dem Element `private_data` ein Element bereit, um dort instanzenspezifische Parameter anzukoppeln.
- In der Applikation ist eine Treiberinstanz durch den Filedeskriptor repräsentiert.

# *Embedded Systems - Devicetree*

Jürgen Quade, Hochschule Niederrhein

30.05.2022

## 1 Problemstellung

## 2 Realisierung

## 3 Syntax

## 4 Treiberzugriff

**Prof. Dr.-Ing. Jürgen Quade**  
**Hochschule Niederrhein**

## Section 1

### Problemstellung

# Hardware-Vielfalt

Embedded  
Systems -  
Devicetree

Jürgen Quade,  
Hochschule  
Niederrhein

Problemstellung

Realisierung

Syntax

Treiberzugriff

- ARM definiert einen Befehlssatz, aber nicht die Implementierung und die Ankopplung an Peripherie.
- Für den Zugriff auf Hardware müssen die Adressen der Hardwareregister oder die verdrahteten Interruptsignale bekannt sein.
- Adressen unterscheiden sich von Plattform zu Plattform

## Konsequenz

- Für jede Plattform muss ein Kernel mit angepassten Adressen generiert werden
- Gerätetreiber müssen auf jede Plattform angepasst werden
- Die Variantenvielfalt wächst ständig.

# Lösung

## Trennung von Code und Daten

- Adressen sind Daten
- Adressen - oder verallgemeinert Ressourcen – werden unabhängig vom Code in einer textuellen Konfigurationsdatei beschrieben
- Der Code *erfragt* seine Ressourcen über ein standardisiertes Interface.
- Für die Zuordnung zwischen Treiber (Code) und den passenden Ressourcen werden Strings (*compatible*) verwendet.

# Historie

Embedded  
Systems -  
Devicetree

Jürgen Quade,  
Hochschule  
Niederrhein

Problemstellung

Realisierung

Syntax

Treiberzugriff

- Ursprung der Device-Tree's: **Open Firmware**
  - *Erfinder:* Sun Microsystems für SPARC und PowerPC
  - Daher auch der Prefix **of\_**
- BIOS-Konkurrent
- Verbreitung der Hardwarebeschreibung zunächst auf PowerPC, danach ARM

# Begriffe

Embedded  
Systems -  
Devicetree

Jürgen Quade,  
Hochschule  
Niederrhein

Problemstellung

Realisierung

Syntax

Treiberzugriff

**Device Tree** Beschreibung der Ressourcen, insbesondere  
HW-Adressen, Interrupts, DMA

**Device Tree Compiler** Kompiler (Programmname dtc), der eine  
textuelle Beschreibung (.dts) in ein Binärformat  
(Device Tree Blob, .dtb) überführt

**Device Tree Blob** Binäre Darstellung des Device Tree

**Device Tree Overlay** Subtree, der nachträglich geladen einen  
vorhandenen Device-Tree modifiziert oder ergänzt  
(.dtbo)

## Embedded Systems - Devicetree

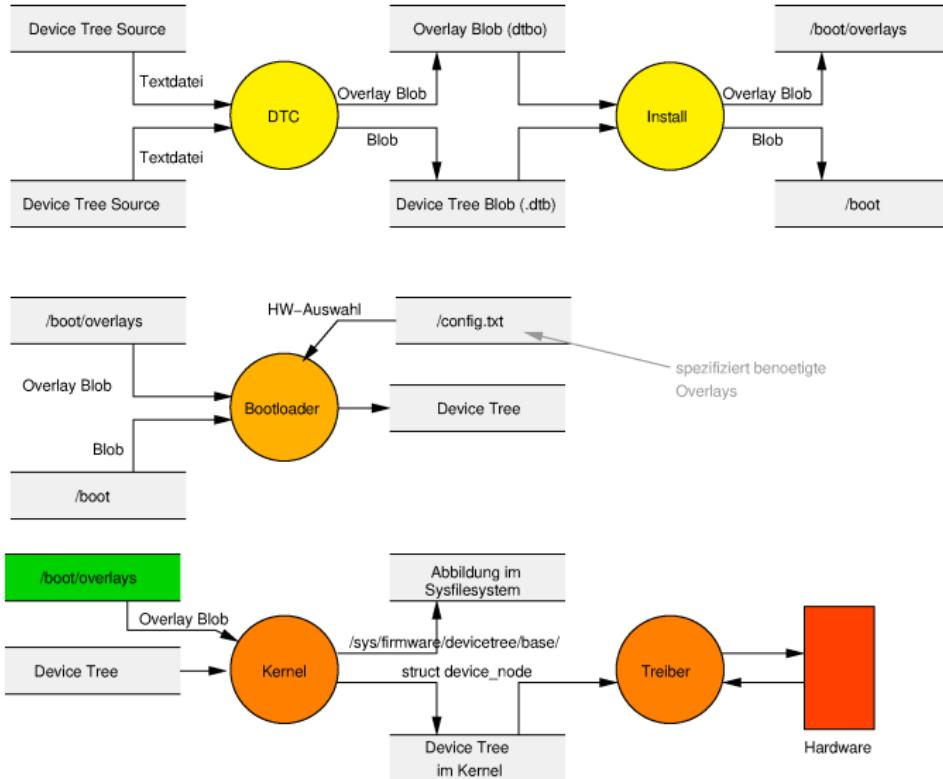
Jürgen Quade,  
Hochschule  
Niederrhein

## Problemstellung

### Realisierung

### Syntax

### Treiberzugriff



## Section 2

### Realisierung

# Technische Realisierung

Embedded  
Systems -  
Devicetree

Jürgen Quade,  
Hochschule  
Niederrhein

Problemstellung

Realisierung

Syntax

Treiberzugriff

- Hardware wird anhand eines Baumes beschrieben.
- Unterschiedliche Subsysteme (I2C, SPI, USB, CPU) bilden die Gabelungen.
- Darstellungsformat JSON

## Device Tree Bindings

- Ein **Device Tree Binding** spezifiziert, wie eine Hardware beschrieben wird
  - welche Attribute existieren
  - welche Einheiten/Formate Attribute aufweisen
  - welche Attribute verpflichtend, welche optional sind
  - welche Defaultwerte verwendet werden
- Ablage im Kernel unter Documentation/devicetree/bindings/
- Werden in YAML (vereinfachte Auszeichnungssprache) geschrieben

# Beispiel *Devicetree*

Embedded  
Systems -  
Devicetree

Jürgen Quade,  
Hochschule  
Niederrhein

Problemstellung

Realisierung

Syntax

Treiberzugriff

```
1 /dts-v1/;  
2 /plugin/;  
3  
4 / {  
5     compatible = "brcm,bcm2708", "brcm,bcm2709";  
6  
7     fragment@0 {  
8         target = <&gpio>;  
9         __overlay__ {  
10            pinctrl-names = "default";  
11            pinctrl-0 = <&my_pins>;  
12  
13            my_pins: my_pins {  
14                brcm,pins = <7 8 9>; /* gpio no. */  
15                brcm,function = <0 0 0>; /* 0:in, 1:out */  
16                brcm,pull = <1 1 2>; /* 2:up 1:down 0:none */  
17            };  
18        };  
19    };
```

# Einsatz

Embedded  
Systems -  
Devicetree

Jürgen Quade,  
Hochschule  
Niederrhein

Problemstellung

Realisierung

Syntax

Treiberzugriff

- Device Tree kann direkt zum Kernel dazugebunden werden
- Device Tree wird beim Booten im Speicher abgelegt
- Speicheradresse wird dem Kernel als Parameter übergeben
- Device Tree Beschreibungen finden sich im Kernel unter arch/<ARCH>/boot/dts/
- Device Tree wird *visualisiert* im Sys-Filesystem über /sys/firmware/devicetree/base/

## Section 3

### Syntax

# Überblick

```
1 / {  
2     #address-cells = <1>;  
3     #size-cells = <1>;  
4     model = "STMicroelectronics STM32MP157C-DK2 Discovery Board";  
5     compatible = "st,stm32mp157c-dk2", "st,stm32mp157";  
6     cpus { ... };  
7     memory@0 { ... };  
8     chosen { ... };  
9     intc: interrupt-controller@a0021000 { ... };  
10    soc {  
11        i2c1: i2c@40012000 { ... };  
12        ethernet0: ethernet@5800a000 { ... };  
13    };  
14};
```

# Knoten

Embedded  
Systems -  
Devicetree

Jürgen Quade,  
Hochschule  
Niederrhein

Problemstellung

Realisierung

Syntax

Treiberzugriff

Device Tree besteht aus Knoten, die einen Namen haben

## Beispiel

```
node@0 {  
    ...  
};
```

# Attribute

Embedded  
Systems -  
Devicetree

Jürgen Quade,  
Hochschule  
Niederrhein

Problemstellung

Realisierung

Syntax

Treiberzugriff

- Attribute eines Knoten (Node) werden innerhalb von geschweiften Klammern spezifiziert
  - Key-Value Paar
  - Attribute plus Werte
- Attribute sind grundsätzlich im *Big Endian* Format angegeben

## Beispiel

```
node@0 {  
    first-child-property; // bool value  
    second-child-property = <1>;  
    a-reference-to-something = <&node1>;  
};
```

# Kindknoten

- Kindknoten werden innerhalb der Elternknoten definiert (Hierarchie)

```
node@0 {  
    ...  
    child-node@0 {  
        ...  
    };  
};
```

# Referenzen

- Referenzen erfolgen über so genannte **phandle**
- Vor Knotennamen lassen sich Sprungziele (Label) stellen

## Beispiel

```
node@0 {  
    ...  
    a-reference-to-someging = <&node1>;  
};  
  
node1: node@1 {  
    ...  
};
```

# Beschreibungselemente

*Embedded  
Systems -  
Devicetree*

Jürgen Quade,  
Hochschule  
Niederrhein

Problemstellung

Realisierung

Syntax

Treiberzugriff

Device Tree Beschreibung!

(Thomas Petazzoni: Device Tree 101.

<https://bootlin.com/pub/conferences/2021/webinar/petazzoni-device-tree-101/petazzoni-device-tree-101.pdf>)

## Embedded Systems - Devicetree

Jürgen Quade,  
Hochschule  
Niederrhein

Problemstellung

Realisierung

Syntax

Treiberzugriff

```
root@raspberrypi:/home/quade/dt# dtc -@ -I dts -O dtb -o linmag.dtbo linmag.dts
Warning (unit_address_vs_reg): Node /fragment@0 has a unit name, but no reg property
root@raspberrypi:/home/quade/dt# dtoverlay linmag.dtbo
root@raspberrypi:/home/quade/dt# cd /sys/firmware/devicetree/base/soc/gpio@7e200000/
root@raspberrypi:/sys/firmware/devicetree/base/soc/gpio@7e200000# ls
#address-cells    i2c0          reg          string-property
audio_pins        i2c1          sdhost_pins   u16
bt_pins           i2s           sdio_pins     u32
bytes             #interrupt-cells #size-cells  uart0_pins
compatible        interrupt-controller spi0_cs_pins  uart1_pins
foo               interrupts      spi0_pins
#gpio-cells       name          status
gpio-controller   phandle       string-list-property
root@raspberrypi:/sys/firmware/devicetree/base/soc/gpio@7e200000# cat string-property
Linux Magazin
root@raspberrypi:/sys/firmware/devicetree/base/soc/gpio@7e200000# hd u16
00000000 73 77
|sw|
00000002
root@raspberrypi:/sys/firmware/devicetree/base/soc/gpio@7e200000# hd bytes
00000000 45 76 61
|Eva|
00000003
root@raspberrypi:/sys/firmware/devicetree/base/soc/gpio@7e200000# cd foo
root@raspberrypi:/sys/firmware/devicetree/base/soc/gpio@7e200000/foo# ls
compatible  my_string  name  status
root@raspberrypi:/sys/firmware/devicetree/base/soc/gpio@7e200000/foo# █
```

## Section 4

### Treiberzugriff

# Allgemeines

Embedded  
Systems -  
Devicetree

Jürgen Quade,  
Hochschule  
Niederrhein

Problemstellung

Realisierung

Syntax

Treiberzugriff

- Kernel versucht Device Tree Beschreibungen direkt (über den *compatible*-String) den Geräten zuzuordnen.
- Device Tree Parameter (Attribute) sind daher in der `struct device` mit abgelegt.

# Allgemeine Schnittstelle

Embedded  
Systems -  
Devicetree

Jürgen Quade,  
Hochschule  
Niederrhein

Problemstellung

Realisierung

Syntax

Treiberzugriff

- `struct device_node *of_find_node_by_path( char *nodename );`
- `of_get_property()`

## Funktionen zur Konvertierung von Byteformaten

### Konvertierungsfunktion

---

`be16_to_cpup()`  
`be32_to_cpup()`  
`be64_to_cpup()`

---

# Match-Tabelle im Treiber

```
1 static struct of_device_id
2     linmag_match[] = {
3         { .compatible = "linmag" },
4         {}
5     };
6
7 MODULE_DEVICE_TABLE( of, linmag_match );
```

- Zur automatisierten Zuordnung von Device Tree Einträgen zum Treiber spezifiziert der Treiber eine Match-Tabelle.

# Auslesen im Treiber

```
static int linuxmag_probe_device( struct platform_device *pdev )
{
    struct device *dev = &pdev->dev;
    struct device_node *nodeptr = dev->of_node;
    const void *prop_value;
    int size, i;
    u16 value;
    const __be16 *beptr;

    nodeptr = of_find_node_by_path("gpio");

    prop_value = of_get_property( nodeptr, "string-property", &size );
    if (prop_value)
        printk("prop_value: %s\n", (char *)prop_value);
    ...
}
```

# Auslesen im Treiber (Fortsetzung)

```
...
prop_value = of_get_property( nodeptr, "bytes", &size );
printf("byte array: [ ");
for ( i=0; i<size; i++ ) {
    printf("%x ", *(char *)(prop_value+i));
}
printf(" ]\n");

beptr = of_get_property( nodeptr, "u16", &size );
if (prop_value) {
    value = be16_to_cpup( beptr );
    printf("value: %x\n", value);
}
return 0;
}
```

```
root@raspberrypi:/home/quade/treiber# apt-get install raspberrypi-kernel-headers
Reading package lists... Done
Building dependency tree
Reading state information... Done
raspberrypi-kernel-headers is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
root@raspberrypi:/home/quade/treiber# make
make -C /lib/modules/4.9.24-v7+/build M=/home/quade/treiber modules
make[1]: Entering directory '/usr/src/linux-headers-4.9.24-v7+'
CC [M] /home/quade/treiber/string.o
Building modules, stage 2.
MODPOST 1 modules
CC      /home/quade/treiber/string.mod.o
LD [M]  /home/quade/treiber/string.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.9.24-v7+'
root@raspberrypi:/home/quade/treiber# insmod string.ko
root@raspberrypi:/home/quade/treiber# head -n 2 /dev/linuxmag
Linux Magazin
Linux Magazin
root@raspberrypi:/home/quade/treiber# tail -n 10 /var/log/messages
May 29 09:48:58 raspberrypi kernel: [ 3807.871305] prop_value: Linux Magazin
May 29 09:48:58 raspberrypi kernel: [ 3807.871305]
May 29 09:48:58 raspberrypi kernel: [ 3807.871315] driver_open(): prop_value=b9b94f00, size: 3
May 29 09:48:58 raspberrypi kernel: [ 3807.871319] byte array: [
May 29 09:48:58 raspberrypi kernel: [ 3807.871324] 45
May 29 09:48:58 raspberrypi kernel: [ 3807.871327] 76
May 29 09:48:58 raspberrypi kernel: [ 3807.871331] 61
May 29 09:48:58 raspberrypi kernel: [ 3807.871334] ]
May 29 09:48:58 raspberrypi kernel: [ 3807.871340] driver_open(): prop_value=b9b94f00, size: 2
May 29 09:48:58 raspberrypi kernel: [ 3807.871344] value: 7377
root@raspberrypi:/home/quade/treiber#
```

# *Embedded Systems - Kernelprogrammierung - Softirqs und Kernel-Threads*

Jürgen Quade, Hochschule Niederrhein

11.05.2022

## 1 Allgemeines

## 2 ISR

## 3 Soft-IRQs

## 4 Kernel-Threads

Prof. Dr.-Ing. Jürgen Quade

Hochschule Niederrhein

## Section 1

### Allgemeines

# Übersicht

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Softirqs und  
Kernel-  
Threads

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines

ISR

Soft-IRQs

Kernel-  
Threads

## Treiberprogrammierung

Funktionen des Gerätetreibers wie `driver_open()` oder `driver_read()` werden durch die Applikation *getriggert*.

## Kernelprogrammierung

Funktion, die unabhängig von Applikationen sind (also asynchron zu Applikationen arbeiten). Die Aktivierung erfolgt über

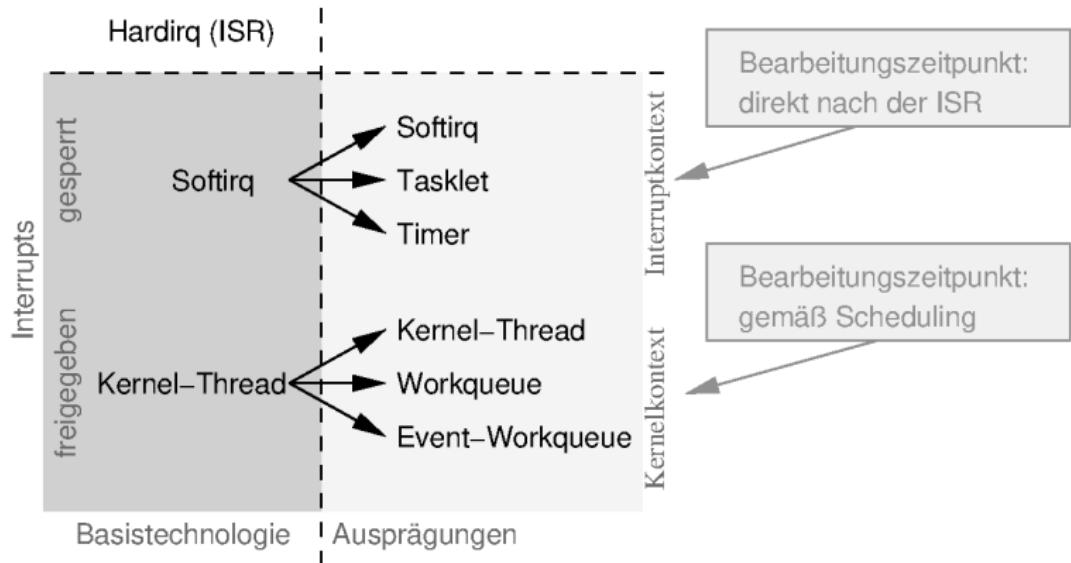
- Interrupts beziehungsweise Ereignisse
- zeitgesteuert

# Asynchrone Treiberfunktionen

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Softirqs und  
Kernel-  
Threads

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
ISR  
Soft-IRQs  
Kernel-  
Threads



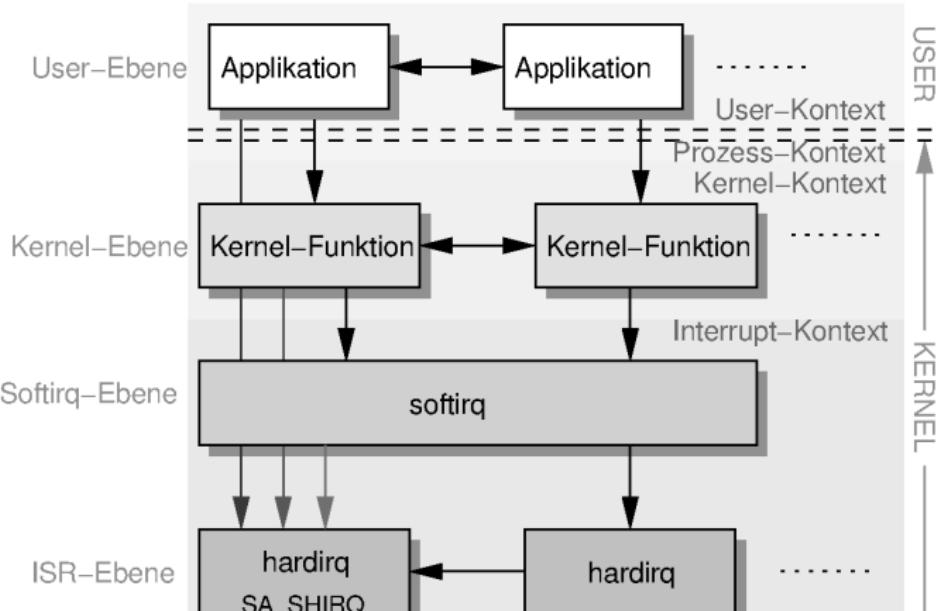
# Ebenen-, Kontext- und Unterbrechungsmodell

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Softirqs und  
Kernel-  
Threads

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
ISR  
Soft-IRQs

Kernel-  
Threads



a → b    a kann durch b  
unterbrochen werden

# Definitionen

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Softirqs und  
Kernel-  
Threads

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
ISR  
Soft-IRQs  
Kernel-  
Threads

- Kontext: Die *Umgebung* gibt an, auf welche Dienste und Ressourcen ein Codefragment Zugriff hat.
- Die Programmiererin/der Programmierer muss daher den Kontext kennen, um nicht unerlaubte Funktionen einzusetzen.
- Manche Kernelfunktionen (z.B. `kmalloc()`) verhalten sich abhängig vom Kontext unterschiedlich, daher gibt man ihnen beim Aufruf den Kontext als Parameter mit.

# Interrupt-Kontext

## Eigenschaft

Limitierte Funktionsauswahl, insbesondere kein *Schlafen legen* und kein *Zugriff auf Applikations-Speicherbereiche*.

## Beispiele

- ISRs, Soft-IRQs, Tasklets und Timer.
- Define im Kernel: GFP\_ATOMIC

# Kernel-Kontext

## Eigenschaft

Zugriff auf sämtliche Funktionen des Kernels, inklusive *Schlafen legen.*

## Beispiele

- Kernel-Threads, Workqueues, Event-Workqueue
- Define im Kernel: `GFP_KERNEL`

# Prozess-/User-Kontext

## Eigenschaft

Wie Kernel-Kontext, zusätzlich noch Zugriff auf die Datenbereiche des gerade aktiven Rechenprozesses.

## Beispiele

- Applikationsgetriggerte Treiberfunktionen, Systemcalls GFP\_USER
- Define im Kernel: GFP\_USER

*Embedded  
Systems -  
Kernelpro-  
grammierung -  
Softirqs und  
Kernel-  
Threads*

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
ISR

Soft-IRQs

Kernel-  
Threads

## Section 2

### ISR

# Hardware ISR

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Softirqs und  
Kernel-  
Threads

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
ISR  
Soft-IRQs  
Kernel-  
Threads

- (Typischerweise) nicht unterbrechbar.

## Codebeispiel ISR

```
static irqreturn_t driver_isr( int irq, void *dev_id )
{
    return IRQ_HANDLED;
}
```

# Hardware ISR (Fortsetzung)

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Softirqs und  
Kernel-  
Threads

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
ISR  
Soft-IRQs  
Kernel-  
Threads

## Codebeispiel Anmelden

```
static int device_init(struct pci_dev *dev,
                      const struct pci_device_id *id)
{
    ...
    if(request_irq(dev->irq,driver_isr,
                   IRQF_DISABLED|IRQF_SHARED, "pci_drv",dev)) {
        goto cleanup_mem;
    }
    ...
}
```

## Section 3

### Soft-IRQs

# Soft-IRQs

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Softirqs und  
Kernel-  
Threads

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
ISR  
Soft-IRQs  
Kernel-  
Threads

- Werden im Interrupt-Kontext abgearbeitet:
  - Kein Schlafen-Legen: `wait_event()`...
  - Kein Zugriff auf den User-Space: `copy_to_user()`...
- Abarbeitungszeitpunkt: direkt nach einer ISR.
- Werden durch Interrupts unterbrochen.
- Insgesamt stehen 32 Soft-IRQs zur Verfügung, die bei spezifischen Interrupts vom Kernel (nacheinander) bearbeitet werden. (Ein Soft-IRQ wird von der ISR aktiviert.)

# Soft-IRQs (Fortsetzung)

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Softirqs und  
Kernel-  
Threads

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines

ISR

Soft-IRQs

Kernel-  
Threads

- 10 Soft-IRQs sind vordefiniert. Dort lassen sich Funktionen einhängen (schedulen), die bei Aufruf des Soft-IRQs abgearbeitet werden.
- Die übrigen Soft-IRQs sollen nicht verwendet werden.

## Wesentliche Ausprägungen

- Tasklet: Wird mit dem nächsten Interrupt abgearbeitet.
- Timer: Wird zum angegebenen Zeitpunkt abgearbeitet.

