

# uClinux für High Performance Embedded Systems

Robert Baumgartl

TU Chemnitz  
Juniorprofessur Echtzeitsysteme

4. März 2007 / Chemnitzer Linux-Tage

# Gliederung

## 1 Grundlagen

- Eingebettete Systeme
- Prozessorarchitektur
- Virtueller Speicher
- uClinux in a Nutshell

## 2 Programmierung

- Präliminarien
- „Hello, World!“
- Komplexere Applikationen
- uClinux vs. Linux
- Kernelprogrammierung

## 3 Zusammenfassung

## Ziel des Vortrages

„Embedded bedeutet nicht *lahm* und *klein*.“

# Eingebettete Systeme

*“Embedded computers are defined to be those where the computer is used as a component within a system: not as a computing engine in its own right. [...]”*

*(Jim Cooling, Software Engineering for Real-Time Systems)*

# Eingebettete Systeme – Merkmale

- komplexe Interaktion mit Umwelt und anderen eingebetteten Systemen
- werden häufig autonom betrieben
- hohe Anforderungen an Zuverlässigkeit
- ersetzen zunehmend mechanische Systeme
- Hard- und Software werden gemeinsam entworfen (HW/SW-Codesign)
- Einsatz eines Prozessors „von außen“ häufig nicht erkennbar
- arbeiten häufig unter Echtzeit-Bedingungen
- Kosten für Reparatur übersteigen Kosten des Systems.

# Microcontroller Units (MCU)

= Mikroprozessor mit

- Peripherie on-Chip (Timer, Speicher, Schnittstellen, Wandler, ...)
- sehr große Vielfalt, organisiert in Familien
- hohe Codedichte
- mit und ohne virtuellem Speicher
- niedrige Verlustleistung, **niedriger Preis**
- $\rightsquigarrow$  Einsatz in *Eingebetteten Systemen*

# Klassifizierung per Verarbeitungsbreite

| 8 Bit            | 16 Bit                         | 32 Bit            |
|------------------|--------------------------------|-------------------|
| Intel MCS-51     | Intel MCS-96                   | Intel XScale      |
| Freescale 68HC11 | DSP 56800                      | Freescale MPC 860 |
| Atmel AVR        | Infineon XC166                 | DEC StrongARM     |
| Zilog Z8         | <b>Analog Devices Blackfin</b> |                   |

**Tabelle:** Beispiele für Microcontroller-Familien

# Digitale Signalprozessoren (DSP)

= Microcontroller mit

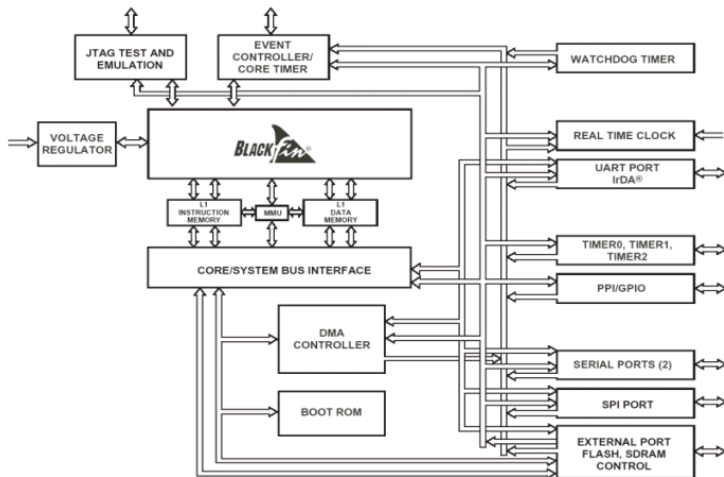
- schnelle Multiplizier-Addier-Operation (MAC)
- sehr hohe Bandbreite RAM↔CPU
- spezielle Adressierungsmodi (Ringpuffer, FFT)
- “Zero-Overhead-Looping”
- minimale Interrupt-Latenz
- umfangreiche on-chip-Peripherie (AD/DA-Wandler, Schnittstellen, RAM, DMA-Controller, Timer)
- keine MMU
- Programmierung in C und/oder Assembler



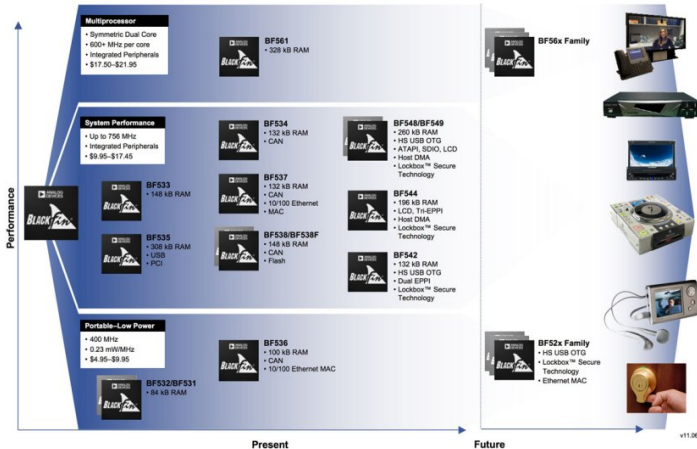
## Prozessor AD Blackfin BF-537

- DSP-Microcontroller-Hybride
- Takt maximal 750 MHz
- 132 kByte internes RAM
- extrem niedrige Leistungsaufnahme
- bis zu zwei Fetch-Operanden pro Takt, zwei MAC-Operationen pro Takt
- insgesamt 40 Register á 32 Bit
- PPI, SPI, Ethernet, CAN on-board
- Preis: 15-19 US\$ (1000er Quantitäten)
- besitzt MMU, diese kann jedoch keinen VM
- Cache und Sprungvorhersage

# Prozessor AD Blackfin BF-537



# Blackfin Roadmap



# Entwicklungsplattform: STAMP-Board



# Entwicklungsplattform: STAMP-Board

- BF537, 500 MHz
- 64 MB SDRAM, 4 MByte Flash
- RS-232, 100 MBit Ethernet, 2xCAN
- JTAG, Peripherie-Konnektoren
- „Low Cost Eval Platform“ (164 € bei Spoerle)

# Virtueller Speicher