# Priorisierung

|                 |                         |
|-----------------|-------------------------|
| HI_SOFTIRQ      | Hochpriores Tasklet     |
| TIMER_SOFTIRQ   | Zeitgesteuerte Aufgaben |
| NET_TX_SOFTIRQ  | Netzwerk-Senden         |
| NET_RX_SOFTIRQ  | Netzwerk-Empfangen      |
| BLOCK_SOFTIRQ   | Blockgeräte-Subsystem   |
| BLOCK_IOPOLL    | Blockgeräte-Subsystem   |
| TASKLET_SOFTIRQ | Normales Tasklet        |
| SCHED_SOFTIRQ   | Scheduler               |
| HRTIMER_SOFTIRQ | Hochauflösende Timer    |
| RCU_SOFTIRQ     | RCU                     |

# Tasklets

- Funktion, die im Kernel *so bald als möglich* ausgeführt wird.
- Ein Tasklet wird maximal einmal aufgerufen.

## Einsatz

- Hochpriore Funktion
- Interrupts sind freigegeben

## ACHTUNG: Race-Condition

Falls Tasklet-Code vor dem Aufruf entladen wird

# Tasklets

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Softirqs und  
Kernel-  
Threads

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
ISR  
Soft-IRQs

Kernel-  
Threads

```
#include <linux/module.h>
#include <linux/init.h> Tasklet-Funktion

// Funktion, die aktiviert werden soll
static void tasklet_function( unsigned long data )
{
    printk("Tasklet called...\n");
    return;
}

// Tasklet anlegen
DECLARE_TASKLET( tldescr, tasklet_function, 0L );
...
```

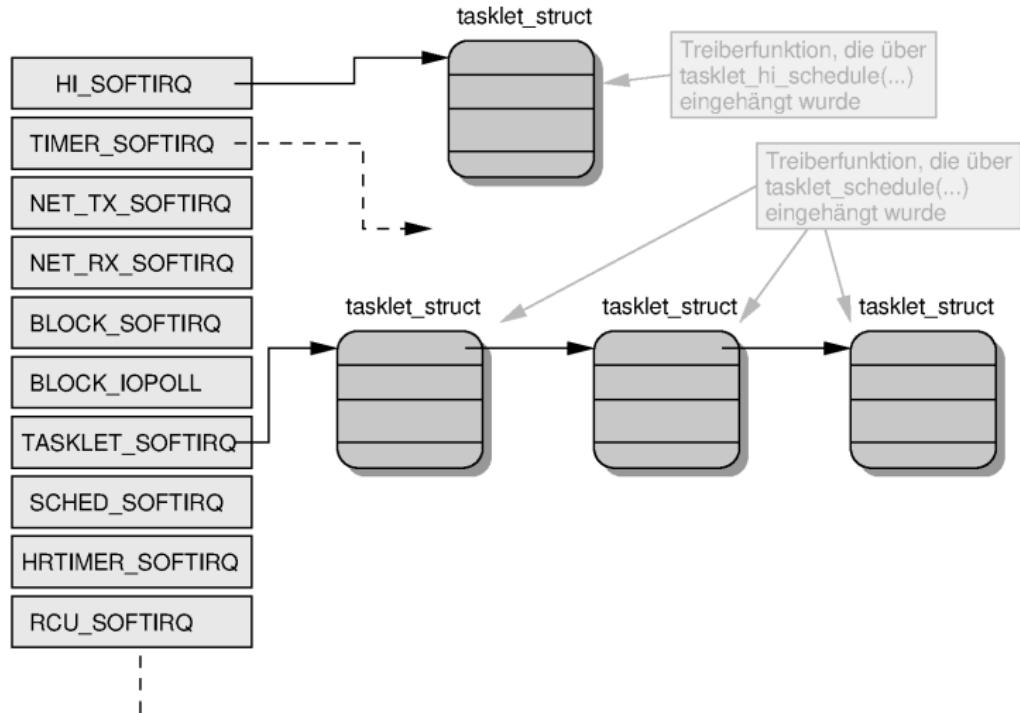
# Tasklets (Fortsetzung)

```
static int __init mod_init(void)
{
    printk("mod_init called\n");
    tasklet_schedule( &tldescr );
    return 0;
}

static void __exit mod_exit(void)
{
    printk("mod_exit called\n");
    tasklet_kill( &tldescr ); // tasklet entfernen
}

module_init( mod_init );
module_exit( mod_exit );
```

# Tasklets (Fortsetzung)



# Tasklet-Management

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Softirqs und  
Kernel-  
Threads

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
ISR  
Soft-IRQs  
Kernel-  
Threads

## Statische Defintion

```
DECLARE_TASKLET( struct tasklet_struct name, void  
    (*func)(unsigned long), unsigned long data );
```

## Initialisierung zur Laufzeit

```
void tasklet_init( struct tasklet_struct t, void  
    (*func)(unsigned long), unsigned long data );
```

# Tasklet-Laufzeitfunktionen

Normales Tasklet zur Abarbeitung freigeben.

```
void tasklet_schedule( struct tasklet_struct *t );
```

Hochpriores Tasklet zur Abarbeitung freigeben.

```
void tasklet_hi_schedule( struct tasklet_struct *t );
```

Freigegebenes Tasklet beenden.

```
void tasklet_kill( struct tasklet_struct *t );
```

# Timer

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Softirqs und  
Kernel-  
Threads

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines

ISR

Soft-IRQs

Kernel-  
Threads

- Aufruf einer Funktion zu einem späteren, definierten Zeitpunkt.
- Zeiten werden absolut auf Basis von Jiffies angegeben.
- Timer dürfen zu einem Zeitpunkt maximal einmal eingehängt werden.
- Die Timer-Funktion wird im Interruptkontext abgearbeitet (Interrupts sind freigegeben).

## ACHTUNG: Race-Condition

Falls der Timer-Code vor dem Aufruf entladen wird...

# Timer - Codebeispiel

```
#include <linux/module.h>
static struct timer_list mytimer; // Timer Objekt

static void inc_count(unsigned long arg)
{
    printk("inc_count called (%ld)...\\n", mytimer.expires );
}

static int __init ktimer_init(void)
{
    init_timer( &mytimer ); // Initialisierung
    mytimer.function = inc_count;
    mytimer.data = 0;
    mytimer.expires = jiffies + (2*HZ); // 2 seconds
    add_timer( &mytimer ); // Instanzierung
    return 0;
}
static void __exit ktimer_exit(void)
{
    if( del_timer_sync( &mytimer ) )
        printk("Aktiver Timer deaktiviert\\n");
}
```

# Timer Funktionen

*Embedded  
Systems -  
Kernelpro-  
grammierung -  
Softirqs und  
Kernel-  
Threads*

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines

ISR

Soft-IRQs

Kernel-  
Threads

- `init_timer()`
- `add_timer()`
- `mod_timer()`
- `del_timer_sync()`
- `timer_pending()`

# Zwischenstand

Embedded  
Systems -  
Kernelpro-  
grammierung -  
SoftIRQs und  
Kernel-  
Threads

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
ISR  
Soft-IRQs  
Kernel-  
Threads

- Über Soft-IRQs lassen sich hochprior Funktionen abarbeiten.
- Der Soft-IRQ *Tasklet* wird nach dem nächsten Interrupt abgearbeitet. Hochpriore Tasklets werden direkt nach einem Interrupt abgearbeitet.
- Der Soft-IRQ *Timer* ermöglicht die zeitgesteuerte Abarbeitung einer hochprioren Funktion.
- Andere Soft-IRQs werden beispielsweise vom Netzwerk-Subsystem oder vom SCSI-Subsystem eingesetzt.

## Section 4

### Kernel-Threads

# Definition

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Softirqs und  
Kernel-  
Threads

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
ISR  
Soft-IRQs  
Kernel-  
Threads

- Kernel-Thread = Thread, der im Kernel abgearbeitet wird und typischerweise keine Ressourcen im Userland besitzt.
  - In der Taskliste sichtbar.
- Ist durch einen eigenen PCB definiert.
- *Steuerung* des Threads über Signals.
- Kernel-Threads können schlafen.
- Kernel-Threads laufen bis
  - sie sich beenden
  - sie sich schlafen legen

## ACHTUNG: Race Condition

Thread muss vor dem Entfernen des zugehörigen Codes beendet sein.

# Beispiele

In der Taskliste sind Kernel-Threads durch die Klammerung  
identifizierbar.

```
quade@ezs-cock-a:~$ ps ax | head
  PID TTY      STAT   TIME COMMAND
      1 ?        Ss      0:18 /sbin/init splash
      2 ?        S       0:00 [kthreadd]
      3 ?        I<     0:00 [rcu_gp]
      4 ?        I<     0:00 [rcu_par_gp]
      6 ?        I<     0:00 [kworker/0:0H-events_highpri]
      9 ?        I<     0:00 [mm_percpu_wq]
     10 ?       S       0:00 [rcu_tasks_rude_]
     11 ?       S       0:00 [rcu_tasks_trace]
     12 ?       S      0:08 [kssoftirqd/0]
quade@ezs-cock-a:~$
```

# Struktur

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Softirqs und  
Kernel-  
Threads

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
ISR  
Soft-IRQs  
Kernel-  
Threads

`allow_signal(...)` – Signale freigeben

`!kthread_should_stop()` – Solange der Thread nicht stoppen soll

DER EIGENTLICHE THREAD-CODE STEHT HIER.

`wait_event_interruptible(...)` – Anderen Rechenprozessen Rechenzeit geben

`complete_and_exit(...)` – Signalisiere dem Modul das Thread-Ende

# Instanzierung

- Zum Erzeugen: von Kernel-Threads stellt der Kernel die Funktion `kthread_create()` und `kthread_run()` zur Verfügung
  - `kthread_create()`: Thread wird erstellt, aber nicht gestartet; muss per `wake_up_process()` gestartet werden
  - `kthread_run()`: Thread wird erzeugt und gestartet
- Der neue Thread wird dann vom *kthread-Daemon* erzeugt
- Der *Kthread-Daemon* ist selbst ein Kernel-Thread, in dessen Kontext der Kind-Thread erzeugt wird.
  - Eindeutiger Kontext.

# Beispielcode *Management*

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Softirqs und  
Kernel-  
Threads

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines

ISR

Soft-IRQs

Kernel-  
Threads

```
#include <linux/kthread.h>
#include <net/sock.h>
...
static int __init kthread_init(void)
{
    init_waitqueue_head(&twq);
    thread_id=kthread_create(thread_code,NULL,"mykthread");
    if( thread_id==0 )
        return -EIO;
    wake_up_process( thread_id );
    return 0;
}

static void __exit kthread_exit(void)
{
    kill_pid( task_pid(thread_id), SIGTERM, 1 );
    wait_for_completion( &on_exit );
}
```

# Beispielcode *Doing*

```
#include <linux/module.h>
#include <linux/kthread.h>

static struct task_struct *thread_id;
static wait_queue_head_t wq;
static DECLARE_COMPLETION( on_exit );

static int thread_code( void *data )
{
    unsigned long timeout;
    int i;

    allow_signal( SIGTERM );
    for( i=0; i<5 && (kthread_should_stop() == 0); i++ ) {
        timeout = HZ; /* wait 1 second */
        timeout = wait_event_interruptible_timeout(
            wq, (timeout == 0), timeout);
        printk("thread_function: woke up ... %ld\n", timeout);
        if( timeout == -ERESTARTSYS ) {
            printk("got signal, break\n");
        }
    }
}
```

# Hilfs-Funktionen

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Softirqs und  
Kernel-  
Threads

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
ISR  
Soft-IRQs  
Kernel-  
Threads

- `kthread_should_stop()`
  - Wird als Bedingung in der Schleife verwendet
- `kthread_stop(struct task_struct *k)`
  - Kann aufgerufen werden um den Thread  $k$  anzuhalten (falls dieser das mit dem Aufruf verknüpfte Flag über die Funktion `kthread_should_stop()` überprüft)

# Workqueue

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Softirqs und  
Kernel-  
Threads

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines

ISR

Soft-IRQs

Kernel-  
Threads

- Das Management von zyklischen/periodischen Kernel-Threads ist kompliziert:
  - Anlegen des Workqueue-Objektes
  - Schlafen-Legen
  - ...
- Workqueues sind *vorgefertigte* Kernel-Threads, in deren Kontext Funktionen (einmal) abgearbeitet werden.

# Workqueue (Fortsetzung)

- Zwei Objekt-Klassen sind notwendig:
  - Workqueue-Objekt: repräsentiert den Kernel-Thread
  - Work-Objekt: repräsentiert die abzuarbeitende Funktion.
- Eine dem Workqueue-Objekt übergebene Funktion (Work-Objekt) wird einmal abgearbeitet.
- Falls gewünscht wird pro CPU wird ein Workqueue-Thread angelegt:
  - `create_workqueue()`,  
`create_singlethread_workqueue()`

Achtung: Race-Condition

Falls der Workqueue-Code entladen wird ...

# Workqueue: Beteiligte Funktionen

| Funktion                             | Kommentar                                          |
|--------------------------------------|----------------------------------------------------|
| <code>create_workqueue()</code>      | Kein IR-Kontext                                    |
| <code>flush_workqueue()</code>       | Abarbeitung erzwingen                              |
| <code>destroy_workqueue()</code>     | Freigeben des Objekts                              |
| <code>DECLARE_WORK()</code>          | Statische Definition                               |
| <code>INIT_WORK()</code>             | Initialisierung zur Laufzeit                       |
| <code>queue_work()</code>            | Zur Abarbeitung freigeben                          |
| <code>schedule_work()</code>         | freigeben für<br>Event-Workqueue                   |
| <code>schedule_delayed_work()</code> | freigeben für<br>Event-Workqueue                   |
| <code>flush_scheduled_work()</code>  | Abarbeitung in der<br>Event-Workqueue<br>erzwingen |

# Workqueue Codebeispiel (Teil 1)

```
static struct workqueue_struct *wq;

static void work_queue_func( void *data )
{
    pr_debug( "work_queue_func...\n" );
    return;
}

static DECLARE_WORK( work_obj, work_queue_func, NULL );

static int __init drv_init(void)
{
    wq = create_singlethread_workqueue( "DrvSmpl" );
    if( queue_work( wq, &work_obj ) )
        pr_debug( "queue_work successful ...\n" );
    else
        pr_debug( "queue_work not successful ...\n" );
    return 0;
}
```

# Workqueue Codebeispiel (Teil 2)

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Softirqs und  
Kernel-  
Threads

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
ISR  
Soft-IRQs  
Kernel-  
Threads

```
...
static void __exit drv_exit(void)
{
    pr_debug("drv_exit called\n");
    if( wq )
        destroy_workqueue( wq );
}
```

# Event-Workqueue

Embedded  
Systems -  
Kernelpro-  
grammierung -  
Softirqs und  
Kernel-  
Threads

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
ISR

Soft-IRQs

Kernel-  
Threads

- Für einfache, wenig zeitkritische Aktionen gibt es einen noch einfacheren Mechanismus, eine Funktion im Prozesskontext abarbeiten zu lassen:
  - Für jede CPU instanziert der Kernel eine Workqueue (Event-Workqueue genannt).
  - Der Event-Workqueue kann direkt das Work-Objekt (mit der Funktionsadresse) übergeben werden.

## Funktionen

– `schedule_work()` – `schedule_delayed_work()` –  
`flush_scheduled_work()`

# Event-Workqueue Codebeispiel

```
#include <linux/module.h>

static void work_queue_function( void *data )
{
    pr_debug( "work_queue_function()\n" );
    return;
}

static DECLARE_WORK(work, work_queue_function, NULL );

static int __init mod_init(void)
{
    if( schedule_work( &work )==0 )
        return -EFAULT;
    return 0;
}

static void __exit mod_exit(void)
{
    flush_scheduled_work();
}
```

# Zusammenfassung Kernel-Threads

- Der Kern stellt folgende Konstrukte zur Verfügung, mit denen – unabhängig von einer Applikation – Code im Kernel ausgeführt wird:
  - Tasklets
  - Timer
  - Kernel-Threads
  - Workqueues
  - Event-Workqueue
- Problem: Es muss sichergestellt werden, dass vor dem Entladen eines Moduls keine Objekte des Moduls aktiv sind.

*Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte*

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

# *Embedded Systems - Schutz kritischer Abschnitte*

Jürgen Quade, Hochschule Niederrhein

13.06.2022

## 1 Allgemeines

## 2 Interruptsperrre

## 3 Atomare Operationen

## 4 Mutex

## 5 Spinlock

## 6 Memory Barrier

Prof. Dr.-Ing. Jürgen Quade

Hochschule Niederrhein

## Section 1

### Allgemeines

# Einführung

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

- Durch die Parallelität und Unterbrechbarkeit (Preemption) gibt es Konflikte beim Zugriff auf gemeinsame Betriebsmittel (z.B. auf Speicherbereiche) und bei der Synchronisation.
- Bereits gesehen:
  - Globale Variablen im Treiber
  - Softirqs (Tasklets, Timer)
  - Kernel-Threads (Workqueues, Event-Workqueue)

# Einführung (Fortsetzung)

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

## Bekannte Lösung (1)

- Gegenseitiger Ausschluss (**Mutual Exclusion**)
- Ausprägungen im Kernel
  - Interruptsperrre
  - Atomare Variablen
  - Semaphor/Mutex
    - RT-Mutex
  - Spinlocks
  - Memory Barriers

# Einführung (Fortsetzung)

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

## Bekannte Lösung (2)

- Completion Objects
- Sequence Locks
- Per-CPU Variable

# Einführung (Fortsetzung)

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

## Generelle Problematik

- Sequenzialisierung führt zu **Latenzzeiten!**
  - Stichwort *Prioritätsinversion*
- Verschachtelungen (mehrere Locks werden gehalten) führt zu Deadlocks
- Rekursive Aufrufe
- Codesequenzen müssen *ausnahmslos* gesichert werden
- Höherer Ressourcenverbrauch (Zeit, Speicher)

## Section 2

### Interruptsperrre

# Interrupt- und Preemptionsperre

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

## Preemptionsperre

Verhindert das Scheduling (auf dem lokalen Prozessor)

## Interruptsperre

- Auftretende Interrupts werden zurückgehalten
- Nur für Einprozessor-Systeme geeignet
  - Globale versus lokale Interruptsperrre
- Beide Sperren sollten generell vermieden werden

# Funktionen

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

- `cli()` – Interrupts sperren
- `sti()` – Interrupts freigeben
- `preempt_disable()` – Scheduling deaktivieren
- `preempt_enable()` – Scheduling aktivieren

## Section 3

### Atomare Operationen

# Beschreibung

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

- Mit einer *atomaren Variablen* kann atomar auf einen
  - **Integerwert** (32 oder 64 Bit) oder einen
  - **Bitwert** zugegriffen werden.
- Der Zugriff auf die atomare Variable erfolgt über Funktionen.
- Der Zugriff ist sowohl im Kernel-, Prozess- als auch im Interruptkontext möglich.

# Ablauf

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

- Die Modifikation einer Variablen vollzieht sich in 3 Schritten:
  - Variable in ein internes Register kopieren.
  - Registerinhalt modifizieren.
  - Registerinhalt zurückspeichern.
- Finden während dieser drei Schritte parallele Zugriffe auf eine Variable statt, kann es unter Umständen zu einem falschen Ergebnis kommen (siehe nächste Folie).
- Zum Schutz vor einer derartigen „Race Condition“ muss der Zugriff auf die Variable „atomar“, also ungeteilt stattfinden.

# Problemstellung

Embedded Systems -  
Schutz kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

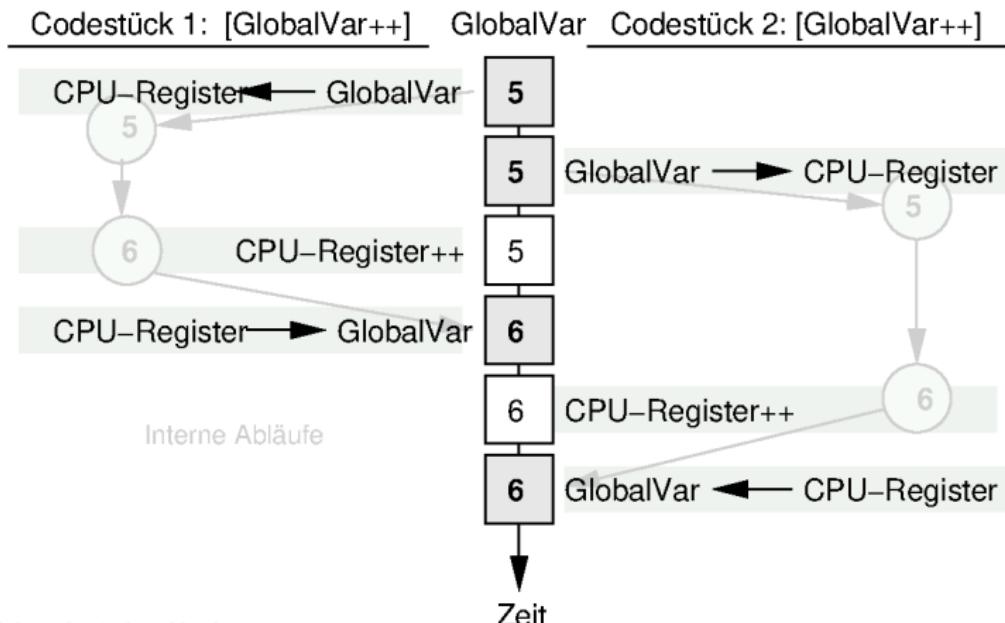
Sequencelock

Completion  
Object

Konfliktvermeidung

Zusammenfassung - Schutz kritischer Abschnitte

Beim parallelen Zugriff auf eine (ungeschützte) Variable kann es zu falschen Ergebnissen kommen.



# Liste von Funktionen

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

## Standardgrößen

```
int atomic_read(atomic_t *v);
void atomic_set(atomic_t *v, int i);
void atomic_add_(int i, atomic_t *v);
void atomic_sub(int i, atomic_t *v);
void atomic_inc(atomic_t *v);
void atomic_dec(atomic_t *v);
int atomic_sub_and_test(int i, atomic_t *v);
int atomic_inc_and_test(atomic_t *v);
int atomic_dec_and_test(atomic_t *v);
int atomic_add_negative(int i, atomic_t *v);
```

# Liste von Funktionen (Fortsetzung)

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

## Bitgrößen

```
void set_bit(int nr, unsigned long *a);
void clear_bit(int nr, unsigned long *a);
void change_bit(int nr, unsigned long *a);
int test_and_set_bit(int nr, unsigned long *a);
int test_and_clear_bit(int nr, unsigned long *a);
int test_and_change_bit(int nr, unsigned long *a);
int test_bit(int nr, void *a);
```

# Liste von Funktionen (Fortsetzung)

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

## Long

```
void set_bit(int nr, unsigned long *a);
int atomic_long_read(atomic_long_t *l);
void atomic_long_set(atomic_long_t *s, long i);
void atomic_long_add_(long i, atomic_long_t *l)
void atomic_long_sub(long i, atomic_long_t *l)
void atomic_long_inc(atomic_long_t *l)
void atomic_long_dec(atomic_long_t *l)
```

## Section 4

### Mutex

# Mutex und Semaphor

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

- Schützt komplexe Datenstrukturen
- Zugriff nur im Prozesskontext
- Mutexe sind deutlich effizienter (weniger Speicher), daher sollten diese Semaphoren vorgezogen werden.

## Funktionsliste

```
DECLARE_MUTEX( mutex );
init_MUTEX( &mutex );
sema_init( &mutex, n );
down( &mutex );
down_interruptible( &mutex );
down_trylock( &mutex );
up( &mutex );
```

# Mutex und Semaphor

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

## Funktionsliste

```
DEFINE_MUTEX(name);  
mutex_init(mutex);  
void mutex_lock(struct mutex *lock);  
int mutex_lock_interruptible(struct mutex *lock);  
int mutex_trylock(struct mutex *lock);  
void mutex_unlock(struct mutex *lock);  
int mutex_is_locked(struct mutex *lock);
```

# Definition und Initialisierung

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines

Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

## Statische Definition, Initialisierung durch den Compiler

```
#include <linux/mutex.h>
...
DEFINE_MUTEX( my_mutex );
```

## Initialisierung zur Laufzeit

```
struct mutex my_mutex;
...
mutex_init( my_mutex );
```

# Mutex im Einsatz

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre  
Atomare  
Operationen

**Mutex**

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

## Kein Abbruch bei Signal

```
mutex_lock( my_mutex );  
... // kritische Codesequenz  
mutex_unlock( my_mutex );
```

# Mutex im Einsatz (Fortsetzung)

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

## Abbruch bei Signal

```
if( mutex_lock_killable( my_mutex ) ) {  
    pr_info("got signal\n");  
    return -ERESTARTSYS;  
}  
... /* kritische Codesequenz */  
mutex_unlock( my_mutex );
```

# Verwendungshinweise

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

- Freigabe nur durch den Besitzer möglich.
- Darf nicht mehrfach freigegeben werden.
- Darf nicht mehrfach (rekursiv) reserviert werden.
- Eine Task darf nicht mit einem gehaltenen Mutex beendet werden.
- Mutexe dürfen nicht im Interrupt-Kontext verwendet werden.
- Reservierte Mutexe dürfen nicht reinitialisiert werden.

# Realtime-Mutex

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

## Mutex mit Prioritätsvererbung

```
rt_mutex_init()  
rt_mutex_destroy()  
rt_mutex_lock()  
rt_mutex_lock_interruptible()  
rt_mutex_timed_lock()  
rt_mutex_trylock()
```

# Prioritätsvererbung

Embedded Systems - Schutz kritischer Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

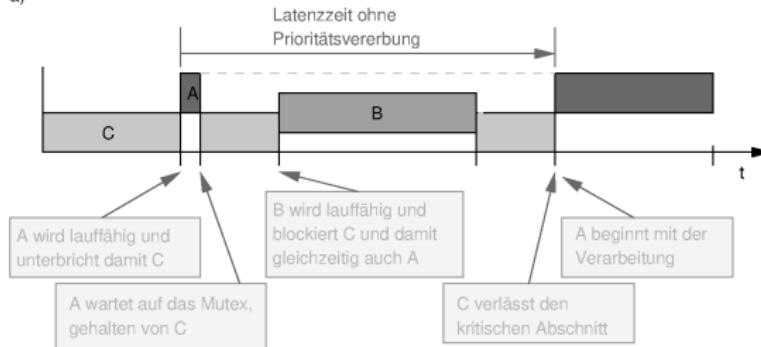
Sequencelock

Completion  
Object

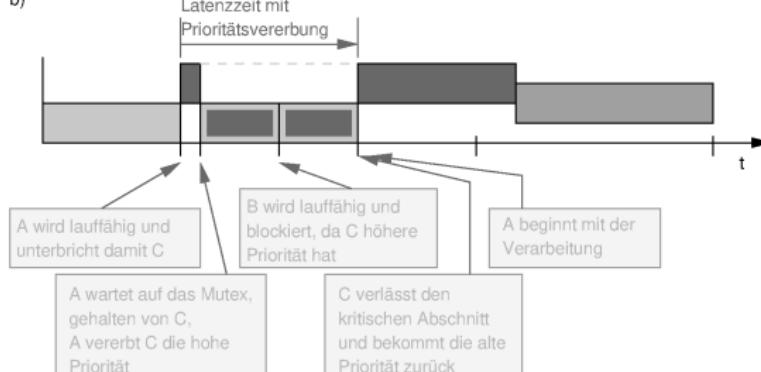
Konfliktvermeidung

Zusammenfassung - Schutz kritischer Abschnitte

a)



b)



# Semaphor: Anwendung/Einsatz

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines

Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

```
while( down_interruptible(&Semaphore) == -EINTR ) {  
    Betreten des kritischen Abschnitts  
    ... // Behandlung des Signals  
}  
... // Modifikation der Daten  
Der kritische Abschnitt selbst  
up( &Semaphore ); Verlassen des kritischen Abschnitts
```

# Ausprägungen

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

- *Normale Semaphore*
- *Schreib-/Lese-Semaphore*
  - `down_read()`
  - `down_read_trylock()`
  - `down_write()`
  - `down_write_trylock()`
  - `downgrade_write()`
  - `up_read()`
  - `up_write()`

## Section 5

### Spinlock

# Aufgaben

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

- Synchronisationselement zum Schutz eines *kurzen* kritischen Abschnitts.
- Realisiert ein aktives Warten.
- Im Prozess- und Interruptkontext einsetzbar!
- In der Theorie nur für Mehrprozessormaschinen geeignet.
- Praktisch auch auf Einprozessormaschinen einsetzbar.

# Initialisierung

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines

Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

## Statische Initialisierung (durch den Compiler)

```
static spinlock_t my_lock = SPIN_LOCK_UNLOCKED;
```

## Dynamische Initialisierung (während der Laufzeit)

```
spin_lock_init( &my_lock );
```

# Ausprägungen

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

```
1  spin_lock()
2  spin_unlock()
3  spin_lock_irq()
4  spin_unlock_irq()
5  spin_lock_irqsave()
6  spin_unlock_irqsave()
7  spin_lock_bh()
8  spin_unlock_bh()
```

# Codebeispiel

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

```
1 #include <asm/spinlock.h>
2 ...
3 unsigned long irqflags;
4 static struct spinlock_t my_lock=SPIN_LOCK_UNLOCKED;
5 ...
6 ...
7     spin_lock_irqsave( &my_lock, &irqflags );
8     ... // kritischer Abschnitt
9     spin_unlock_irqrestore( &my_lock, &irqflags );
10    ...
```

# Weitere Ausprägungen

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

```
1  read_lock, write_lock
2  read_unlock, write_unlock
3  read_lock_irq, write_lock_irq
4  read_unlock_irq, write_unlock_irq
5  read_lock_irqsave, write_lock_irqsave
6  read_unlock_irqrestore, write_unlock_irqrestore
7  read_lock_bh, write_lock_bh
8  read_unlock_bh, write_unlock_bh
```

# Einsatz

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

## Beispiel

Kritischer Abschnitt zwischen einer Treiberfunktion  
`driver_read()` und einer ISR.

## Denkbare Szenarien

- ① Gleichzeitiger Aufruf von `driver_read()` auf zwei CPU-Kernen.
- ② Aufruf der ISR auf einem anderen Kern.
- ③ Gleichzeitiger Aufruf von `driver_read()` auf dem eigenen, lokalen CPU-Kern (Preemption).
- ④ Unterbrechung durch die ISR auf der lokalen CPU (Preemption).

# Methoden

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

- Szenario 1 und 2 (Paralleler Aufruf auf einer zweiten CPU) lässt sich durch ein einfaches Spinlock schützen.
- Szenario 3 und 4 (Paralleler Aufruf auf der lokalen CPU) lässt sich durch das einfache Spinlock nicht schützen:
  - Interrupts auf der lokalen CPU müssen zusätzlich gesperrt werden.

# Codebeispiel spin\_lock\_irq()

```
1 static ssize_t driver_read( struct file *instanz, char __user *us
2                         size_t count, loff_t *offset )
3 {
4     ...
5     spin_lock_irq( &mylock );
6     /* Auf der lokalen CPU sind jetzt zusätzlich Interrupts g
7     ... /* Kritischer Abschnitt */
8     spin_unlock_irq( &mylock );
9     ...
10 }
11
12 irqreturn_t interrupt_service_routine( int irq, void *dev_id )
13 {
14     ...
15     /* Interrupts sind bereits gesperrt, spin_lock reicht aus.
16     spin_lock( &mylock );
17     ... /* Kritischer Abschnitt */
18     spin_unlock( &mylock );
19     ...
20 }
```

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

Zusammenfassung } Schutz kritischer Abschnitte

# Fragen

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

## Frage 1

Warum werden im vorherigen Beispiel innerhalb der ISR nur die Funktionen `spin_lock()` und `spin_unlock()`, nicht jedoch die Varianten `spin_lock_irq()` und `spin_unlock_irq()` verwendet?

## Frage 2

Wann verwenden Sie die Varianten `spin_lock_irqsave()` und `spin_unlock_irqsave()`?

# Beispiel irqsav

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines

Interruptsperr

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

```
1 static void modify_private_list( void )
2 {
3     unsigned long iflags;
4
5     spin_lock_irqsave( &mylock, &iflags );
6     ... /* Kritischer Abschnitt */
7     spin_unlock_irqrestore( &mylock, &iflags );
8 }
9 ...
```

# Beispiel irqsave (Fortsetzung)

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte  
  
Jürgen Quade,  
Hochschule  
Niederrhein  
  
Allgemeines  
  
Interruptsperrre  
  
Atomare  
Operationen  
  
Mutex  
  
Spinlock  
  
Memory  
Barrier  
  
Sequencelock  
  
Completion  
Object  
  
Konfliktvermeidung

```
1 static ssize_t driver_read( struct file *instanz,
2                             char __user *userbuffer, size_t count, loff_t *offset )
3 {
4     ...
5     modify_private_list();
6     ...
7 }
8
9 irqreturn_t interrupt_service_routine( int irq, void *dev_id )
10 {
11     ...
12     modify_private_list();
13     ...
14 }
```

# Frage bh

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

Wann verwenden Sie die Varianten `spin_lock_bh()` und `spin_unlock_bh()`?

## Auswahl des Schutzelements

| Treiberinstanz   |                  | Kernel-Thread    | Workqueue        | Sofitirq         | Tasklet                                                          | Hardirq                                                |
|------------------|------------------|------------------|------------------|------------------|------------------------------------------------------------------|--------------------------------------------------------|
|                  |                  | Event-Workqueue  |                  |                  | Timer                                                            |                                                        |
| Mutex            | Mutex            |                  |                  |                  | Spinlock-bh<br>RW-Lock-bh<br>Spinlock-irqsave<br>RW-Lock-irqsave | Spinlock-irqsave<br>RW-Lock-irqsave<br>Seqlock-irqsave |
| RT-Mutex         | RT-Mutex         |                  |                  |                  | Spinlock-bh<br>RW-Lock-bh<br>Spinlock-irqsave<br>RW-Lock-irqsave | Spinlock-irqsave<br>RW-Lock-irqsave<br>Seqlock-irqsave |
| RW-Semaphore     | RW-Semaphore     |                  |                  |                  | Spinlock-bh<br>RW-Lock-bh<br>Spinlock-irqsave<br>RW-Lock-irqsave | Spinlock-irqsave<br>RW-Lock-irqsave<br>Seqlock-irqsave |
| Spinlock         | Spinlock         |                  |                  |                  | Spinlock-bh<br>RW-Lock-bh<br>Spinlock-irqsave<br>RW-Lock-irqsave | Spinlock-irqsave<br>RW-Lock-irqsave<br>Seqlock-irqsave |
| RW-Lock          | RW-Lock          |                  |                  |                  | Spinlock-bh<br>RW-Lock-bh<br>Spinlock-irqsave<br>RW-Lock-irqsave | Spinlock-irqsave<br>RW-Lock-irqsave<br>Seqlock-irqsave |
| Seqlock          | Seqlock          |                  |                  |                  | Spinlock-bh<br>RW-Lock-bh<br>Spinlock-irqsave<br>RW-Lock-irqsave | Spinlock-irqsave<br>RW-Lock-irqsave<br>Seqlock-irqsave |
| Treiberinstanz   | Mutex            |                  |                  |                  | Spinlock-bh<br>RW-Lock-bh<br>Spinlock-irqsave<br>RW-Lock-irqsave | Spinlock-irqsave<br>RW-Lock-irqsave<br>Seqlock-irqsave |
|                  | RT-Mutex         |                  |                  |                  | Spinlock-bh<br>RW-Lock-bh<br>Spinlock-irqsave<br>RW-Lock-irqsave | Spinlock-irqsave<br>RW-Lock-irqsave<br>Seqlock-irqsave |
| Kernel-Thread    | RW-Semaphore     |                  |                  |                  | Spinlock-bh<br>RW-Lock-bh<br>Spinlock-irqsave<br>RW-Lock-irqsave | Spinlock-irqsave<br>RW-Lock-irqsave<br>Seqlock-irqsave |
|                  | Spinlock         |                  |                  |                  | Spinlock-bh<br>RW-Lock-bh<br>Spinlock-irqsave<br>RW-Lock-irqsave | Spinlock-irqsave<br>RW-Lock-irqsave<br>Seqlock-irqsave |
| Kernel-Workqueue | RW-Lock          |                  |                  |                  | Spinlock-bh<br>RW-Lock-bh<br>Spinlock-irqsave<br>RW-Lock-irqsave | Spinlock-irqsave<br>RW-Lock-irqsave<br>Seqlock-irqsave |
|                  | Seqlock          |                  |                  |                  | Spinlock-bh<br>RW-Lock-bh<br>Spinlock-irqsave<br>RW-Lock-irqsave | Spinlock-irqsave<br>RW-Lock-irqsave<br>Seqlock-irqsave |
| Workqueue        | Spinlock         |                  |                  |                  | Spinlock-bh<br>RW-Lock-bh<br>Spinlock-irqsave<br>RW-Lock-irqsave | Spinlock-irqsave<br>RW-Lock-irqsave<br>Seqlock-irqsave |
|                  | RW-Lock          |                  |                  |                  | Spinlock-bh<br>RW-Lock-bh<br>Spinlock-irqsave<br>RW-Lock-irqsave | Spinlock-irqsave<br>RW-Lock-irqsave<br>Seqlock-irqsave |
| Event-Workqueue  | Seqlock          |                  |                  |                  | Spinlock-bh<br>RW-Lock-bh<br>Spinlock-irqsave<br>RW-Lock-irqsave | Spinlock-irqsave<br>RW-Lock-irqsave<br>Seqlock-irqsave |
|                  |                  |                  |                  |                  | Spinlock-bh<br>RW-Lock-bh<br>Spinlock-irqsave<br>RW-Lock-irqsave | Spinlock-irqsave<br>RW-Lock-irqsave<br>Seqlock-irqsave |
| Sofitirq         | Spinlock-bh      | Spinlock-bh      | Spinlock-bh      | Spinlock         | Spinlock                                                         | Spinlock-irqsave<br>RW-Lock-irqsave<br>Seqlock-irqsave |
|                  | RW-Lock-bh       | RW-Lock-bh       | RW-Lock-bh       |                  |                                                                  | Spinlock-irqsave<br>RW-Lock-irqsave<br>Seqlock-irqsave |
| Tasklet          | Spinlock-irqsave | Spinlock-irqsave | Spinlock-irqsave | Spinlock         | Spinlock                                                         | Spinlock-irqsave<br>RW-Lock-irqsave<br>Seqlock-irqsave |
|                  | RW-Lock-irqsave  | RW-Lock-irqsave  | RW-Lock-irqsave  |                  |                                                                  | Spinlock-irqsave<br>RW-Lock-irqsave<br>Seqlock-irqsave |
| Timer            | Seqlock-irqsave  | Seqlock-irqsave  | Seqlock-irqsave  | Spinlock         | Spinlock                                                         | Spinlock-irqsave<br>RW-Lock-irqsave<br>Seqlock-irqsave |
|                  |                  |                  |                  |                  |                                                                  | 1)                                                     |
| Hardirq          | Spinlock-irqsave | Spinlock-irqsave | Spinlock-irqsave | Spinlock-irqsave | Spinlock-irqsave                                                 | Spinlock                                               |
|                  | RW-Lock-irqsave  | RW-Lock-irqsave  | RW-Lock-irqsave  | RW-Lock-irqsave  | RW-Lock-irqsave                                                  | RW-Lock                                                |
| Hardirq          | Seqlock-irqsave  | Seqlock-irqsave  | Seqlock-irqsave  | Seqlock-irqsave  | Seqlock-irqsave                                                  | Seqlock                                                |
|                  |                  |                  |                  |                  |                                                                  |                                                        |

- 1) Nur bei unterschiedlichen Tasklets. Ein Tasklet läuft zu einem Zeitpunkt garantiert nicht zweimal.
- 2) Nur bei unterschiedlichen ISRs. Ein und dieselbe ISR läuft zu einem Zeitpunkt nicht zweimal ab.

## Section 6

### Memory Barrier

# Problematik Reorderung

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

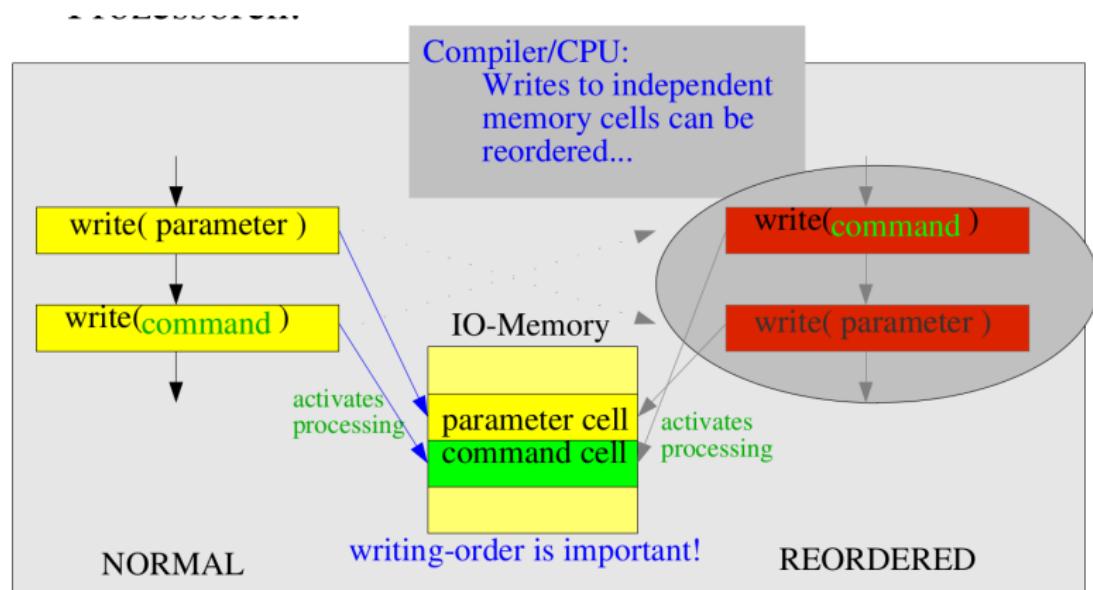
Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

Zusammenfassung - Schutz kritischer Abschnitte



# Lösung

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

- Spinlocks/Semaphore
- Besser: Memory Barrier

## Wirkungsweise

*Memory Barrier* stellen sicher, dass die programmierte Abarbeitungsreihenfolge eingehalten wird.

## Befehle

```
mb();           // memory barrier
rmb();          // read memory barrier
wmb();          // write memory barrier
barrier();      // verhindert Reordering durch den Compiler
```

# Beispiel

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

```
wmb();  
outw( DATA_PORT, 0x0001 ); // write parameter  
wmb();  
outw( COMMAND_PORT, 0x1122 ); // write command
```

## Section 7

### Sequencelock

# Eigenschaften

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

- Abwandlung der Read-Write-Locks
- Ermöglichen den schnellen, effizienten Zugriff auf Daten
  - kurze Latenzzeiten
- Geeignet für sehr kurze, kritische Abschnitte

# Grundprinzip

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

- Schreibender Zugriff wird (einem Thread) direkt gewährt.  
Bei parallelen Schreibzugriffen: aktives Warten
- Kein Blockieren beim lesenden Zugriff
- Nach dem Zugriff wird die Gültigkeit der gelesenen Werte überprüft
- Bei ungültigen (modifizierten) Werten wird erneut gelesen

# Struktogramme

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

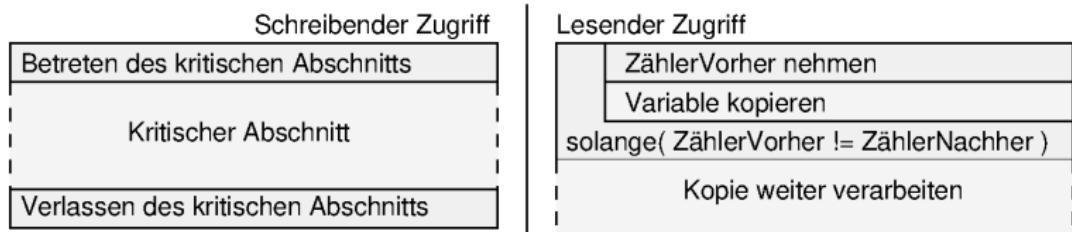
Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

Zusammenfassung - Schutz kritischer Abschnitte



# Einsatz

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines

Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

## Schreiben

```
1 ...
2 write_seqlock_irq( &time_lock );
3 time.tv_sec = value;
4 time.tv_nsec= 0;
5 write_sequnlock_irq( &time_lock );
6 ...
```

## Lesen

```
1 ...
2 unsigned long seq;
3
4 do {
5     seq = read_seqbegin( &time_lock );
6     now = time;
7 } while ( read_seqretry( &time_lock, seq ) );
```

# Varianten

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

```
1 write_seqlock_irqsave()
2 write_seqlock_irq()
3 write_seqlock_bh()
4 write_sequnlock_irqrestore()
5 write_sequnlock_irq()
6 write_seqlock_bh()
```

## Section 8

### Completion Object

# Beschreibung

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines

Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

Dient der Synchronisation zwischen zwei Jobs: – Job 2 wartet darauf, dass Job 1 eine Aktion beendet.

## Funktionsliste

```
1 DECLARE_COMPLETION()      // statische Initialisierung (Compiler)
2 init_completion()          // dynamische Initialisierung (Laufzeit)
3 complete()                 // signalisiert das Ende der Codesequenz
4 complete_all()             // signalisiert das Ende der Codesequenz
5 complete_and_exit()        // Codesequenz ende zusammen mit Threadende
6 wait_for_completion()      // schlafe auf das Ende der Codesequenz
```

# Codebeispiel

Embedded Systems - Schutz kritischer Abschnitte  
Jürgen Quade, Hochschule Niederrhein  
Allgemeines Interruptspeicher  
Atomare Operationen  
Mutex  
Spinlock  
Memory Barrier  
Sequencelock  
Completion Object  
Konfliktvermeidung

```
1  DECLARE_COMPLETION( on_exit );
2  ...
3  static int kernelthread( void *data )
4  {
5      ...
6      if( signal_pending( current ) ) {
7          complete_and_exit( &on_exit, 0 );
8          return 0;
9      }
10     ...
11     complete_and_exit( &on_exit, 0 );
12     return 0;
13 }
14 ...
15 static void __exit mod_exit( void )
16 {
17     kill_proc( thread_id, SIGTERM );
18     wait_for_completion( &on_exit );
19     ...
20 }
```

## Section 9

### Konfliktvermeidung

# Prinzip *Per-CPU-Variable*

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeid

- Besser als der *Schutz kritischer Abschnitte* ist, erst gar keine kritische Abschnitte zu haben.
- Kritische Abschnitte lassen sich manchmal vermeiden, in dem man ausreichend Ressourcen (Betriebsmittel) zur Verfügung stellt.
  - *ausreichend*: Soviele Ressourcen wie maximal parallel verwendet werden können.
  - maximale (echte) Parallelität ist durch die Anzahl der Prozessorkerne (CPU) limitiert.
- Strategie : Kritische Datenstrukturen für jeden CPU-Kern einmal zur Verfügung stellen
  - Per-CPU-Variable

## Section 10

### Zusammenfassung

# Zusammenfassung

Embedded  
Systems -  
Schutz  
kritischer  
Abschnitte

Jürgen Quade,  
Hochschule  
Niederrhein

Allgemeines  
Interruptsperrre

Atomare  
Operationen

Mutex

Spinlock

Memory  
Barrier

Sequencelock

Completion  
Object

Konfliktvermeidung

- Linux bietet eine Reihe unterschiedlicher Methoden zum Schutz kritischer Abschnitte.
- Die Vielfalt resultiert aus dem Wunsch, ein möglichst **preiswertes** Verfahren anbieten zu können.
- Das Erkennen und das daraus erfolgende Absichern von kritischen Abschnitten ist eine der großen Herausforderungen bei der Kernel-Programmierung.
- Zusätzlich zu den vorgestellten Möglichkeiten gibt es noch das Sequence-Lock.

# *Embedded Systems - Industrial IO*

Jürgen Quade, Hochschule Niederrhein

14.06.2022

## 1 Hintergrund

## 2 Architektur

## 3 Addons

## 4 Treibertechnik

## 5 Praxis

**Prof. Dr.-Ing. Jürgen Quade**  
**Hochschule Niederrhein**

## Section 1

### Hintergrund

# Definition

- Mit Industrial IO wird ein im Linux-Kernel integriertes Sensor-Interface bezeichnet, das einen professionellen Zugang zu Daten ermöglicht.
- Sensorwerte sind bezüglich *Auflösung* und *Repräsentation* (Ganzzahl, Float) standardisiert.
- Falls vom Treiber unterstützt können Sensorwerte automatisiert inklusive Auftrittszeitpunkt erfasst werden.
- Zugriff auf Daten über das Sys-Filesystem. Alternativ gibt es eine **libiio** und einen Satz Werkzeuge (Libiio-Utils).
- Unterstützung für mehrere hundert (>300) unterschiedliche Sensoren und einige Aktoren.

# Problemstellung

- Einheitliches Interface für den *professionellen* Umgang mit Daten
  - Festlegung von Datentypen
  - Attributierung um Zeitstempel

## Beispiel Temperatur

- Wird die Temperatur in Celsius oder Fahrenheit zurückgegeben?
- Welche Auflösung hat der Temperaturwert? Grad oder Milligrad?

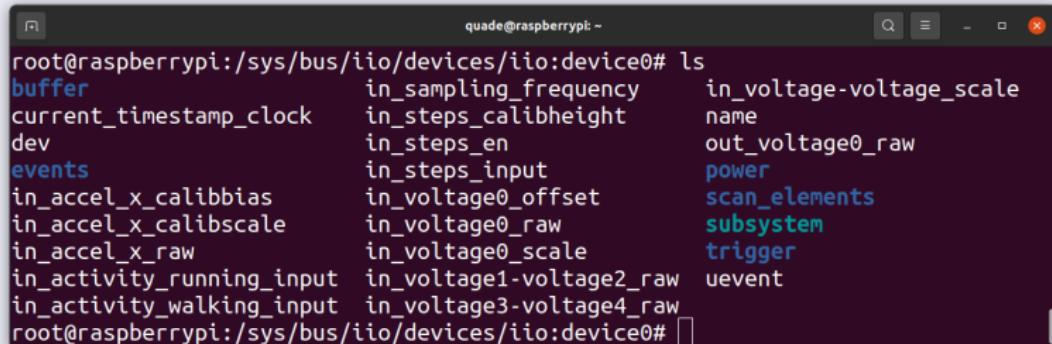
# Kategorisierung

- Pressure
- Temperature
- Acceleration
- Proximity
- Chemical
- ADC
- DAC

# Zugriff

Zugriff erfolgt über das Sys-Filesystem, z.B. per cat

```
/sys/bus/iio/  
/sys/bus/iio/devices/  
/sys/bus/iio/devices/iio:device0/  
/sys/bus/iio/devices/iio:device1/  
/sys/bus/iio/devices/iio:device2/
```



```
quade@raspberrypi: ~  
root@raspberrypi:/sys/bus/iio/devices/iio:device0# ls  
buffer                      in_sampling_frequency    in_voltage-voltage_scale  
current_timestamp_clock      in_steps_calibheight   name  
dev                          in_steps_en            out_voltage0_raw  
events                       in_steps_input          power  
in_accel_x_calibbias        in_voltage0_offset     scan_elements  
in_accel_x_calibscale       in_voltage0_raw         subsystem  
in_accel_x_raw               in_voltage0_scale      trigger  
in_activity_running_input   in_voltage1-voltage2_raw uevent  
in_activity_walking_input   in_voltage3-voltage4_raw  
root@raspberrypi:/sys/bus/iio/devices/iio:device0#
```

# Kanäle

- Moderne Sensoren liefern häufig mehrere Meßwerte (BME280: Luftdruck, Temperatur, Luftfeuchte)
- Unterschiedliche Meßwerte werden auf *Kanäle* abgebildet

# Kanäle (Fortsetzung)

## Kanalname

- Namensstamm (accel\_x, Beschleunigung in X-Richtung)
- Präfix
  - in\_ (Eingang)
  - out\_ (Ausgang)
- Suffix
  - \_raw (unskaliert)
  - \_offset
  - \_scale (Skalierung)
  - \_ratio (Verhältnis)

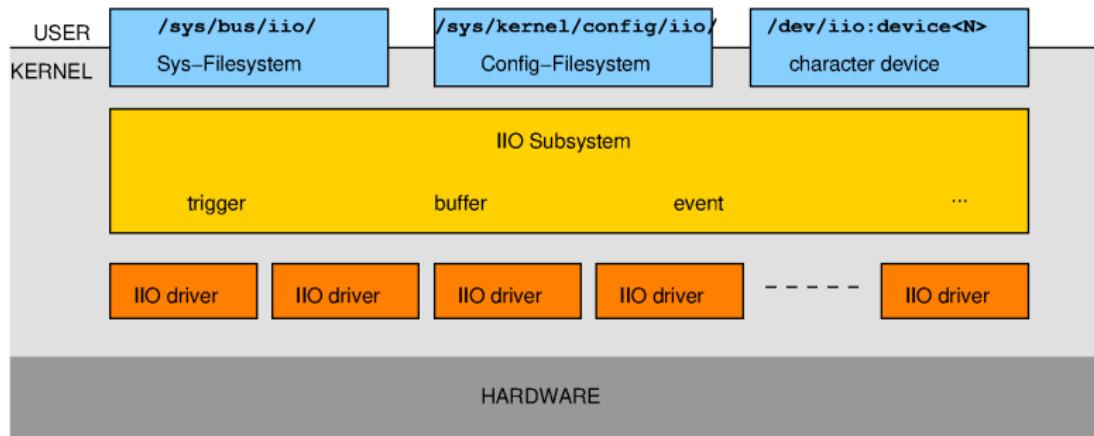
## Berechnung

```
value = (raw+offset)*scale
```

## Section 2

### Architektur

# Architektur



## Section 3

### Addons

# Trigger und Buffer

- Das Industrial-IO-Subsystem ermöglicht (falls vom Treiber unterstützt) die automatische Erfassung von Meßwerten
- Anwender definieren Trigger-Ereignisse
  - Abtastfrequenz
- Einlesen erfolgt über die Gerätedatei

## Section 4

### Treibertechnik

# Grundprinzip

Embedded  
Systems -  
Industrial IO

Jürgen Quade,  
Hochschule  
Niederrhein

Hintergrund

Architektur

Addons

Treibertechnik

Praxis

## Traditioneller Treiber

- Implementierung der Funktionen
  - `driver_open()`
  - `driver_read()`
  - `driver_write()`
  - `driver_close()`

## IIO Device

- Geräte werden über Datenstrukturen beschrieben.
- Implementierung der Funktionen
  - `read_raw()`
  - `write_raw()`
- Austausch der Daten über (zwei) vordefinierte Variablen

# Treiber-Implementierung IIO Device

Embedded  
Systems -  
Industrial IO

Jürgen Quade,  
Hochschule  
Niederrhein

Hintergrund

Architektur

Addons

Treibertechnik

Praxis

- Innerhalb von `read_raw()` findet der Hardware-Zugriff statt.
- Transfervariablen werden als Pointer auf Integervariablen übergeben (`val` und `val2`)
  - `val` enthält den Vorkommateil
  - `val2` enthält den Nachkommanteil
- Der Returnwert spezifiziert die Repräsentationsform

## Ergebnis-Repräsentationsformen

- `IIO_VAL_INT`
- `IO_VAL_INT_PLUS_NANO`
- `IO_VAL_INT_PLUS_MIKRO`
- ...

# Codebeispiel

```
static int foo_iio_read_raw(struct iio_dev *indio_dev,
    struct iio_chan_spec const *channel,
    int *val, int *val2, long mask)
{
    if (channel->channel==0) {
        read_voltage_from_hardware( val, val2 );
        return IIO_VAL_INT_PLUS_MICRO;
    }
    return -EINVAL;
}
```

# Beschreibung der IIO-Kanäle

```
static const struct iio_chan_spec foo_channels[] = {
{
    .type = IIO_VOLTAGE,
    .indexed = 1,
    .channel = 0,
    .address = 0,
    .info_mask_separate = BIT(IIO_CHAN_INFO_RAW),
    .info_mask_shared_by_type = BIT(IIO_CHAN_INFO_SCALE)
        | BIT(IIO_CHAN_INFO_OFFSET),
},
{
    .type = IIO_VOLTAGE,
    .indexed = 1,
    .channel = 1,
    .address = 1,
    .info_mask_separate = BIT(IIO_CHAN_INFO_RAW),
    .info_mask_shared_by_type = BIT(IIO_CHAN_INFO_SCALE),
},
IIO_CHAN_SOFT_TIMESTAMP(2),
```

# Initialisierung

```
static int foo_pdrv_probe (struct platform_device *pdev)
{
    int ret;
    struct iio_dev *indio_dev;
    struct foo_private_data *data;

    indio_dev = devm_iio_device_alloc(&pdev->dev, sizeof(*data));
    if (!indio_dev)
        return -ENOMEM;
    indio_dev->dev.parent = &pdev->dev;
    indio_dev->info = &foo_iio_info;
    indio_dev->name = KBUILD_MODNAME;
    indio_dev->modes = INDIO_DIRECT_MODE;
    indio_dev->channels = foo_channels;
    indio_dev->num_channels = ARRAY_SIZE(foo_channels);
    ...
}
```

# Initialisierung (Fortsetzung)

```
...
    ret = iio_device_register(indio_dev);
    if (ret < 0) {
        return -EINVAL;
    }
    platform_set_drvdata(pdev, indio_dev);
    return 0;
}
```

## Section 5

Praxis

# Sensor BME280

Embedded  
Systems -  
Industrial IO

Jürgen Quade,  
Hochschule  
Niederrhein

Hintergrund

Architektur

Addons

Treibertechnik

Praxis

- Sensorwerte: Luftdruck, Luftfeuchtigkeit und Temperatur
- Sensor wird per  $I^2C$  oder SPI angeschlossen
- Fixe Adresse am  $I^2C$ -Bus 0x77 (oder 0x76)
- Verschaltung über insgesamt vier Leitungen
- Treiber BMP280
- Treiber wird von Raspi-OS direkt mitgeliefert

# Verschaltung

Embedded  
Systems -  
Industrial IO

Jürgen Quade,  
Hochschule  
Niederrhein

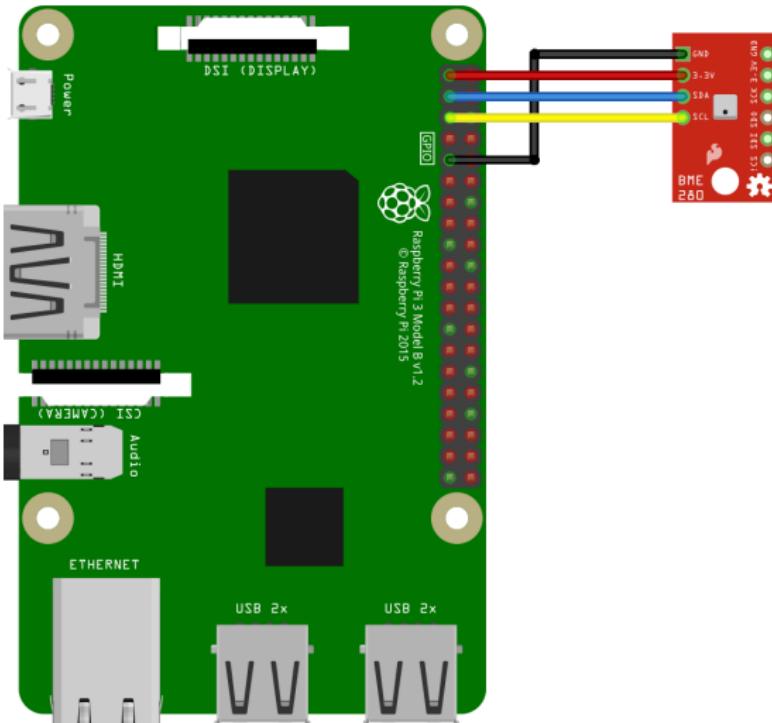
Hintergrund

Architektur

Addons

Treibertechnik

Praxis



fritzing

# Aktivierung

Um auf das Gerät zugreifen zu können, muss das Gerät z.B. per echo "bme280 0x77" bekannt gegeben werden.

The screenshot shows a terminal window titled 'pi@raspberrypi: ~'. The terminal displays the following command sequence:

```
pi@raspberrypi:~ $ sudo su
root@raspberrypi:/home/pi# i2cdetect -y 1
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  -
10:  -
20:  -
30:  -
40:  -
50:  -
root@raspberrypi:/home/pi# echo "bme280 0x77" > /sys/bus/i2c/devices/i2c-1/new_device
root@raspberrypi:/home/pi# i2cdetect -y 1
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  -
10:  -
20:  -
30:  -
40:  -
50:  -
60:  -
70:  - UU
root@raspberrypi:/home/pi#
```

# Aktivierung

*Embedded  
Systems -  
Industrial IO*

Jürgen Quade,  
Hochschule  
Niederrhein

Hintergrund

Architektur

Addons

Treibertechnik

Praxis

- Durch die Bekanntgabe werden die benötigten Treiber automatisch geladen.
- Zugriff ist über das IIO-Subsystem direkt möglich.

# iio\_info

Embedded  
Systems -  
Industrial IO

Jürgen Quade,  
Hochschule  
Niederrhein

Hintergrund

Architektur

Addons

Treibertechnik

Praxis

```
pi@raspberrypi:~$ root@raspberrypi:/proc/device-tree/soc/gpio@7e200000# iio_info
Library version: 0.16 (git tag: v0.16)
Compiled with backends: local xml ip usb serial
IIO context created with local backend.
Backend version: 0.16 (git tag: v0.16)
Backend description string: Linux raspberrypi 5.10.17-v7l+ #1414 SMP Fri Apr 30 13:20
:47 BST 2021 armv7l
IIO context has 1 attributes:
    local,kernel: 5.10.17-v7l+
IIO context has 2 devices:
    iio:device0: bme280
        3 channels found:
            humidityrelative: (input)
            2 channel-specific attributes found:
                attr 0: input value: 53224
                attr 1: oversampling_ratio value: 16
            pressure: (input)
            2 channel-specific attributes found:
                attr 0: input value: 101.437328125
                attr 1: oversampling_ratio value: 16
            temp: (input)
            2 channel-specific attributes found:
                attr 0: input value: 27190
                attr 1: oversampling_ratio value: 2
    iio:device1: srf04
        1 channels found:
            distance: (input)
            2 channel-specific attributes found:
                attr 0: raw value: 1401
                attr 1: scale value: 0.001000
root@raspberrypi:/proc/device-tree/soc/gpio@7e200000#
```

# Sensor HC-SR04

Embedded  
Systems -  
Industrial IO

Jürgen Quade,  
Hochschule  
Niederrhein

Hintergrund

Architektur

Addons

Treibertechnik

Praxis

- Ultraschallsensor zur Abstandsmessung
- 5V Technik

## Ansteuerung

- 10 us Signal Trigger
- Zeitdauer messen für Signal am Echo-Eingang
- Zeitdauer korreliert mit der Entfernung

# Problem

- Treiber ist zwar im Linux-Quellcode vorhanden, aber nicht standardmäßig aktiviert.
- Treiber ist ein Plattform-Treiber
  - Gerät muss z.B. per Devicetree bekannt gegeben werden.

# Generierung des Treibers

## Lösung 1

- ① Treiber in der Kernel-Konfiguration aktivieren
- ② Kernel-Module neu generieren
- ③ Kernel-Module installieren

## Lösung 2

- Treiber-Quellcode in ein separates Verzeichnis kopieren
- Makefile erstellen
- Treiber kompilieren
- Treiber per insmod laden.

# Devicetree

```
/dts-v1/;  
/plugin/;  
  
/ {  
    compatible = "brcm,bcm2835", "brcm,bcm2711";  
  
    fragment@0 {  
        target-path = "/";  
  
        __overlay__ {  
            proximity {  
                compatible = "devantech,srf04";  
                status = "okay";  
                trig-gpios = < &gpio 5 0 >;  
                echo-gpios = < &gpio 6 0 >;  
            };  
        };  
    };  
};
```

# Devicetree (Fortsetzung)

*Embedded  
Systems -  
Industrial IO*

Jürgen Quade,  
Hochschule  
Niederrhein

Hintergrund  
Architektur

Addons

Treibertechnik

Praxis

- Devicetree wird generiert und danach in den Kernel geladen

```
dtc srf04.dts -o srf04.dtbo  
dtoverlay srf04.dtbo
```

# Automatisierung des Ladens des Devicetree-Overlays

Embedded  
Systems -  
Industrial IO

Jürgen Quade,  
Hochschule  
Niederrhein

Hintergrund  
Architektur  
Addons

Treibertechnik  
Praxis

## Raspi OS

- Kopieren des Devicetree-Overlays ins Boot Verzeichnis
- Ergänzen der Konfiguration in /boot/config.txt

```
cp srf04.dtbo /boot/overlays/  
echo "dtoverlay=srf04" >> /boot/config.txt
```

# Result

*Embedded  
Systems -  
Industrial IO*

Jürgen Quade,  
Hochschule  
Niederrhein

Hintergrund

Architektur

Addons

Treibertechnik

Praxis

Mit der Bekanntgabe des Gerätes ist der Zugriff über das Sys-Filesystem möglich.

## *Embedded Systems - MQTT*

Jürgen Quade, Hochschule Niederrhein

28.06.2022

## 1 Defintion

## 2 Architektur

## 3 MQTT in der Praxis

Prof. Dr.-Ing. Jürgen Quade  
Hochschule Niederrhein

## Section 1

Defintion

# Defintion

*Message Queuing Telemetry Transport (MQTT) ist ein offenes Netzwerkprotokoll für Machine-to-Machine-Kommunikation (M2M), das die Übertragung von Telemetriedaten in Form von Nachrichten zwischen Geräten ermöglicht, trotz hoher Verzögerungen oder beschränkter Netzwerke.<sup>[1]</sup> Entsprechende Geräte reichen von Sensoren und Aktoren, Mobiltelefonen, Eingebetteten Systemen in Fahrzeugen oder Laptops bis zu voll entwickelten Rechnern.*

# Defintion (Fortsetzung)

Embedded  
Systems -  
MQTT

Jürgen Quade,  
Hochschule  
Niederrhein

Defintion

Architektur

MQTT in der  
Praxis

*Interessant ist, dass ein MQTT-Server („Broker“) die gesamte Datenlage seiner Kommunikationspartner hält, und so als Zustands-Datenbank benutzt werden kann. So ist es möglich, kleine unperformante MQTT-Geräte mit einem MQTT-Broker zu verbinden, wobei die Geräte Daten einsammeln und/oder Befehle entgegennehmen, während ein komplexes Lagebild nur auf dem MQTT-Broker entsteht und hier oder durch einen leistungsfähigen Kommunikationspartner ausgewertet werden kann. Stelleingriffe können so von einer oder mehreren leistungsfähigen Instanzen an den MQTT-Broker übermittelt und auf die einzelnen Geräte verbreitet werden. Dadurch eignet sich MQTT sehr gut für Automatisierungslösungen und findet im Bereich IoT durch die einfache Verwendung große Verbreitung.*

[Quelle: <https://de.wikipedia.org/wiki/MQTT>]

# Definition (Fortsetzung)

MQTT ist ein Protokoll zwischen Server und Clients. Der Server ist der MQTT-Broker, der die Verteilstation von Nachrichten nach dem Publisher-Subscriber-Prinzip darstellt. Publisher versenden ihre Daten unter einem Topic an den Broker. Subscriber (Clients) abonnieren beim Broker Topics. Sobald für ein Topic eine neue Nachricht ankommt, verteilt der Broker diese an die Clients, die sich auf die Nachricht abonniert haben.

## Section 2

### Architektur

# Topics

Embedded  
Systems -  
MQTT

Jürgen Quade,  
Hochschule  
Niederrhein

Definition

Architektur

MQTT in der  
Praxis

Die Topics sind ähnlich den Dateipfaden hierarchisch organisiert, vom Systemarchitekten aber frei wählbar.

## Beispiel für Topics

system/core/ipadress system/core/hostname  
motorcontrol/motorleft/rpm

## Weitere Infos

<https://www.embedded-software-engineering.de/was-ist-mqtt-a-725485/>

## Section 3

MQTT in der Praxis

# Mosquitto

Embedded  
Systems -  
MQTT

Jürgen Quade,  
Hochschule  
Niederrhein

Definition

Architektur

MQTT in der  
Praxis

Mit `mosquitto` steht sowohl die Implementierung eines Brokers als auch von Clients zur Verfügung. `mosquitto` ist bereits in Buildroot integriert und muss nur ausgewählt werden.

Mit den Programmen `mosquitto_pub` und `mosquitto_sub` stehen Publisher- und Subscriber-Programme zur Verfügung.

Für Tablets und Smartphone gibt es diverse MQTT-Dashboards, z.B. *MQTT Dash* für Android.

**Um per Smartphone auf den Broker zugreifen zu können, muss sich das Smartphone im gleichen Netz befinden. Buchen Sie also Ihr Smartphone ins WLAN des Raspberry Pi ein!**

Bei den Apps ist als erstes der Broker zu kontaktieren.

# Beispiel: Ansteuerung einer LED über MQTT

Um kein eigenes C-Programm schreiben zu müssen, wird im Rahmen eines Rapid-Prototyping die LED per Shell-Skript angesteuert. Das Shell-Skript liest von STDIN Nachrichten und schreibt diese in die zur LED gehörenden Gerätedatei (`/dev/led_one`).

```
mosquitto_sub -t led -h 192.168.222.1 | ./bash.script
```

# Beispiel Ansteuerung einer LED

## Bash-Skript

```
#!/bin/sh

while true
do
    echo "waiting for input ..."
    read
    echo "switching led to ${REPLY}"
    echo -e -n "\x${REPLY}" >/dev/led_one
done
```



# Device Tree 101

Organized in partnership with ST  
February 9, 2021

Thomas Petazzoni

*thomas.petazzoni@bootlin.com*

© Copyright 2004-2021, Bootlin.

Creative Commons BY-SA 3.0 license.

Corrections, suggestions, contributions and translations are welcome!





# Who is speaking ?

- ▶ Thomas Petazzoni
- ▶ **Chief Technical Officer** at Bootlin
- ▶ Joined in 2008, employee #1
- ▶ **Embedded Linux & Linux kernel engineer**,  
open-source contributor
  - ▶ Author of the *Device Tree for Dummies* talk in  
2013/2014
  - ▶ Buildroot co-maintainer
  - ▶ Linux kernel contributor:  $\approx$  900 contributions
- ▶ Member of Embedded Linux Conference  
(Europe) program committee
- ▶ Based in Toulouse, France





# Agenda

- ▶ Bootlin introduction
- ▶ STM32MP1 introduction
- ▶ Why the Device Tree ?
- ▶ Basic Device Tree syntax
- ▶ Device Tree inheritance
- ▶ Device Tree specifications and bindings
- ▶ Device Tree and Linux kernel drivers
- ▶ Common properties and examples





- ▶ In business since 2004
- ▶ Team based in France
- ▶ Serving **customers worldwide**
  - ▶ 18% revenue from France
  - ▶ 44% revenue from EU except France
  - ▶ 38% revenue outside EU
- ▶ **Highly focused and recognized expertise**
  - ▶ Embedded Linux
  - ▶ Linux kernel
  - ▶ Embedded Linux build systems
- ▶ Activities
  - ▶ **Training** courses ( $\simeq 20\%$  revenue)
  - ▶ **Engineering** services ( $\simeq 80\%$  revenue)

bootlin





# Bootlin training courses

## Embedded Linux system development

On-site: 4 or 5 days  
Online: 7 \* 4 hours

## Linux kernel driver development

On-site: 5 days  
Online: 7 \* 4 hours

## Yocto Project system development

On-site: 3 days  
Online: 4 \* 4 hours

## Buildroot system development

On-site: 3 days  
Online: 4 \* 4 hours

## Understanding the Linux graphics stack

On-site: 2 days  
Online: 4 \* 4 hours

## Embedded Linux boot time optimization

On-site: 3 days  
Online: 4 \* 4 hours



# Why choose Bootlin training courses ?

- ▶ Complete training materials **freely available**
  - ▶ **Open-source license:** Creative Commons
  - ▶ Allows to **verify** in detail the course contents
  - ▶ Shows Bootlin commitment to **knowledge sharing**
  - ▶ **Unique** in the training industry





# Why choose Bootlin training courses ?

- ▶ Complete training materials **freely available**
  - ▶ **Open-source license:** Creative Commons
  - ▶ Allows to **verify** in detail the course contents
  - ▶ Shows Bootlin commitment to **knowledge sharing**
  - ▶ **Unique** in the training industry
- ▶ **Experienced** trainers
  - ▶ Bootlin trainers are also engineers
  - ▶ Working on **real engineering** projects
  - ▶ Up-to-date and in-field experience





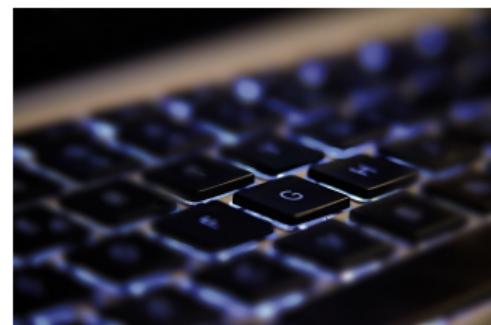
# Why choose Bootlin training courses ?

- ▶ Complete training materials **freely available**
  - ▶ **Open-source license:** Creative Commons
  - ▶ Allows to **verify** in detail the course contents
  - ▶ Shows Bootlin commitment to **knowledge sharing**
  - ▶ **Unique** in the training industry
- ▶ **Experienced** trainers
  - ▶ Bootlin trainers are also engineers
  - ▶ Working on **real engineering** projects
  - ▶ Up-to-date and in-field experience
- ▶ Worldwide **recognized** training courses
  - ▶ Taught 100s of sessions
  - ▶ To **1000s of engineers**
  - ▶ For the past 15 years





- ▶ Main activities
  - ▶ Linux **Board Support Package** development, update and maintenance
  - ▶ Linux **kernel drivers** development
  - ▶ Bootloader and Linux kernel **porting**
  - ▶ System integration: Yocto, Buildroot, boot time, secure boot, etc.
  - ▶ **Upstreaming**
  - ▶ **Consulting** and technical support
  - ▶ Focus on the low-level software stack
- ▶ Customers
  - ▶ **Silicon vendors**: interested in U-Boot, Linux, Buildroot or Yocto support for their product, usually upstream
  - ▶ **Embedded system manufacturers**: complete BSP, specific drivers, debugging, optimization, consulting





# Bootlin: open-source contributor

- ▶ Bootlin 20th contributing company worldwide to the **Linux kernel**
- ▶ **7600+ patches** contributed, mainly around hardware support
- ▶ Maintainers of several subsystems of the kernel: I3C, RTC, MTD, and several platforms
- ▶ Key contributor to Buildroot: co-maintainer, **5000+ patches** contributed
- ▶ Contributor to the **Yocto Project**
- ▶ Contributions to Barebox, Linux Test Project, etc.
- ▶ Freely available training materials
- ▶ Numerous **talks** at conferences to share technical knowledge

|       |                                      |                |
|-------|--------------------------------------|----------------|
| No.1  | Unknown                              | 140019(15.26%) |
| No.2  | Intel                                | 94806(10.33%)  |
| No.3  | Red Hat                              | 78140(8.52%)   |
| No.4  | Hobbyists                            | 73603(8.02%)   |
| No.5  | Novell                               | 39218(4.27%)   |
| No.6  | IBM                                  | 35085(3.82%)   |
| No.7  | Linaro                               | 28288(3.08%)   |
| No.8  | AMD                                  | 22426(2.44%)   |
| No.9  | Google                               | 20489(2.23%)   |
| No.10 | Renesas Electronics                  | 18443(2.01%)   |
| No.11 | Oracle                               | 17729(1.93%)   |
| No.12 | Samsung                              | 17514(1.91%)   |
| No.13 | Texas Instruments                    | 16372(1.78%)   |
| No.14 | HuaWei                               | 13377(1.46%)   |
| No.15 | Mellanox Technologies                | 11477(1.25%)   |
| No.16 | ARM                                  | 8919(0.97%)    |
| No.17 | Academics                            | 8560(0.93%)    |
| No.18 | Consultants                          | 8073(0.88%)    |
| No.19 | Broadcom                             | 8011(0.87%)    |
| No.20 | Bootlin                              | 7611(0.83%)    |
| No.21 | NXP                                  | 7549(0.82%)    |
| No.22 | Linutronix                           | 7430(0.81%)    |
| No.23 | NVIDIA                               | 6951(0.76%)    |
| No.24 | Canonical                            | 6855(0.75%)    |
| No.25 | Linux Foundation                     | 6369(0.69%)    |
| No.26 | Code Aurora Forum                    | 6280(0.68%)    |
| No.27 | Pengutronix                          | 6201(0.68%)    |
| No.28 | VISION Engraving and Routing Systems | 6045(0.66%)    |
| No.29 | Analog Devices                       | 5944(0.65%)    |
| No.30 | Fujitsu                              | 5120(0.56%)    |
| No.31 | QUALCOMM                             | 4903(0.53%)    |
| No.32 | Freescale                            | 4694(0.51%)    |
| No.33 | Wolfson Microelectronics             | 4180(0.46%)    |
| No.34 | Marvell                              | 4178(0.46%)    |
| No.35 | Nokia                                | 4097(0.45%)    |
| No.36 | Cisco                                | 4071(0.44%)    |
| No.37 | Parallels                            | 3841(0.42%)    |
| No.38 | Imagination Technologies             | 3774(0.41%)    |
| No.39 | Facebook                             | 3484(0.38%)    |
| No.40 | QLogic                               | 3394(0.37%)    |
| No.41 | ST Microelectronics                  | 3188(0.35%)    |
| No.42 | Astaro                               | 2981(0.32%)    |
| No.43 | NetApp                               | 2860(0.31%)    |



# STM32MP157F system-on-chip

| STM32MP157F                                       |  |                                       |  |  |  |
|---------------------------------------------------|--|---------------------------------------|--|--|--|
| <b>System</b>                                     |  | <b>Dual Cortex-A7 @ 800MHz</b>        |  |  |  |
| 5x LDOs                                           |  | Core 1 @ 800MHz<br>L1 32KB I / 32KB D |  |  |  |
| Crystal & Internal oscillators                    |  | Core 2 @ 800MHz<br>L1 32KB I / 32KB D |  |  |  |
| MDMA + 2x DMA                                     |  | NEON SIMD                             |  |  |  |
| Reset and Clock                                   |  | NEON SIMD                             |  |  |  |
| Watchdogs (2x I & W)                              |  | 256KB L2 cache                        |  |  |  |
| 96-bit unique ID                                  |  | <b>Cortex-M4 @ 209MHz</b>             |  |  |  |
| Up to 176 GPIOs                                   |  | FPU      MPU                          |  |  |  |
| <b>Security</b>                                   |  |                                       |  |  |  |
| TrustZone                                         |  |                                       |  |  |  |
| DES, TDES, AES-256                                |  |                                       |  |  |  |
| SHA-256, MD5, HMAC                                |  |                                       |  |  |  |
| 3x Tamper Pins with 1 active                      |  |                                       |  |  |  |
| T°, V and 32KHz detection                         |  |                                       |  |  |  |
| Secure ROM and RAMs                               |  |                                       |  |  |  |
| Secure Peripherals                                |  |                                       |  |  |  |
| Secure RTC                                        |  |                                       |  |  |  |
| Analog true RNG                                   |  |                                       |  |  |  |
| <b>DDR3/DDR3L 32-bit @ 533MHz</b>                 |  | <b>LPDDR2/LPDDR3 32-bit @ 533MHz</b>  |  |  |  |
| System RAM 256KB                                  |  | MCU System RAM 384KB                  |  |  |  |
| Retention RAM 64KB                                |  | Backup RAM 4KB                        |  |  |  |
| Boot ROM 128KB                                    |  | OTP Fuse 3Kb                          |  |  |  |
| <b>Control</b>                                    |  |                                       |  |  |  |
| 2x 16-bit motor control PWM synchronized AC timer |  |                                       |  |  |  |
| 10x 16-bit timer                                  |  | 5x 16-bit LP timer                    |  |  |  |
| 2x 32-bit timer                                   |  |                                       |  |  |  |
| <b>3D GPU OpenGL ES2.0 @ 533MHz</b>               |  |                                       |  |  |  |
| 26Mtri/sec, 133Mpix/sec                           |  |                                       |  |  |  |
| <b>Connectivity</b>                               |  |                                       |  |  |  |
| 24-bit Parallel RGB Display                       |  | MIPI DSI 2 lanes @ 1Gbps              |  |  |  |
| Camera Interface                                  |  | HDMI CEC                              |  |  |  |
| 1Gbps Ethernet                                    |  | 2x FDCAN / TTCAN                      |  |  |  |
| 2x USB2.0 Host HS                                 |  | USB2.0 OTG FS/HS                      |  |  |  |
| MDIO                                              |  | DFSDM 8 channels / 6 filters          |  |  |  |
| 6x PC                                             |  | 4x UART, 4x USART                     |  |  |  |
| 6x SPI / 3x PS                                    |  | 4x SAI                                |  |  |  |
| SPDIF Tx / Rx 4 inputs                            |  | Dual Quad SPI                         |  |  |  |
| 3x SDIO3.0 / SD3 / eMMC 4.51                      |  | 16-bit SLC NAND, 8-bit-ECC            |  |  |  |
| <b>Analog</b>                                     |  |                                       |  |  |  |
| 2x 16-bit ADC                                     |  |                                       |  |  |  |
| 2x 12-bit DAC                                     |  |                                       |  |  |  |
| Temperature sensor                                |  |                                       |  |  |  |



# STM32MP1 system-on-chip family

| STM32MP1 Series<br>Arm® Cortex®-A7 – up to 800 MHz |               |                 |                     |                 |                     |        |                     |           |        |                      |
|----------------------------------------------------|---------------|-----------------|---------------------|-----------------|---------------------|--------|---------------------|-----------|--------|----------------------|
| Acceleration                                       | Product lines | Cortex®-A7 core | $f_{Core}$<br>(MHz) | Cortex®-M4 core | $f_{Core}$<br>(MHz) | 3D GPU | $f_{Core}$<br>(MHz) | HW Crypto | FD-CAN | MIP®-DSI             |
|                                                    |               |                 |                     |                 |                     |        |                     |           |        | Junction temperature |
|                                                    | STM32MP151A   | 1               | 650                 | 1               | 209                 | -      | -                   | -         | -      | -40°C to 125°C       |
|                                                    | STM32MP151C   |                 |                     |                 |                     |        |                     | *         |        |                      |
|                                                    | STM32MP151D   | 1               | 800                 | 1               | 209                 | -      | -                   | -         |        | -20°C to 105°C       |
|                                                    | STM32MP151F   |                 |                     |                 |                     |        |                     | *         |        |                      |
|                                                    | STM32MP153A   | 2               | 650                 | 1               | 209                 | -      | -                   | -         | 2      | -40°C to 125°C       |
|                                                    | STM32MP153C   |                 |                     |                 |                     |        |                     | *         |        |                      |
|                                                    | STM32MP153D   | 2               | 800                 | 1               | 209                 | -      | -                   | -         | 2      | -20°C to 105°C       |
|                                                    | STM32MP153F   |                 |                     |                 |                     |        |                     | *         |        |                      |
|                                                    | STM32MP157A   | 2               | 650                 | 1               | 209                 | *      | 533                 | -         | 2      | -40°C to 125°C       |
|                                                    | STM32MP157C   |                 |                     |                 |                     |        |                     | *         |        |                      |
|                                                    | STM32MP157D   | 2               | 800                 | 1               | 209                 | *      | 533                 | -         | 2      | -20°C to 105°C       |
|                                                    | STM32MP157F   |                 |                     |                 |                     |        |                     | *         |        |                      |

Notes:

\* Not available in all product lines

\*\* 16/32-bit for LFBGA448 and TFBGA361 packages, 16-bit only for LFBGA354 and TFBGA257 packages

\*\*\* 10/100 Ethernet only for LFBGA354 and TFBGA257 packages



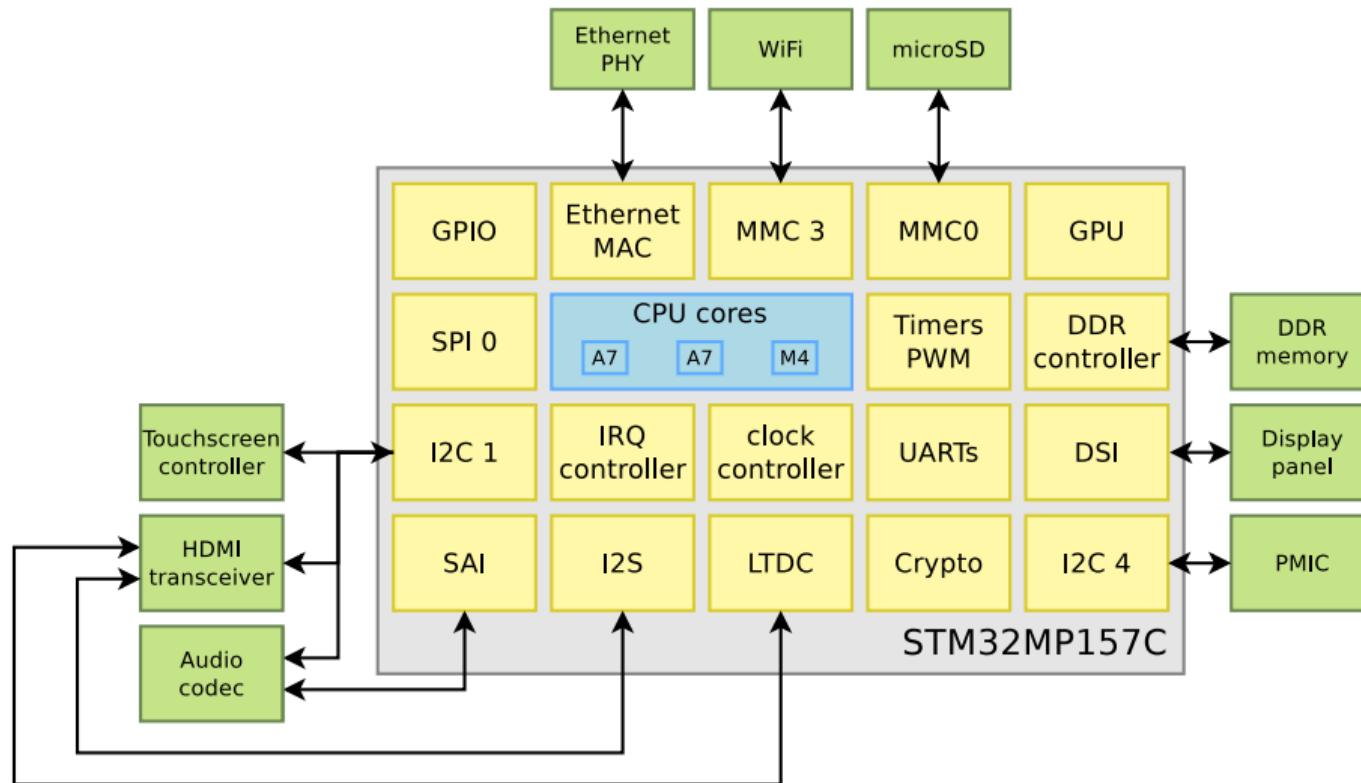
# STM32MP1 Discovery Kits



- ▶ Discovery Kit 1 (DK1)
  - ▶ SoC: STM32MP157**A**
  - ▶ 512 MB DDR, microSD
  - ▶ 1G Ethernet, 1x USB-C, 4x USB-A, LEDs, buttons
  - ▶ HDMI, audio codec, DSI connector
  - ▶ GPIO connectors, Arduino/RaspberryPi shields
  - ▶ On-board ST-Link
- ▶ Discovery Kit 2 (DK2)
  - ▶ SoC: STM32MP157**C**
  - ▶ Same as DK1
  - ▶ WiFi/Bluetooth
  - ▶ Display + touchscreen



# STM32MP1 DK2 partial block diagram





## Discoverable vs. non-discoverable hardware

- ▶ Some hardware busses provide **discoverability** mechanisms
  - ▶ E.g: PCI(e), USB
  - ▶ One does not need to know ahead of time what will be connected on these busses
  - ▶ Devices can be enumerated and identified at runtime
  - ▶ Concept of *vendor ID*, *product ID*, *device class*, etc.



# Discoverable vs. non-discoverable hardware

- ▶ Some hardware busses provide **discoverability** mechanisms
  - ▶ E.g: PCI(e), USB
  - ▶ One does not need to know ahead of time what will be connected on these busses
  - ▶ Devices can be enumerated and identified at runtime
  - ▶ Concept of *vendor ID*, *product ID*, *device class*, etc.
- ▶ But many hardware busses **do not provide discoverability** mechanisms
  - ▶ E.g: I2C, SPI, 1-wire, memory-mapped, etc.
  - ▶ One needs to know what is connected on those busses, and how they are connected to the rest of the system
  - ▶ Embedded systems typically make extensive use of such busses



# Hardware description for non-discoverable hardware

Allows the operating system or bootloader to **know things like**:

- ▶ This **system-on-chip** has:
  - ▶ 2 Cortex-A7 CPU cores
  - ▶ 2 memory-mapped UART controllers of *this* variant, one with registers at `0x5c000000` and IRQ 37, and another with registers at `0x4000e000` and IRQ 38
  - ▶ 3 I2C controllers of *that* variant, with registers at *those* memory-mapped addresses, *those* IRQs and taking their input clock from *this* source
- ▶ This **board** has a CS42L51 audio codec
  - ▶ Connected on the I2C bus 1 of the SoC, at slave address `0x4A`
  - ▶ Connected to the SAI interface 2 of the SoC
  - ▶ With its reset signal connected to GPIO 67 of the SoC
- ▶ ...

→ These details **cannot be guessed** by the operating system/bootloader.



# Describing non-discoverable hardware

1. Directly in the **OS/bootloader code**
  - ▶ Using compiled data structures, typically in C
  - ▶ How it was done on most embedded platforms in Linux, U-Boot.
  - ▶ Considered not maintainable/sustainable on ARM32, which motivated the move to another solution.



# Describing non-discoverable hardware

## 2. Using **ACPI** tables

- ▶ On *x86* systems, but also on a subset of ARM64 platforms
- ▶ Tables provided by the firmware



# Describing non-discoverable hardware

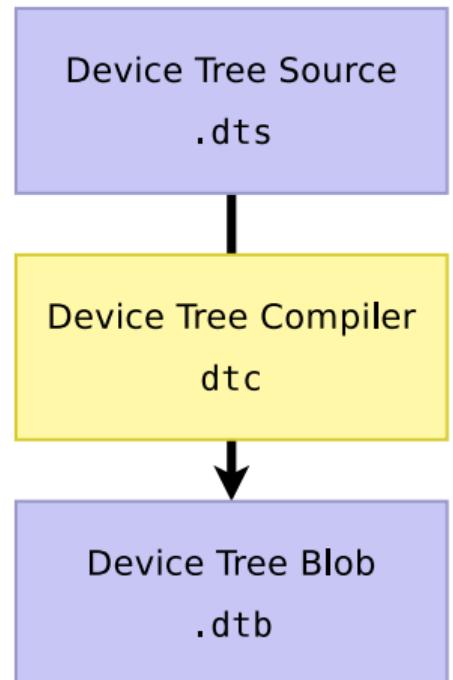
## 3. Using a **Device Tree**

- ▶ Originates from **OpenFirmware**, defined by Sun, used on SPARC and PowerPC
  - ▶ That's why many Linux/U-Boot functions related to DT have a `of_` prefix
- ▶ Now used most embedded-oriented CPU architectures that run Linux: ARC, ARM64, RISC-V, ARM32, PowerPC, Xtensa, MIPS, etc.
- ▶ Writing/tweaking a DT is now always necessary when porting Linux to a new board.
- ▶ The **topic of this talk !**



# Device Tree: from source to blob

- ▶ A tree data structure describing the hardware is written by a developer in a **Device Tree Source** file, .dts
- ▶ Processed by the **Device Tree Compiler**, dtc
- ▶ Produces a more efficient representation: **Device Tree Blob**, .dtb
- ▶ Additional C preprocessor pass
- ▶ .dtb → accurately describes the hardware platform in an **OS-agnostic** way.
- ▶ .dtb ≈ few dozens of kilobytes
- ▶ DTB also called **FDT**, *Flattened Device Tree*, once loaded into memory.
  - ▶ fdt command in U-Boot
  - ▶ fdt\_ APIs





## dtc example

```
$ cat foo.dts
/dts-v1/;

/ {
    welcome = <0xBADCAFE>;
    bootlin {
        webinar = "great";
        demo = <1>, <2>, <3>;
    };
};
```



## dtc example

```
$ cat foo.dts
/dts-v1/;

/ {
    welcome = <0xBADCAFE>;
    bootlin {
        webinar = "great";
        demo = <1>, <2>, <3>;
    };
};
```

```
$ dtc -I dts -O dtb -o foo.dtb foo.dts
$ ls -l foo.dt*
-rw-r--r-- 1 thomas thomas 169 ... foo.dtb
-rw-r--r-- 1 thomas thomas 102 ... foo.dts
```



# dtc example

```
$ cat foo.dts
/dts-v1/;

/ {
    welcome = <0xBADCAFE>;
    bootlin {
        webinar = "great";
        demo = <1>, <2>, <3>;
    };
};
```

```
$ dtc -I dts -O dtb -o foo.dtb foo.dts
$ ls -l foo.dt*
-rw-r--r-- 1 thomas thomas 169 ... foo.dtb
-rw-r--r-- 1 thomas thomas 102 ... foo.dts
```

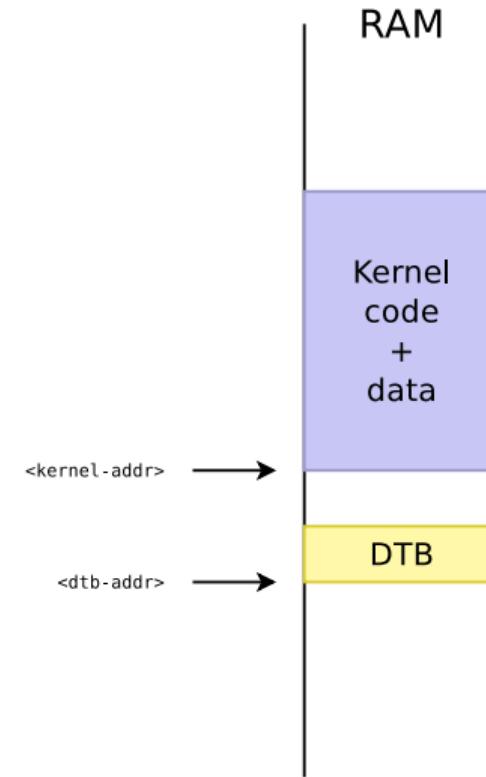
```
$ dtc -I dtb -O dts foo.dtb
/dts-v1/;

/ {
    welcome = <0xbadcafe>;
    bootlin {
        webinar = "great";
        demo = <0x01 0x02 0x03>;
    };
};
```



# Device Tree: using the blob

- ▶ Can be **linked directly** inside a bootloader binary
  - ▶ For example: U-Boot, Barebox
- ▶ Can be **passed** to the operating system by the bootloader
  - ▶ Most common mechanism for the Linux kernel
  - ▶ U-Boot:  
`bootz <kernel-addr> - <dtb-addr>`
  - ▶ The DTB address is passed through a dedicated CPU register to the kernel: `r2` on ARM32
  - ▶ Bootloader can adjust the DTB before passing it to the kernel
- ▶ The DTB parsing can be done using `libfdt`, or ad-hoc code





# Where are Device Tree Sources located ?

- ▶ Even though they are OS-agnostic, **no central and OS-neutral** place to host Device Tree sources and share them between projects
  - ▶ Often discussed, never done
- ▶ In practice, the Linux kernel sources can be considered as the **canonical location** for Device Tree Source files
  - ▶ arch/<ARCH>/boot/dts
  - ▶ ≈ 4700 Device Tree Source files in Linux as of 5.10
- ▶ Duplicated/synced in various projects
  - ▶ U-Boot, Barebox, TF-A



# Example with STM32MP157A-DK1

- ▶ 1st stage: **TF-A**
  - ▶ DT in `fdts/stm32mp157a-dk1.dts`
  - ▶ Build with `PLAT=stm32mp1 DTB_FILE_NAME=stm32mp157a-dk1.dtb`
  - ▶ Bundles the DTB in the resulting `tf-a-stm32mp157a-dk1.stm32`
- ▶ 2nd stage: **U-Boot**
  - ▶ DT in `arch/arm/dts/stm32mp157a-dk1.dts`
  - ▶ Configure with `stm32mp15_trusted_defconfig`
  - ▶ Build with `DEVICE_TREE=stm32mp157a-dk1`
  - ▶ Bundles the DTB in the resulting `u-boot.stm32`
- ▶ OS: **Linux kernel**
  - ▶ DT in `arch/arm/boot/dts/stm32mp157a-dk1.dts`
  - ▶ Configure with `multi_v7_defconfig`
  - ▶ Build
  - ▶ Kernel image: `arch/arm/boot/zImage`, DTB: `arch/arm/boot/dts/stm32mp157a-dk1.dtb`



# Booting Linux on STM32MP157A-DK1

U-Boot 2020.07 (Feb 04 2021 - 16:27:10 +0100)

CPU: STM32MP157AAC Rev.B

Model: STMicroelectronics STM32MP157A-DK1 Discovery Board



# Booting Linux on STM32MP157A-DK1

```
U-Boot 2020.07 (Feb 04 2021 - 16:27:10 +0100)
CPU: STM32MP157AAC Rev.B
Model: STMicroelectronics STM32MP157A-DK1 Discovery Board
```

```
STM32MP> printenv kernel_addr_r
kernel_addr_r=0xc2000000
STM32MP> printenv fdt_addr_r
fdt_addr_r=0xc4000000
```



# Booting Linux on STM32MP157A-DK1

```
U-Boot 2020.07 (Feb 04 2021 - 16:27:10 +0100)
CPU: STM32MP157AAC Rev.B
Model: STMicroelectronics STM32MP157A-DK1 Discovery Board
```

```
STM32MP> printenv kernel_addr_r
kernel_addr_r=0xc2000000
STM32MP> printenv fdt_addr_r
fdt_addr_r=0xc4000000
```

```
STM32MP> ext4load mmc 0:4 ${kernel_addr_r} /boot/zImage
4202992 bytes read in 207 ms (19.4 MiB/s)
STM32MP> ext4load mmc 0:4 ${fdt_addr_r} /boot/stm32mp157a-dk1.dtb
53881 bytes read in 31 ms (1.7 MiB/s)
```



# Booting Linux on STM32MP157A-DK1

```
U-Boot 2020.07 (Feb 04 2021 - 16:27:10 +0100)
CPU: STM32MP157AAC Rev.B
Model: STMicroelectronics STM32MP157A-DK1 Discovery Board
```

```
STM32MP> printenv kernel_addr_r
kernel_addr_r=0xc2000000
STM32MP> printenv fdt_addr_r
fdt_addr_r=0xc4000000
```

```
STM32MP> ext4load mmc 0:4 ${kernel_addr_r} /boot/zImage
4202992 bytes read in 207 ms (19.4 MiB/s)
STM32MP> ext4load mmc 0:4 ${fdt_addr_r} /boot/stm32mp157a-dk1.dtb
53881 bytes read in 31 ms (1.7 MiB/s)
```

```
STM32MP> bootz ${kernel_addr_r} - ${fdt_addr_r}
```



# Booting Linux on STM32MP157A-DK1

```
U-Boot 2020.07 (Feb 04 2021 - 16:27:10 +0100)
```

```
CPU: STM32MP157AAC Rev.B
```

```
Model: STMicroelectronics STM32MP157A-DK1 Discovery Board
```

```
STM32MP> printenv kernel_addr_r
```

```
kernel_addr_r=0xc2000000
```

```
STM32MP> printenv fdt_addr_r
```

```
fdt_addr_r=0xc4000000
```

```
STM32MP> ext4load mmc 0:4 ${kernel_addr_r} /boot/zImage
```

```
4202992 bytes read in 207 ms (19.4 MiB/s)
```

```
STM32MP> ext4load mmc 0:4 ${fdt_addr_r} /boot/stm32mp157a-dk1.dtb
```

```
53881 bytes read in 31 ms (1.7 MiB/s)
```

```
STM32MP> bootz ${kernel_addr_r} - ${fdt_addr_r}
```

```
Kernel image @ 0xc2000000 [ 0x0000000 - 0x4021f0 ]
```

```
## Flattened Device Tree blob at c4000000
```

```
Booting using the fdt blob at 0xc4000000
```

```
Loading Device Tree to cffef000, end cffff278 ... OK
```

```
Starting kernel ...
```

```
[    0.000000] Linux version 5.8.13 (thomas@windsurf) (arm-none-linux-gnueabihf-gcc....)
```

```
[    0.000000] CPU: ARMv7 Processor [410fc075] revision 5 (ARMv7), cr=10c5387d
```

```
...
```

```
[    0.000000] OF: fdt: Machine model: STMicroelectronics STM32MP157A-DK1 Discovery Board
```



# Exploring the DT on the target

- ▶ In `/sys/firmware/devicetree/base`, there is a directory/file representation of the Device Tree contents

```
# ls -l /sys/firmware/devicetree/base/
total 0
-r--r--r--  1 root      root      4 Jan  1 00:00 #address-cells
-r--r--r--  1 root      root      4 Jan  1 00:00 #size-cells
drwxr-xr-x  2 root      root      0 Jan  1 00:00 chosen
drwxr-xr-x  3 root      root      0 Jan  1 00:00 clocks
-r--r--r--  1 root      root     34 Jan  1 00:00 compatible
[...]
-r--r--r--  1 root      root      1 Jan  1 00:00 name
drwxr-xr-x 10 root      root      0 Jan  1 00:00 soc
```

- ▶ If `dtc` is available on the target, possible to "unpack" the Device Tree using:  
`dtc -I fs /sys/firmware/devicetree/base`



# Device Tree base syntax

- ▶ Tree of **nodes**
- ▶ Nodes with **properties**
- ▶ Node ≈ a device or IP block
- ▶ Properties ≈ device characteristics
- ▶ Notion of **cells** in property values
- ▶ Notion of **phandle** to point to other nodes
- ▶ dtc only does syntax checking, no semantic validation

The diagram shows a snippet of Device Tree (DT) source code with various parts annotated:

- Properties of node@0**: A red box highlights the properties of the root node.
- Node name**: Points to the identifier `node@0`.
- Unit address**: Points to the identifier `node@0`.
- Property name**: Points to the first property name in the list.
- Property value**: Points to the first property value in the list.
- Label**: Points to the label `node1:`.
- Bytestring**: Points to the property `a-byte-data-property` and its value `[0x01 0x23 0x34 0x56]`.
- A phandle (reference to another node)**: Points to the property `a-reference-to-something` and its value `<&node1>`.
- Four cells (32 bits values)**: Points to the property `a-cell-property` and its value `<1 2 3 4>`.

```
/ {
    node@0 {
        a-string-property = "A string";
        a-string-list-property = "first string", "second string";
        a-byte-data-property = [0x01 0x23 0x34 0x56];

        child-node@0 {
            first-child-property;
            second-child-property = <1>;
            a-reference-to-something = <&node1>;
        };

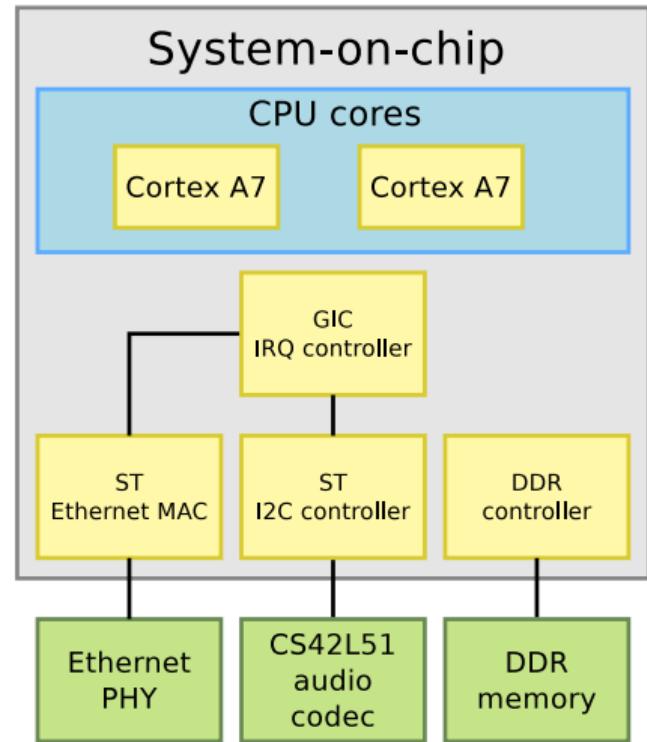
        child-node@1 {
        };
    };

    node1: node@1 {
        an-empty-property;
        a-cell-property = <1 2 3 4>;
        child-node@0 {
        };
    };
};
```



# DT overall structure: simplified example

```
/ {  
    #address-cells = <1>;  
    #size-cells = <1>;  
    model = "STMicroelectronics STM32MP157C-DK2 Discovery Board";  
    compatible = "st,stm32mp157c-dk2", "st,stm32mp157";  
  
    cpus { ... };  
    memory@0 { ... };  
    chosen { ... };  
    intc: interrupt-controller@a0021000 { ... };  
    soc {  
        i2c1: i2c@40012000 { ... };  
        ethernet0: ethernet@5800a000 { ... };  
    };  
};
```

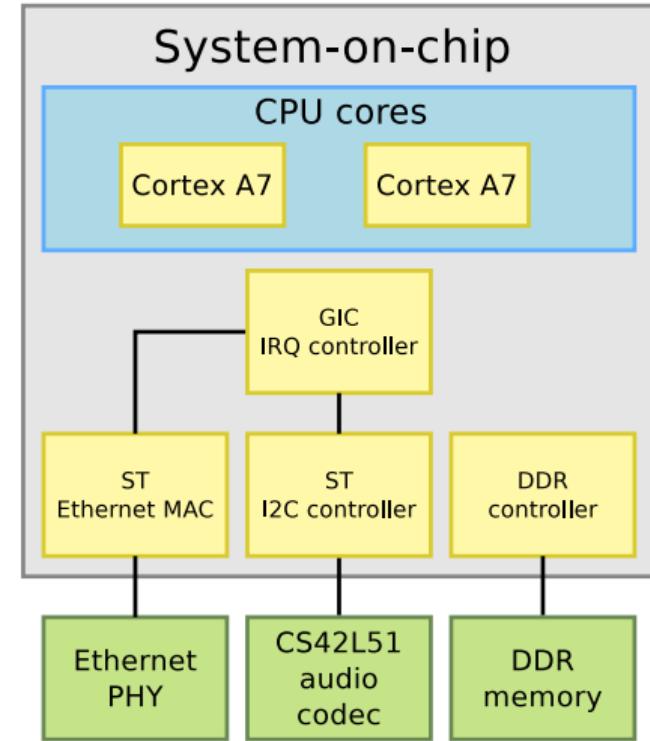




# DT overall structure: simplified example

```
/ {
    cpus {
        #address-cells = <1>;
        #size-cells = <0>;
        cpu0: cpu@0 {
            compatible = "arm,cortex-a7";
            clock-frequency = <650000000>;
            device_type = "cpu";
            reg = <0>;
        };
        cpu1: cpu@1 {
            compatible = "arm,cortex-a7";
            clock-frequency = <650000000>;
            device_type = "cpu";
            reg = <1>;
        };
    };

    memory@0 { ... };
    chosen { ... };
    intc: interrupt-controller@a0021000 { ... };
    soc {
        i2c1: i2c@40012000 { ... };
        ethernet0: ethernet@5800a000 { ... };
    };
};
```

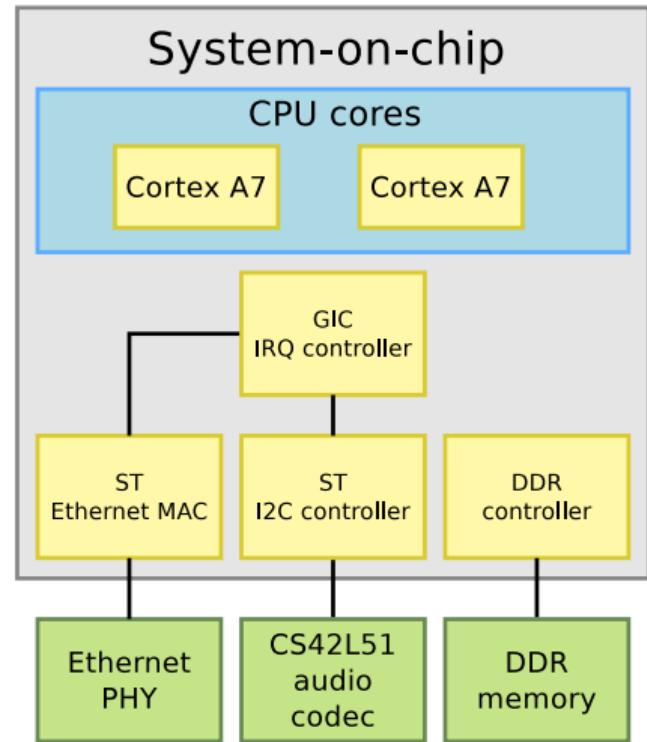




# DT overall structure: simplified example

```
/ {
    cpus { ... };
    memory@0 {
        device_type = "memory";
        reg = <0x0 0x20000000>;
    };

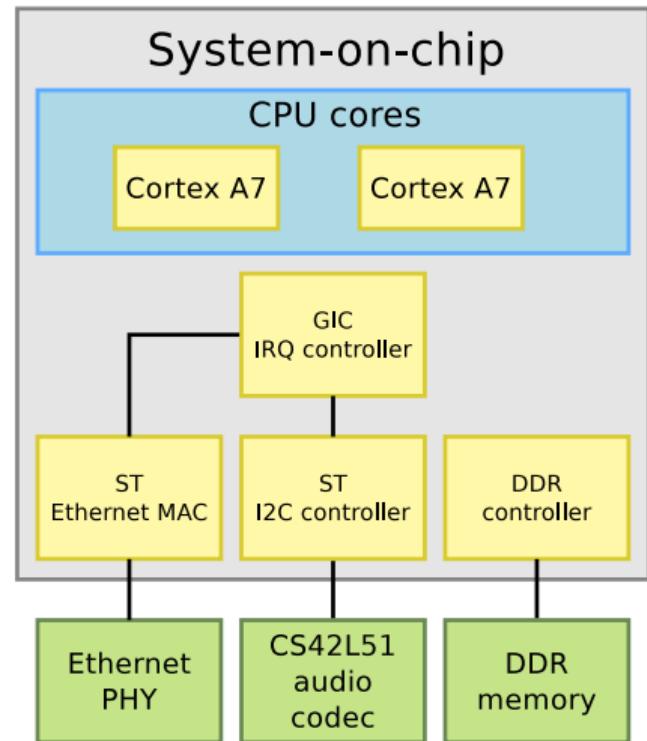
    chosen {
        bootargs = "";
        stdout-path = "serial0:115200n8";
    };
    intc: interrupt-controller@a0021000 { ... };
    soc {
        i2c1: i2c@40012000 { ... };
        ethernet0: ethernet@5800a000 { ... };
    };
};
```





# DT overall structure: simplified example

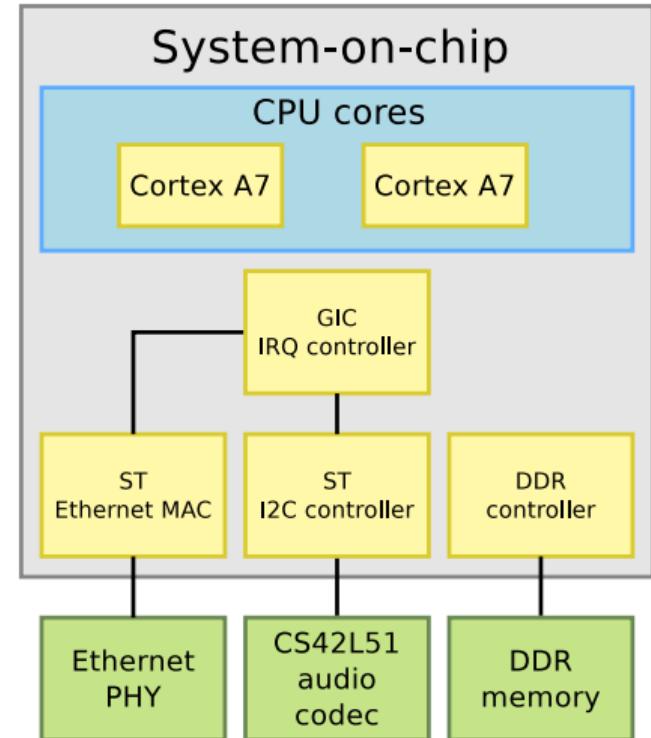
```
/ {  
    cpus { ... };  
    memory@0 { ... };  
    chosen { ... };  
  
    intc: interrupt-controller@a0021000 {  
        compatible = "arm,cortex-a7-gic";  
        #interrupt-cells = <3>;  
        interrupt-controller;  
        reg = <0xa0021000 0x1000>,  
              <0xa0022000 0x2000>;  
    };  
  
    soc {  
        compatible = "simple-bus";  
        #address-cells = <1>;  
        #size-cells = <1>;  
        interrupt-parent = <&intc>;  
  
        i2c1: i2c@40012000 { ... };  
        ethernet0: ethernet@5800a000 { ... };  
    };  
};
```





# DT overall structure: simplified example

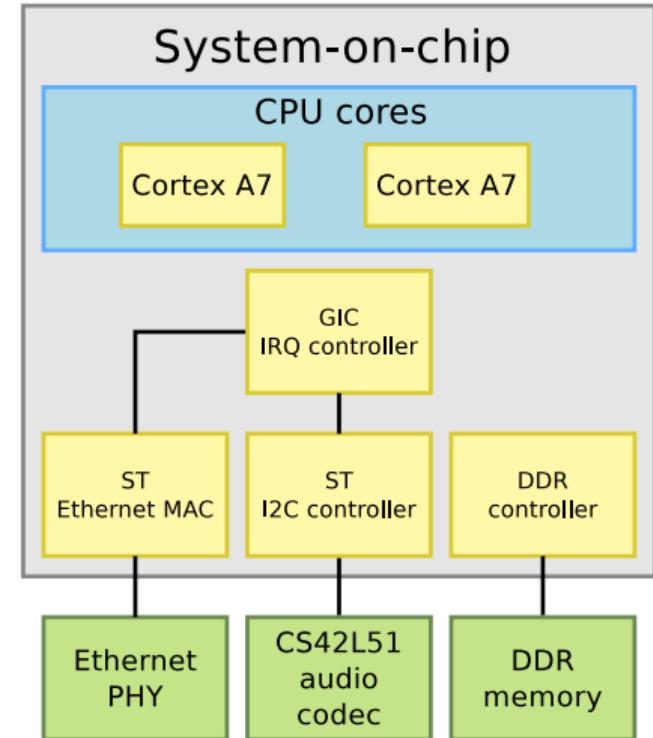
```
/ {  
    cpus { ... };  
    memory@0 { ... };  
    chosen { ... };  
    intc: interrupt-controller@a0021000 { ... };  
    soc {  
        i2c1: i2c@40012000 {  
            compatible = "st,stm32mp15-i2c";  
            reg = <0x40012000 0x400>;  
            interrupts = <GIC_SPI 31 IRQ_TYPE_LEVEL_HIGH>,  
                         <GIC_SPI 32 IRQ_TYPE_LEVEL_HIGH>;  
            #address-cells = <1>;  
            #size-cells = <0>;  
            status = "okay";  
  
            cs42l51: cs42l51@4a {  
                compatible = "cirrus,cs42l51";  
                reg = <0x4a>;  
                reset-gpios = <&gpio9 9 GPIO_ACTIVE_LOW>;  
                status = "okay";  
            };  
        };  
        ethernet0: ethernet@5800a000 { ... };  
    };  
};
```





# DT overall structure: simplified example

```
/ {  
    cpus { ... };  
    memory@0 { ... };  
    chosen { ... };  
    intc: interrupt-controller@a0021000 { ... };  
    soc {  
        compatible = "simple-bus";  
        ...  
        interrupt-parent = <&intc>;  
        i2c1: i2c@40012000 { ... };  
  
        ethernet0: ethernet@5800a000 {  
            compatible = "st,stm32mp1-dwmac", "snps,dwmac-4.20a";  
            reg = <0x5800a000 0x2000>;  
            interrupts-extended = <&intc GIC_SPI 61 IRQ_TYPE_LEVEL_HIGH>;  
            status = "okay";  
  
            mdio0 {  
                #address-cells = <1>;  
                #size-cells = <0>;  
                compatible = "snps,dwmac-mdio";  
                phy0: ethernet-phy@0 {  
                    reg = <0>;  
                };  
            };  
        };  
    };  
};
```





# Device Tree inheritance

- ▶ Device Tree files are not monolithic, they can be split in several files, including each other.
- ▶ `.dtsi` files are included files, while `.dts` files are *final* Device Trees
  - ▶ Only `.dts` files are accepted as input to `dtc`
- ▶ Typically, `.dtsi` will contain
  - ▶ definitions of SoC-level information
  - ▶ definitions common to several boards
- ▶ The `.dts` file contains the board-level information
- ▶ The inclusion works by **overlaid** the tree of the including file over the tree of the included file.
- ▶ Allows an including file to **override** values specified by an included file
- ▶ Uses the C pre-processor `#include` directive



# Device Tree inheritance example

Definition of the STM32MP157A SoC

```
/ {
    soc {
        i2c1: i2c@40012000 {
            compatible = "st,stm32mp15-i2c";
            reg = <0x40012000 0x400>;
            interrupts = <GIC_SPI 31 IRQ...HIGH>,
                         <GIC_SPI 32 IRQ...HIGH>;
            status = "disabled";
        };
    };
};
```



stm32mp157.dtsi

Definition of the STM32MP157A-DK1 board

```
#include "stm32mp157.dtsi"

/ {
    soc {
        i2c1: i2c@40012000 {
            pinctrl-names = "default", "sleep";
            pinctrl-0 = <&i2c1_pins_a>;
            pinctrl-1 = <&i2c1_sleep_pins_a>;
            status = "okay";
            cs42l51: cs42l51@4a {
                compatible = "cirrus,cs42l51";
                reg = <0x4a>;
            };
        };
    };
};
```

stm32mp157a-dk1.dts

Note 1

The actual Device Trees for this platform are more complicated. This example is highly simplified.

Compiled DTB

```
/ {
    soc {
        i2c1: i2c@40012000 {
            compatible = "st,stm32mp15-i2c";
            reg = <0x40012000 0x400>;
            interrupts = <GIC_SPI 31 IRQ...HIGH>,
                         <GIC_SPI 32 IRQ...HIGH>;
            pinctrl-names = "default", "sleep";
            pinctrl-0 = <&i2c1_pins_a>;
            pinctrl-1 = <&i2c1_sleep_pins_a>;
            status = "okay";
            cs42l51: cs42l51@4a {
                compatible = "cirrus,cs42l51";
                reg = <0x4a>;
            };
        };
    };
};
```

stm32mp157a-dk1.dtb

Note 2

The real DTB is in binary format. Here we show the text equivalent of the DTB contents.



# Inheritance and labels

Doing:

## soc.dtsi

```
/ {
    soc {
        usart1: serial@5c000000 {
            compatible = "st,stm32h7-uart";
            reg = <0x5c000000 0x400>;
            status = "disabled";
        };
    };
};
```

## board.dts

```
#include "soc.dtsi"

/ {
    soc {
        serial@5c000000 {
            status = "okay";
        };
    };
};
```



# Inheritance and labels

Doing:

`soc.dtsi`

```
/ {
  soc {
    usart1: serial@5c000000 {
      compatible = "st,stm32h7-uart";
      reg = <0x5c000000 0x400>;
      status = "disabled";
    };
  };
};
```

`board.dts`

```
#include "soc.dtsi"

/ {
  soc {
    serial@5c000000 {
      status = "okay";
    };
  };
};
```

Is exactly equivalent to:

`soc.dtsi`

```
/ {
  soc {
    usart1: serial@5c000000 {
      compatible = "st,stm32h7-uart";
      reg = <0x5c000000 0x400>;
      status = "disabled";
    };
  };
};
```

`board.dts`

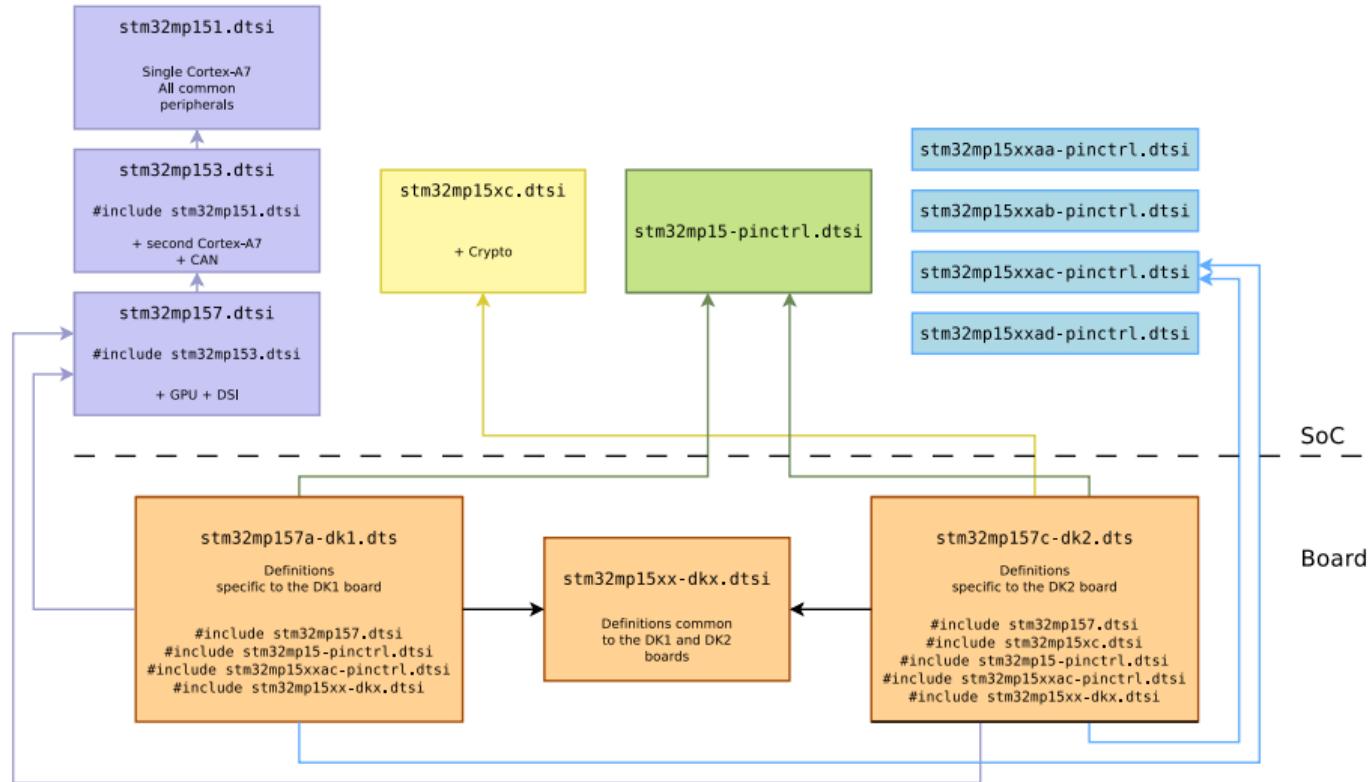
```
#include "soc.dtsi"

&usart1 {
  status = "okay";
};
```

→ this solution is now often preferred



# DT inheritance in STM32MP1 support





# Device Tree design principles

- ▶ **Describe hardware** (how the hardware is), not configuration (how I choose to use the hardware)
- ▶ **OS-agnostic**
  - ▶ For a given piece of HW, Device Tree should be the same for U-Boot, FreeBSD or Linux
  - ▶ There should be no need to change the Device Tree when updating the OS
- ▶ **Describe integration of hardware components**, not the internals of hardware components
  - ▶ The details of how a specific device/IP block is working is handled by code in device drivers
  - ▶ The Device Tree describes how the device/IP block is connected/integrated with the rest of the system: IRQ lines, DMA channels, clocks, reset lines, etc.
- ▶ Like all beautiful design principles, these principles are sometimes violated.



# Device Tree specifications

- ▶ How to write the correct nodes/properties to describe a given hardware platform ?
- ▶ **DeviceTree Specifications** → base Device Tree syntax + number of standard properties.
  - ▶ <https://www.devicetree.org/specifications/>
  - ▶ Not sufficient to describe the wide variety of hardware.
- ▶ **Device Tree Bindings** → documents that each specify how a piece of HW should be described
  - ▶ Documentation/devicetree/bindings/ in Linux kernel sources
  - ▶ Reviewed by DT bindings maintainer team
  - ▶ Legacy: human readable documents
  - ▶ New norm: YAML-written specifications



Devicetree Specification  
Release v0.3

devicetree.org

13 February 2020



# Device Tree binding: old style

Documentation/devicetree/bindings/mtd/spear\_smi.txt  
This IP is *not* used on STM32MP1.

## \* SPEAr SMI

### Required properties:

- compatible : "st,spear600-smi"
- reg : Address range of the mtd chip
- #address-cells, #size-cells : Must be present if the device has sub-nodes representing partitions.
- interrupts: Should contain the STMMAC interrupts
- clock-rate : Functional clock rate of SMI in Hz

### Optional properties:

- st,smi-fast-mode : Flash supports read in fast mode

### Example:

```
smi: flash@fc000000 {
    compatible = "st,spear600-smi";
    #address-cells = <1>;
    #size-cells = <1>;
    reg = <0xfc000000 0x1000>;
    interrupt-parent = <&vic1>;
    interrupts = <12>;
    clock-rate = <50000000>;           /* 50MHz */
}

flash@f8000000 {
    st,smi-fast-mode;
    ...
};

};
```



# Device Tree binding: YAML style

Documentation/devicetree/bindings/i2c/st,stm32-i2c.yaml

```
# SPDX-License-Identifier: (GPL-2.0-only OR BSD-2-Clause)
%YAML 1.2
---
$id: http://devicetree.org/schemas/i2c/st,stm32-i2c.yaml#
$schema: http://devicetree.org/meta-schemas/core.yaml#
title: I2C controller embedded in STMicroelectronics STM32 I2C platform

maintainers:
  - Pierre-Yves MORDRET <pierre-yves.mordret@st.com>

properties:
  compatible:
    enum:
      - st,stm32f4-i2c
      - st,stm32f7-i2c
      - st,stm32mp15-i2c

    reg:
      maxItems: 1

    interrupts:
      items:
        - description: interrupt ID for I2C event
        - description: interrupt ID for I2C error

    resets:
      maxItems: 1
```

```
clocks:
  maxItems: 1

dmas:
  items:
    - description: RX DMA Channel phandle
    - description: TX DMA Channel phandle

  ...

clock-frequency:
  description: Desired I2C bus clock frequency in Hz. If not specified,
    the default 100 kHz frequency will be used.
  For STM32F7, STM32H7 and STM32MP1 SoCs, if timing
  parameters match, the bus clock frequency can be from
  1Hz to 1MHz.
  default: 100000
  minimum: 1
  maximum: 1000000

required:
  - compatible
  - reg
  - interrupts
  - resets
  - clocks
```



# Device Tree binding: YAML style example

```
examples:
- |
  //Example 3 (with st,stm32mp15-i2c compatible on stm32mp)
  #include <dt-bindings/interrupt-controller/arm-gic.h>
  #include <dt-bindings/clock/stm32mp1-clks.h>
  #include <dt-bindings/reset/stm32mp1-resets.h>
  i2c@40013000 {
    compatible = "st,stm32mp15-i2c";
    #address-cells = <1>;
    #size-cells = <0>;
    reg = <0x40013000 0x400>;
    interrupts = <GIC_SPI 33 IRQ_TYPE_LEVEL_HIGH>,
                 <GIC_SPI 34 IRQ_TYPE_LEVEL_HIGH>;
    clocks = <&rcc I2C2_K>;
    resets = <&rcc I2C2_R>;
    i2c-scl-rising-time-ns = <185>;
    i2c-scl-falling-time-ns = <20>;
    st,syscfg-fmp = <&syscfg 0x4 0x2>;
  };
};
```



# Validating Device Tree in Linux

- ▶ `dtc` only does syntaxic validation
- ▶ YAML bindings allow to do semantic validation
- ▶ Linux kernel `make` rules:
  - ▶ `make dt_binding_check`  
verify that YAML bindings are valid
  - ▶ `make dtbs_check`  
validate DTs currently enabled against YAML bindings
  - ▶ `make DT_SCHEMA_FILES=Documentation/devicetree/bindings/trivial-devices.yaml dtbs_check`  
validate DTs against a specific YAML binding



# The compatible property

- ▶ Is a list of strings
  - ▶ From the most specific to the less specific
- ▶ Describes the specific **binding** to which the node complies.
- ▶ It uniquely identifies the **programming model** of the device.
- ▶ Practically speaking, it is used by the operating system to find the **appropriate driver** for this device.
- ▶ When describing real hardware, typical form is `vendor, model`
- ▶ Examples:
  - ▶ `compatible = "arm,armv7-timer";`
  - ▶ `compatible = "st,stm32mp1-dwmac", "snps,dwmac-4.20a";`
  - ▶ `compatible = "regulator-fixed";`
  - ▶ `compatible = "gpio-keys";`
- ▶ Special value: `simple-bus` → bus where all sub-nodes are memory-mapped devices



# compatible property and Linux kernel drivers

- ▶ Top-level DT nodes with a `compatible` property and nodes that are sub-nodes of `simple-bus` will cause Linux to identify those devices as **platform devices**
  - ▶ Instantiated automatically at boot time
- ▶ Sub-nodes of I2C controllers → *I2C devices*
- ▶ Sub-nodes of SPI controllers → *SPI devices*
- ▶ Each Linux driver has a table of compatible strings it supports
  - ▶ `struct of_device_id []`
- ▶ When a DT node compatible string matches a given driver, the device is *bound* to that driver.

```
/ {
    timer { ───────────────────→ Platform device
            compatible = "...";
    };
    soc {
        compatible = "simple-bus";
        uart@1000 { ─────────────────→ Platform device
                     compatible = "...";
        };
        i2c@2000 { ─────────────────→ Platform device
                     compatible = "...";
                     eeprom@65 { ─────────────────→ I2C device
                                  compatible = "...";
                     };
        };
    };
};
```



# Matching with drivers in Linux: platform driver

## drivers/tty/serial/stm32-usart.c

```
static const struct of_device_id stm32_match[] = {
    { .compatible = "st,stm32-uart", .data = &stm32f4_info},
    { .compatible = "st,stm32f7-uart", .data = &stm32f7_info},
    { .compatible = "st,stm32h7-uart", .data = &stm32h7_info},
    {},
};

MODULE_DEVICE_TABLE(of, stm32_match);

static struct platform_driver stm32_serial_driver = {
    .probe          = stm32_serial_probe,
    .remove         = stm32_serial_remove,
    .driver = {
        .name      = DRIVER_NAME,
        .pm        = &stm32_serial_pm_ops,
        .of_match_table = of_match_ptr(stm32_match),
    },
};
```



# Matching with drivers in Linux: I2C driver

## sound/soc/codecs/cs42l51-i2c.c

```
const struct of_device_id cs42l51_of_match[] = {
    { .compatible = "cirrus,cs42l51", },
    { }
};

MODULE_DEVICE_TABLE(of, cs42l51_of_match);

static struct i2c_driver cs42l51_i2c_driver = {
    .driver = {
        .name = "cs42l51",
        .of_match_table = cs42l51_of_match,
        .pm = &cs42l51_pm_ops,
    },
    .probe = cs42l51_i2c_probe,
    .remove = cs42l51_i2c_remove,
    .id_table = cs42l51_i2c_id,
};

};
```



## reg property

- ▶ Most important property after `compatible`
- ▶ **Memory-mapped** devices: base physical address and size of the memory-mapped registers. Can have several entries for multiple register areas.

```
sai4: sai@50027000 {  
    reg = <0x50027000 0x4>, <0x500273f0 0x10>;  
};
```



## reg property

- ▶ Most important property after `compatible`
- ▶ **Memory-mapped** devices: base physical address and size of the memory-mapped registers. Can have several entries for multiple register areas.
- ▶ **I2C** devices: address of the device on the I2C bus.

```
&i2c1 {  
    hdmi-transmitter@39 {  
        reg = <0x39>;  
    };  
    cs42151: cs42151@4a {  
        reg = <0x4a>;  
    };  
};
```



## reg property

- ▶ Most important property after `compatible`
- ▶ **Memory-mapped** devices: base physical address and size of the memory-mapped registers. Can have several entries for multiple register areas.
- ▶ **I2C** devices: address of the device on the I2C bus.
- ▶ **SPI** devices: chip select number

```
&qspi {  
    flash0: mx66151235l@0 {  
        reg = <0>;  
    };  
    flash1: mx66151235l@1 {  
        reg = <1>;  
    };  
};
```



## reg property

- ▶ Most important property after `compatible`
- ▶ **Memory-mapped** devices: base physical address and size of the memory-mapped registers. Can have several entries for multiple register areas.
- ▶ **I2C** devices: address of the device on the I2C bus.
- ▶ **SPI** devices: chip select number
- ▶ The unit address must be the address of the first `reg` entry.

```
sai4: sai@50027000 {  
    reg = <0x50027000 0x4>, <0x500273f0 0x10>;  
};
```



# Cells concept

- ▶ Integer values represented as 32-bit integers called cells

```
soc {  
    /* This property has 1 cell */  
    foo = <0xdeadbeef>;  
};
```



# Cells concept

- ▶ Integer values represented as 32-bit integers called cells
- ▶ Encoding a 64-bit value requires two cells

```
soc {  
    /* This property has 2 cells */  
    foo = <0xdeadbeef 0xbadcafe>;  
};
```



# Cells concept

- ▶ Integer values represented as 32-bit integers called cells
- ▶ Encoding a 64-bit value requires two cells
- ▶ `#address-cells` and `#size-cells`: how many cells are used in sub-nodes to encode the address and size in the `reg` property

```
soc {  
    compatible = "simple-bus";  
    #address-cells = <1>;  
    #size-cells = <1>;  
  
    i2c@f1001000 {  
        reg = <0xf1001000 0x1000>;  
        #address-cells = <1>;  
        #size-cells = <0>;  
  
        eeprom@52 {  
            reg = <0x52>;  
        };  
    };  
};
```



# Cells concept

- ▶ Integer values represented as 32-bit integers called cells
- ▶ Encoding a 64-bit value requires two cells
- ▶ `#address-cells` and `#size-cells`: how many cells are used in sub-nodes to encode the address and size in the `reg` property
- ▶ `#interrupts-cells`: how many cells are used to encode interrupt specifiers for this interrupt controller

```
soc {  
    intc: interrupt-controller@f1002000 {  
        compatible = "foo,bar-intc";  
        reg = <0xf1002000 0x1000>;  
        interrupt-controller;  
        #interrupt-cells = <2>;  
    };  
  
    i2c@f1001000 {  
        interrupt-parent = <&intc>;  
        /* Must have two cells */  
        interrupts = <12 24>;  
    };  
};
```



# Cells concept

- ▶ Integer values represented as 32-bit integers called cells
- ▶ Encoding a 64-bit value requires two cells
- ▶ `#address-cells` and `#size-cells`: how many cells are used in sub-nodes to encode the address and size in the `reg` property
- ▶ `#interrupts-cells`: how many cells are used to encode interrupt specifiers for this interrupt controller
- ▶ Ditto `#clock-cells`, `#gpio-cells`, `#phy-cells`, `#pwm-cells`, `#dma-cells`, etc.

```
soc {  
    clkc: clock@f1003000 {  
        compatible = "foo,bar-clock";  
        reg = <0xf1003000 0x1000>;  
        #clock-cells = <3>;  
    };  
  
    i2c@f1001000 {  
        /* Must have three cells */  
        clocks = <&clkc 12 24 32>;  
    };  
};
```



## Status property

- ▶ status property indicates if the device is really in use or not
  - ▶ okay or ok → the device is really in use
  - ▶ any other value, by convention disabled → the device is not in used
- ▶ In Linux, controls if a device is instantiated
- ▶ In .dtsi files describing SoCs: all devices that interface to the outside world have  
`status = disabled`
- ▶ Enabled on a per-device basis in the board .dts



# Interrupt description

- ▶ Nodes describing **interrupt controllers**
  - ▶ Devices like any other
  - ▶ `interrupt-controller` boolean
- ▶ Devices using interrupts:
  - ▶ `interrupts + interrupt-parent`,  
`interrupts` specifies the interrupt number and flags,  
`interrupt-parent` the interrupt controller, searched recursively in the parent nodes
  - ▶ `interrupts-extended` specifies the interrupt controller, interrupt number and flags

```
/ {  
    intc: interrupt-controller@a0021000 {  
        compatible = "arm,cortex-a7-gic";  
        #interrupt-cells = <3>;  
        interrupt-controller;  
        reg = <0xa0021000 0x1000>,  
              <0xa0022000 0x2000>;  
    };  
  
    soc {  
        interrupt-parent = <&intc>;  
        spi2: spi@4000b000 {  
            interrupts = <GIC_SPI 36 IRQ_TYPE_LEVEL_HIGH>;  
            ...  
        };  
  
        ipcc: mailbox@4c001000 {  
            interrupts-extended =  
                <&intc GIC_SPI 100 IRQ_TYPE_LEVEL_HIGH>,  
                <&intc GIC_SPI 101 IRQ_TYPE_LEVEL_HIGH>,  
                <&exti 61 1>;  
            ...  
        };  
    };  
};
```



## Other resources: clocks, DMA, reset lines, ...

- ▶ Similar pattern for other resources shared by multiple hardware blocks
  - ▶ Clock controllers
  - ▶ DMA controllers
  - ▶ Reset controllers
  - ▶ ...
- ▶ A Device Tree node describing the *controller* as a device
- ▶ References from other nodes that use resources provided by this *controller*

```
rcc: rcc@50000000 {
    compatible = "st,stm32mp1-rcc", "syscon";
    reg = <0x50000000 0x1000>;
    #clock-cells = <1>;
    #reset-cells = <1>;
};

dmamux1: dma-router@48002000 {
    compatible = "st,stm32h7-dmamux";
    reg = <0x48002000 0x1c>;
    #dma-cells = <3>;
    dma-requests = <128>;
    dma-masters = <&dma1 &dma2>;
    dma-channels = <16>;
    clocks = <&rcc DMAMUX>;
    resets = <&rcc DMAMUX_R>;
};

spi3: spi@4000c000 {
    ...
    clocks = <&rcc SPI3_K>;
    resets = <&rcc SPI3_R>;
    dmas = <&dmamux1 61 0x400 0x05>,
           <&dmamux1 62 0x400 0x05>;
};
```



## -names properties

- ▶ Some properties are associated to a corresponding `<prop>-names` property
- ▶ Gives some human-readable names to entries of the corresponding `<prop>` properties

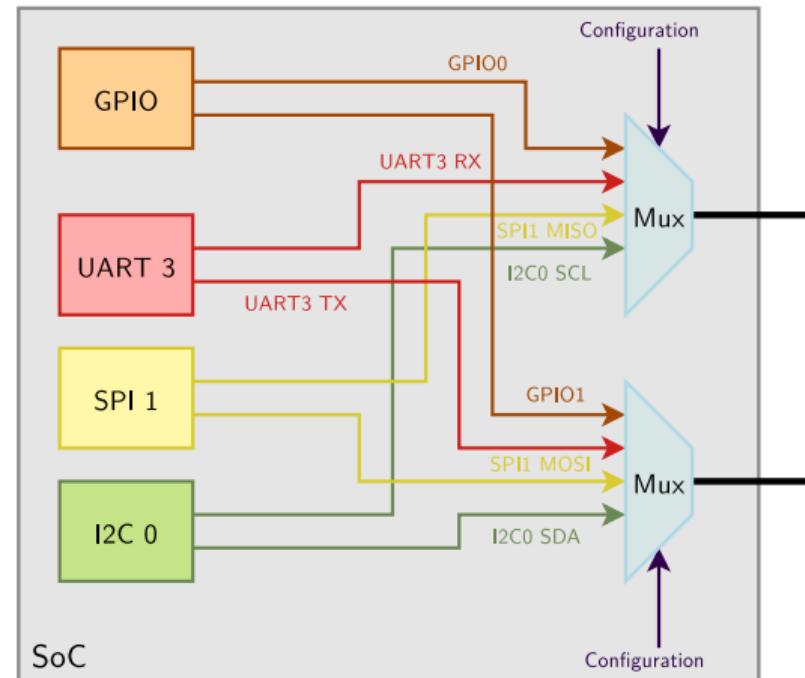
```
interrupts      = <0 59 0>, <0 70 0>;
interrupt-names = "macirq", "macpmt";
clocks          = <&car 39>, <&car 45>, <&car 86>, <&car 87>;
clock-names     = "gnssm_rgmii", "gnssm_gmac", "rgmii", "gmac";
```

- ▶ Such names can be typically be used by the driver
  - ▶ `platform_get_irq_byname(pdev, "macirq");`



# Pin-muxing description

- ▶ Most modern SoCs, including the STM32MP1, have more features than they have pins to expose those features to the outside world.
- ▶ Pins are muxed: a given pin can be used for one function **or** another
- ▶ A specific IP block in the SoC controls the muxing of pins: the **pinmux controller**
- ▶ The Device Tree describes which pin configurations are possible, and which configurations are used by the different devices.





# Pin-muxing controllers on STM32MP1

[arch/arm/boot/dts/stm32mp151.dtsi](#)

```
pinctrl: pin-controller@50002000 {
    #address-cells = <1>;
    #size-cells = <1>;
    compatible = "st,stm32mp157-pinctrl";

    gpioa: gpio@50002000 { ... };
    gpiob: gpio@50003000 { ... };
    gpioc: gpio@50004000 { ... };
    gpiod: gpio@50005000 { ... };
    gpioe: gpio@50006000 { ... };
    gpiof: gpio@50007000 { ... };
    ...
};

pinctrl_z: pin-controller-z@54004000 {
    #address-cells = <1>;
    #size-cells = <1>;
    compatible = "st,stm32mp157-z-pinctrl";
    ranges = <0 0x54004000 0x400>

    gpioz: gpio@54004000 { .... };
};
```



# Pin-muxing configuration

arch/arm/boot/dts/stm32mp151.dtsi

```
&pinctrl {
    i2c1_pins_a: i2c1-0 {
        pins {
            pinmux = <STM32_PINMUX('D', 12, AF5)>, /* I2C1_SCL */
                    <STM32_PINMUX('F', 15, AF5)>; /* I2C1_SDA */
            bias-disable;
            drive-open-drain;
            slew-rate = <0>;
        };
    };

    m_can1_pins_a: m-can1-0 {
        pins1 {
            pinmux = <STM32_PINMUX('H', 13, AF9)>; /* CAN1_TX */
            slew-rate = <1>;
            drive-push-pull;
            bias-disable;
        };
        pins2 {
            pinmux = <STM32_PINMUX('I', 9, AF9)>; /* CAN1_RX */
            bias-disable;
        };
    };
};
```



# Pin-muxing configuration

| Port   |             | AF0                                 | AF1                                     | AF2                                                       | AF3                                                              | AF4                                                                          | AF5                                              | AF6                                                                    | AF7                       |
|--------|-------------|-------------------------------------|-----------------------------------------|-----------------------------------------------------------|------------------------------------------------------------------|------------------------------------------------------------------------------|--------------------------------------------------|------------------------------------------------------------------------|---------------------------|
| Port D | HDP/SYS/RTC | TIM1/2/16/17/<br>LPTIM1/SYS/<br>RTC | SAI1/4/I2C6/<br>TIM3/4/5/12/<br>HDP/SYS | SAI4/I2C2/<br>TIM8/<br>LPTIM2/3/4/5/<br>DFSDM1<br>/SDMMC1 | SAI4/<br>I2C1/2/3/4/5/<br>USART1/<br>TIM15/LPTIM2/<br>DFSDM1/CEC | SPI1/I2S1/<br>SPI2/I2S2/<br>SPI3/I2S3/<br>SPI4/5/6/I2C1/<br>SDMMC1/3/<br>CEC | SPI3/I2S3/<br>SAI1/3/4/<br>I2C4/UART4/<br>DFSDM1 | SPI2/I2S2/<br>SPI3/I2S3/<br>SPI6/<br>USART1/2/3/6/<br>UART7/<br>SDMMC2 |                           |
|        | PD6         | -                                   | TIM16_CH1N                              | SAI1_D1                                                   | DFSDM1_<br>CKIN4                                                 | DFSDM1_<br>DATIN1                                                            | SPI3_MOSI/<br>I2S3_SDO                           | SAI1_SD_A                                                              | USART2_RX                 |
|        | PD7         | TRACED6                             | -                                       | -                                                         | DFSDM1_<br>DATIN4                                                | I2C2_SCL                                                                     | -                                                | DFSDM1_<br>CKIN1                                                       | USART2_CK                 |
|        | PD8         | -                                   | -                                       | -                                                         | DFSDM1_<br>CKIN3                                                 | -                                                                            | -                                                | SAI3_SCK_B                                                             | USART3_TX                 |
|        | PD9         | -                                   | -                                       | -                                                         | DFSDM1_<br>DATIN3                                                | -                                                                            | -                                                | SAI3_SD_B                                                              | USART3_RX                 |
|        | PD10        | RTC_REFIN                           | TIM16_BKIN                              | -                                                         | DFSDM1_<br>CKOUT                                                 | I2C5_SMBA                                                                    | SPI3_MISO/<br>I2S3_SDI                           | SAI3_FS_B                                                              | USART3_CK                 |
|        | PD11        | -                                   | -                                       | -                                                         | LPTIM2_IN2                                                       | I2C4_SMBA                                                                    | I2C1_SMBA                                        | -                                                                      | USART3_CTS/<br>USART3_NSS |
|        | PD12        | -                                   | LPTIM1_IN1                              | TIM4_CH1                                                  | LPTIM2_IN1                                                       | I2C4_SCL                                                                     | I2C1_SCL                                         | -                                                                      | USART3_RTS/<br>USART3_DE  |
|        | PD13        | -                                   | LPTIM1_OUT                              | TIM4_CH2                                                  | -                                                                | I2C4_SDA                                                                     | I2C1_SDA                                         | I2S3_MCK                                                               | -                         |
|        | PD14        | -                                   | -                                       | TIM4_CH3                                                  | -                                                                | -                                                                            | -                                                | SAI3_MCLK_B                                                            | -                         |
|        | PD15        | -                                   | -                                       | TIM4_CH4                                                  | -                                                                | -                                                                            | -                                                | SAI3_MCLK_A                                                            | -                         |



# Pin-muxing consumer

```
&i2c1 {  
    pinctrl-names = "default", "sleep";  
    pinctrl-0 = <&i2c1_pins_a>;  
    pinctrl-1 = <&i2c1_sleep_pins_a>;  
    ...  
};
```

- ▶ Typically board-specific, in .dts
- ▶ pinctrl-0, pinctrl-1, pinctrl-X provides the pin mux configurations for the different **states**
- ▶ pinctrl-names gives a name to each state, mandatory even if only one state
- ▶ States are mutually exclusive
- ▶ Driver is responsible for switching between states
- ▶ default state is automatically set up when the device is *probed*



## Example: LED and I2C device

- ▶ Let's see how to describe an LED and an I2C device connected to the DK1 platform.
- ▶ Create `arch/arm/boot/dts/stm32mp157a-dk1-custom.dts` which includes `stm32mp157a-dk1.dts`

```
#include "stm32mp157a-dk1.dts"
```

- ▶ Make sure `stm32mp157a-dk1-custom.dts` gets compiled to a DTB by changing `arch/arm/boot/dts/Makefile`

```
dtb-$(CONFIG_ARCH_STM32) += \
    ...
    stm32mp157a-dk1.dtb \
    stm32mp157a-dk1-custom.dtb
```

- ▶ `make dtbs`

DTC      `arch/arm/boot/dts/stm32mp157a-dk1-custom.dtb`



# Example: describe an LED

## stm32mp157a-dk1-custom.dts

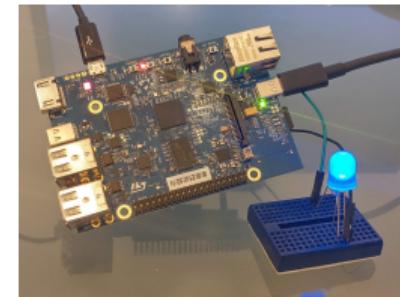
```
#include "stm32mp157a-dk1.dts"

/ {
    leds {
        compatible = "gpio-leds";
        webinar {
            label = "webinar";
            gpios = <&gpioe 1 GPIO_ACTIVE_HIGH>;
        };
    };
};
```

## shell

```
# echo 255 > /sys/class/leds/webinar/brightness
```

|      |   |        |      |           |
|------|---|--------|------|-----------|
| CN14 | 1 | ARD_D0 | PE7  | USART7_RX |
|      | 2 | ARD_D1 | PE8  | USART7_TX |
|      | 3 | ARD_D2 | PE1  | IO        |
|      | 4 | ARD_D3 | PD14 | TIM4_CH3  |
|      | 5 | ARD_D4 | PE10 | IO        |
|      | 6 | ARD_D5 | PD15 | TIM4_CH4  |
|      | 7 | ARD_D6 | PE9  | TIM1_CH1  |
|      | 8 | ARD_D7 | PD1  | IO        |





# Example: connect I2C sensor

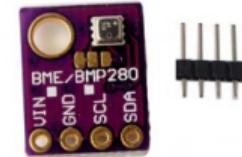
## stm32mp157a-dk1-custom.dts

```
&i2c5 {  
    status = "okay";  
    clock-frequency = <100000>;  
    pinctrl-names = "default", "sleep";  
    pinctrl-0 = <&i2c5_pins_a>;  
    pinctrl-1 = <&i2c5_pins_sleep_a>;  
  
    pressure@76 {  
        compatible = "bosch,bme280";  
        reg = <0x76>;  
    };  
};
```

## shell

```
# cat /sys/bus/iio/devices/iio:device2/in_temp_input  
24380
```

|      | 1  | ARD_D8  | PG3  | IO                     |
|------|----|---------|------|------------------------|
| CN13 | 2  | ARD_D9  | PH6  | TIM12_CH1              |
|      | 3  | ARD_D10 | PE11 | SPI4_NSS and TIM1_CH2  |
|      | 4  | ARD_D11 | PE14 | SPI4_MOSI and TIM1_CH4 |
|      | 5  | ARD_D12 | PE13 | SPI4_MISO              |
|      | 6  | ARD_D13 | PE12 | SPI4_SCK               |
|      | 7  | GND     | -    | GND                    |
|      | 8  | VREFP   | -    | VREF+                  |
|      | 9  | ARD_D14 | PA12 | I2C5_SDA               |
|      | 10 | ARD_D15 | PA11 | I2C5_SCL               |



Details at <https://bootlin.com/blog/building-a-linux-system-for-the-stm32mp1-connecting-an-i2c-sensor/>



## There's more !

- ▶ Lots of Device Tree topics not covered in this talk
- ▶ range property for address translation
- ▶ Complex Device Tree bindings
  - ▶ Audio, display, camera devices
  - ▶ PCIe
- ▶ Linux kernel API for DT
- ▶ U-Boot tooling for DT manipulation
- ▶ DT overlays
- ▶ etc.



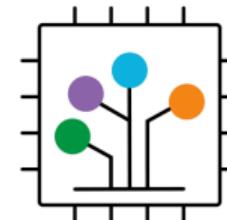
# Resources

- ▶ Device Tree specifications  
<https://www.devicetree.org/>
- ▶ Device Tree bindings  
<https://elixir.bootlin.com/linux/latest/source/Documentation/devicetree/bindings>
- ▶ *Device Tree for Dummies* talk  
<https://www.youtube.com/watch?v=uzBwHFjJ0vU>
- ▶ eLinux.org wiki page on Device Tree  
[https://elinux.org/Device\\_Tree\\_Reference](https://elinux.org/Device_Tree_Reference)
- ▶ Numerous Device Tree talks at the Embedded Linux Conference  
[https://elinux.org/Device\\_Tree\\_Presentations](https://elinux.org/Device_Tree_Presentations)



# Conclusion: about the Device Tree

- ▶ Representation of **non-discoverable hardware**
- ▶ **Tree** of nodes, with properties
- ▶ **Standardization** based on Device Tree **bindings**
- ▶ New description language with lots of properties and sometimes complex bindings
- ▶ Used for numerous CPU architectures
- ▶ Now **widely used** outside of Linux
- ▶ A **must know** for all embedded Linux developers!



Devicetree Specification  
*Release v0.3*

[devicetree.org](https://devicetree.org)

13 February 2020



## Conclusion: about Bootlin

The Bootlin logo consists of the word "bootlin" in a lowercase, sans-serif font. The letter "o" has a small orange triangle pointing upwards and to the right attached to its top right corner.

- ▶ Expertise in embedded Linux
- ▶ Training
- ▶ Engineering services
- ▶ Linux BSP development
- ▶ Kernel drivers
- ▶ Open-source contributor
- ▶ Contact us!
- ▶ [info@bootlin.com](mailto:info@bootlin.com)



Thanks to ST for supporting this webinar!

# Questions ?



<https://bootlin.com>

Slides at <https://bootlin.com/pub/conferences/2021/webinar/petazzoni-device-tree-101/>

# Eingebettete System - Projektaufgabe SS2024

Prof. Dr.-Ing. Jürgen Quade, Hochschule Niederrhein

5.07.2024

## Allgemeines

Die Prüfung im SS 2024 findet in Form einer benoteten, selbstständigen Projektarbeit statt. Im Rahmen dieser Projektarbeit erstellen Sie ein Smarthome-Gateway, über das mit Hilfe des Kommunikationsprotokolls MQTT technische Prozesse überwacht und gesteuert werden können. In der vorliegenden Aufgabe steuern Sie per Smartphone damit eine LED an und visualisieren betriebssystemrelevante Informationen wie beispielsweise *Name des Gateways*, *Laufzeit* oder *IP-Adresse*. Informationen zu MQTT finden Sie weiter unten. Beachten Sie bitte, dass die Aufgabenstellung in einigen Details **individualisiert** ist.

Zur Projektaufgabe gehört neben der Erstellung des Gateways eine Dokumentation im Markdown-Format.



## Voraussetzung

Basis für die Aufgabe ist das während des Semesters entwickelte Buildroot-System, das folgende Komponenten enthält:

- SSH-Server (`dropbear`)
- WLAN-AP (`wpa_supplicant`, inklusive `dnsmasq`)
- Integrierter Treiber zur Ansteuerung einer LED

## Formale Randbedingungen

- Ausgabe der Aufgabenstellung: 15.7.2024
- Letzer Abgabetermin: 9.9.2024 - 23.59 Uhr
- Arbeitsumfang: ca. 40h
- Moodle-Exam-Raum zur Abgabe der Dokumentation: <https://moodle-exam.hsnr.de/course/view.php?id=4298>
- Sciebo-Ordner zur Abgabe der TAR-Datei: <https://hs-niederrhein.sciebo.de/s/OqBrUffr1kFAEPC>

# Vorlage für die Eigenständigkeitserklärung

Eidesstattliche Erklärung  
zur Projektarbeit Eingebettete Systeme im SS2024

Name:  
Matrikelnummer:

Ich versichere durch meine Unterschrift, dass die vorgelegte Arbeit ausschließlich von mir erstellt und verfasst wurde. Es wurden keine anderen als die von mir angegebenen Quellen und Hilfsmittel benutzt.

-----  
Ort, Datum

-----  
Unterschrift

## Abgabe

Einzureichen sind die folgenden Dateien:

- PDF der Dokumentation
- Tar-Datei, die die zum Nachbau notwendigen Dateien enthält

## Generieren der Tar-Datei

Zur Generierung des *Tape Archives* (TAR) legen Sie am besten unterhalb von `~/embedded/` einen Ordner `export` und hier weitere Unterordner an:

```
mkdir ~/embedded/export
cd ~/embedded export
mkdir documentation/ images/ configs/ driver/ scripts/ target/
```

Kopieren Sie danach die benötigten Dateien in die zugehörigen Unterverzeichnisse. Danach erzeugen Sie die Tar-Datei.

```
cd ~/embedded/
tar cvfz 2024-09-08-esy-<IHR NAME>.tgz export/
```

Das TAR **muss** nebst anderen die nachfolgenden Dateien zwingend enthalten:

- Image des selbstgenerierten Userlands “rootfs.cpio”
- Image des selbstgenerierten Kernels “kernel7l.img”
- Buildroot-Konfigurationsdatei “.config” (bitte “`config.buildroot`” nennen)
- Kernel-Konfigurationsdatei “.config” (bitte “`config.linux`” nennen)
- Quellcode des Linux-Treivers `signalri.c`
- `post-build.sh`
- `post-image.sh`
- Entwicklungs-Dokumentation (mit unterschriebener, eingescannter Eigenständigkeitserklärung und Smartphone-Screenshot) sowohl als PDF als auch als Markdown-Datei
- Inhalte der Entwicklungs-Ordner `target/` und `scripts/` (beziehungsweise der Verzeichnisse, in denen Sie die Daten für das Target und die Hilfsskripte für die Generierung abgelegt haben)

Nachfolgend ein beispielhafter Inhalt:

```
tar tvf ~/embedded/2024-09-08-esy-quade.tgz
drwxrwxr-x quade/quade      0 2024-07-05 13:34 export/
drwxrwxr-x quade/quade      0 2024-07-05 13:33 export/images/
-rw-r--r-- quade/quade 18547200 2024-07-05 13:32 export/images/rootfs.cpio
-rw-rw-r-- quade/quade  7713536 2024-07-05 13:32 export/images/kernel7l.img
drwxrwxr-x quade/quade      0 2024-07-05 13:34 export/configs/
-rw-r--r-- quade/quade   107586 2024-07-05 13:33 export/configs/config.buildroot
-rw-rw-r-- quade/quade  210659 2024-07-05 13:33 export/configs/config.linux
drwxrwxr-x quade/quade      0 2024-07-05 13:36 export/driver/
drwxrwxr-x quade/quade      0 2024-07-05 13:38 export/driver/2024-ledone/
-rw-r--r-- quade/quade    758 2024-07-05 13:36 export/driver/20240607-ledone/blink.c
-rw-rw-r-- quade/quade   398 2024-07-05 13:36 export/driver/20240607-ledone/Makefile.application
```

```

-rw-r--r-- quade/quade          290 2024-07-05 13:36 export/driver/20240607-ledone/Makefile
-rw-rw-r-- quade/quade          431 2024-07-05 13:36 export/driver/20240607-ledone/export.sh
-rw-r--r-- quade/quade          3728 2024-07-05 13:36 export/driver/20240607-ledone/led_one.c
               0 2024-07-05 13:35 export/scripts/
               1810 2024-07-05 13:35 export/scripts/post-build.sh
               287 2024-07-05 13:35 export/scripts/post-image.sh
               0 2024-07-05 13:34 export/documentation/
               483 2024-07-05 13:34 export/documentation/Makefile
209899 2024-07-05 13:34 export/documentation/2024-07-03-projekt-esy.pdf
               4404 2024-07-05 13:34 export/documentation/2024-07-03-projekt-esy.md
               0 2024-07-05 13:35 export/target/
               55 2024-07-05 13:35 export/target/index.html
               67 2024-07-05 13:35 export/target/mosquitto.conf
               110 2024-07-05 13:35 export/target/interfaces
               132 2024-07-05 13:35 export/target/ps.cgi
               154 2024-07-05 13:35 export/target/wpa_supplicant.conf
               146 2024-07-05 13:35 export/target/S49mqtpub
               229 2024-07-05 13:35 export/target/S99firewall
               330 2024-07-05 13:35 export/target/dnsmasq.conf
               12530 2024-07-05 13:35 export/target/lighttpd.conf
               3982 2024-07-05 13:35 export/target/modules.conf
               67 2024-07-05 13:35 export/target/S51httpd
               3148 2024-07-05 13:35 export/target/lighttpd.pem
               335 2024-07-05 13:35 export/target/mosquitto-enc.conf
               473 2024-07-05 13:35 export/target/mqtt.sh
               47 2024-07-05 13:35 export/target/sysctl.conf
               27 2024-07-05 13:35 export/target/httpd.conf
               107 2024-07-05 13:35 export/target/S61wpa

```

### Einreichungsort und -frist

- Die Dokumentation **muss** über den Moodle-Exam-Lernraum abgegeben werden: <https://moodle-exam.hsnr.de/course/view.php?id=4298>
- Die TAR-Datei legen Sie auf dem als File-Drop konfigurierten Sciebo-Ordner <https://hs-niederrhein.sciebo.de/s/OqBrUffr1kFAEPC> ab.  
ab. Das Passwort dazu lautet **20ftz24#51**.
- Benennen Sie die TAR-Datei nach folgendem Schema: \<datum\>-esy-\<IHR NAME\>.tgz (Beispiel: **2024-07-05-esy-quade.tgz**).
- Sie können auch mehrere Versionen ablegen; zur Benotung wird die jeweils letzte Version herangezogen.
- Der Zugriff auf den Sciebo-Ordner ist befristet bis zum 9.9.2024 - 23.59 Uhr möglich.

## Bewertung

Die Arbeit gilt als **nicht bestanden**, wenn einer der nachfolgenden Punkte zutrifft:

- Die Einreichung ist **unvollständig**, es fehlen Teile.
- Die Dateien sind nicht **fristgerecht** eingereicht worden.
- Die Eigenständigkeitserklärung liegt nicht vor.
- Die Eigenständigkeitserklärung ist nicht **unterschrieben**.
- Die Projektarbeit ist mit **weniger als 50 Punkten** bewertet worden.
- Das Projekt weist nicht die geforderte **Individualisierung** auf.

## Bewertungskriterien

- Dokumentation: 25 Punkte
- System/Implementierung: 75 Punkte

Notenskala:

00-49 = 5,0

50-54 = 4,0

55-59 = 3,7 | 60-64 = 3,3 | 65-69 = 3,0

70-74 = 2,7 | 75-79 = 2,3 | 80-84 = 2,0

85-90 = 1,7 | 90-94 = 1,3 | 95-100 = 1,0

Insbesondere werden für die folgende Funktionalitäten/Arbeiten Punkte verteilt:

- Booten
- Ansteuerung GPIO
- Einloggen unter spezifischem Namen
- Passwort ist gesetzt
- Vorhandensein des Gerätetreibers unter dem spezifischen Namen
- Gerätedatei mit spezifischem Namen
- Realisierung des Kernelthreads
- Globale Variable mit spezifischem Namen
- Schutz des kritischen Abschnitts
- Automatisches Anlegen der Gerätedatei
- Einstellung der Blinkfrequenz
- WLAN unter dem individualisierten Namen wird aufgespannt
- Einbuchen per Passwort möglich
- Netzadresse 192.168.123.0/24
- Hostname des Systems korrekt gesetzt
- Mosquitto is up and running
- Skript, um Topics zu setzen
- Topics werden gesetzt
- Individualisiertes Topic wird gesetzt
- Skript zur Ansteuerung der LED
- Ansteuerung der LED ist per mqtt möglich
- Ansteuerung der LED per Smartphone ist möglich
- Packaging

# Aufgabenstellung

Das von Ihnen zu entwickelnde, eingebettete System soll die folgenden Anforderungen erfüllen:

- Das zu entwickelnde System bootet per tftp von einem tftp-Server.
  - Das System ist auf einem Raspberry Pi 4 lauffähig.
  - Auf dem System gibt es neben dem Root-Login einen User-Login mit dem Namen **Brimmers** und dem Passwort **Csjnnfst**.
  - Über das System kann eine LED, die an GPIO-23 angeschlossen wird, angesteuert werden.
  - Zur Ansteuerung der LED enthält das System einen Gerätetreiber. Der Gerätetreiber trägt den Namen **signalri.c**.
  - Der Gerätetreiber wird über die Gerätedatei **/dev/led\_onoff\_ri** angesprochen.
  - Gerätedateien werden vom System automatisch angelegt.
  - Der Gerätetreiber realisiert ein Blinklicht, dessen Frequenz durch Schreiben auf die Gerätedatei **/dev/led\_onoff\_ri** an- und eingestellt werden kann:
    - Das Blinken wird über einen Kernel-Thread realisiert, der sich über `wait_event_interruptible_timeout()` schlafen legt.
    - Die Frequenz ist im Treiber in der globalen Variablen `freqBimmers` abgelegt.
    - Der Zugriff auf die Variable muss effizient geschützt sein.
  - Das System spannt ein WLAN mit dem Namen **VimIsTheBest** auf.
  - Das Passwort zum WLAN lautet **ancprcuuaq**.
  - Die IP-Adressen, die per WLAN verteilt werden, stammen aus dem Netz **192.168.123.0/24**.
  - Auf dem System läuft ein SSH-Server, über den man sich remote einloggen kann.
  - Das System hat den Hostnamen **lilibiti66**.
  - Auf dem System läuft der MQTT-Broker **mosquitto**, der beim Hochfahren automatisch gestartet wird.
  - Per Skript auf dem Raspberry Pi sollen folgende Topics per Kommando **mosquitto\_pub** regelmäßig gepublished werden:
    - `system/hostname`
    - `system/ipaddress/eth0`
    - `system/ipaddress/wlan0`
    - `system/uptime`
    - `system/date`
    - `system/Bimmers`
    - `appl/frequency`
  - Über das Topic **system/Bimmers** wird Ihr Name publiziert.
  - Die Ansteuerung der LED erfolgt über das Topic **appl/frequency\_set**
- Hinweise, wie Sie mit Hilfe des Kommandos **mosquitto\_sub** und einem Shell-Skript damit die LED ansteuern, finden Sie unten.
- Auf einem Smartphone soll ein MQTT-Client (MQTT-Dashboard) installiert und konfiguriert werden, so dass über das Smartphone die publizierten Topics visualisiert und das Blinklicht in seiner Frequenz eingestellt werden kann.
  - Zum System gibt es eine Entwicklungsdokumentation, die die nachfolgenden Kriterien erfüllt.

## Dokumentation

Die Dokumentation soll den Leser in die Lage versetzen, mit Hilfe der von Ihnen eingereichten Dateien das System nachzubauen. Er muss dazu beispielsweise nicht die genauen Konfigurationspunkte der Buildroot-Konfig kennen, wohl aber mitgeteilt bekommen, wie er die `config.buildroot` einspielt.

Die Dokumentation ist im **Markdown-Format** zu erstellen und sowohl im PDF-Format als auch als Quellcode (Dateierweiterung `.md`) einzureichen. Im **Moodle-Lernraum** finden Sie dazu ein **Template**, das Hinweise zum Umgang mit Markdown als auch Hinweise zu den Inhalten der Dokumentation gibt. Die Dokumentation wird grob 20 Textseiten umfassen.

# Wissenswertes

## MQTT

*Message Queuing Telemetry Transport (MQTT) ist ein offenes Netzwerkprotokoll für Machine-to-Machine-Kommunikation (M2M), das die Übertragung von Telemetriedaten in Form von Nachrichten zwischen Geräten ermöglicht, trotz hoher Verzögerungen oder beschränkter Netzwerke.[1] Entsprechende Geräte reichen von Sensoren und Aktoren, Mobiltelefonen, Eingebetteten Systemen in Fahrzeugen oder Laptops bis zu voll entwickelten Rechnern.*

*Interessant ist, dass ein MQTT-Server („Broker“) die gesamte Datenlage seiner Kommunikationspartner hält, und so als Zustands-Datenbank benutzt werden kann. So ist es möglich, kleine unperformante MQTT-Geräte mit einem MQTT-Broker zu verbinden, wobei die Geräte Daten einsammeln und/oder Befehle entgegennehmen, während ein komplexes Lagebild nur auf dem MQTT-Broker entsteht und hier oder durch einen leistungsfähigen Kommunikationspartner ausgewertet werden kann. Stelleingriffe können so von einer oder mehreren leistungsfähigen Instanzen an den MQTT-Broker übermittelt und auf die einzelnen Geräte verbreitet werden. Dadurch eignet sich MQTT sehr gut für Automatisierungslösungen und findet im Bereich IoT durch die einfache Verwendung große Verbreitung.*

[Quelle: <https://de.wikipedia.org/wiki/MQTT>]

MQTT ist ein Protokoll zwischen Server und Clients. Der Server ist der MQTT-Broker, der die Verteilstation von Nachrichten nach dem Publisher-Subscriber-Prinzip darstellt. Publisher versenden ihre Daten unter einem Topic an den Broker. Subscriber (Clients) abonnieren beim Broker Topics. Sobald für ein Topic eine neue Nachricht ankommt, verteilt der Broker diese an die Clients, die sich auf die Nachricht abonniert haben.

Die Topics sind ähnlich den Dateipfaden hierarchisch organisiert, vom Systemarchitekten aber frei wählbar.

Beispiel für Topics:

```
system/core/ipaddress system/core/hostname motorcontrol/motorleft/rpm
```

Weitere Infos finden Sie zum Beispiel unter

[\[https://www.embedded-software-engineering.de/was-ist-mqtt-a-725485/\]](https://www.embedded-software-engineering.de/was-ist-mqtt-a-725485/)

## MQTT in der Praxis

Mit `mosquitto` steht sowohl die Implementierung eines Brokers als auch von Clients zur Verfügung. `mosquitto` ist bereits in Buildroot integriert und muss nur ausgewählt werden.

Mit den Programmen `mosquitto_pub` und `mosquitto_sub` stehen Publisher- und Subscriber-Programme zur Verfügung.

Für Tablets und Smartphone gibt es diverse MQTT-Dashboards, z.B. *MQTT Dash* für Android.

**Um per Smartphone auf den Broker zugreifen zu können, muss sich das Smartphone im gleichen Netz befinden. Buchen Sie also Ihr Smartphone ins WLAN des Raspberry Pi ein!**

Bei den Apps ist als erstes der Broker zu kontaktieren.

## Ansteuerung der LED über MQTT

Um kein eigenes C-Programm schreiben zu müssen, wird im Rahmen eines Rapid-Prototyping die LED per Shell-Skript `sw_led_ri.sh` angesteuert. Das Shell-Skript liest von STDIN Nachrichten und schreibt diese in die Gerätedatei `/dev/led_onoff_ri`.

```
mosquitto_sub -t led -h 192.168.123.1 | ./sw_led_ri.sh
```

```
sw_led_ri.sh:
```

```
#!/bin/sh
```

```
while true
do
    echo "waiting for input ..."
    read
    echo "switching led to ${REPLY}"
```

```
echo -e -n "\x${REPLY}" >/dev/led_onoff_ri  
done
```

## User-Verwaltung

Buildroot legt automatisiert User an, wenn eine Datei mit den benötigten Daten vorliegt und in der Buildroot-Konfiguration verlinkt ist. Die Datei `users` hat den folgenden Aufbau:

| username | uid | group | gid | password  | home        | shell   | groups | comment |
|----------|-----|-------|-----|-----------|-------------|---------|--------|---------|
| quade    | -1  | quade | -1  | =blink182 | /home/quade | /bin/sh | adm    | Dozent  |

Das Gleichheitszeichen vor dem Passwort sorgt dafür, dass Buildroot das Passwort als Klartext-Passwort interpretiert und den Hashwert bildet.

Die Datei `users`, die einen Eintrag für den User `foo` enthält, sieht damit folgendermaßen aus:

```
foo -1 bar -1 !=blabla /home/foo /bin/sh alpha,bravo Foo user
```

## Weitere Informationsquellen

<https://buildroot.org/downloads/manual/manual.html>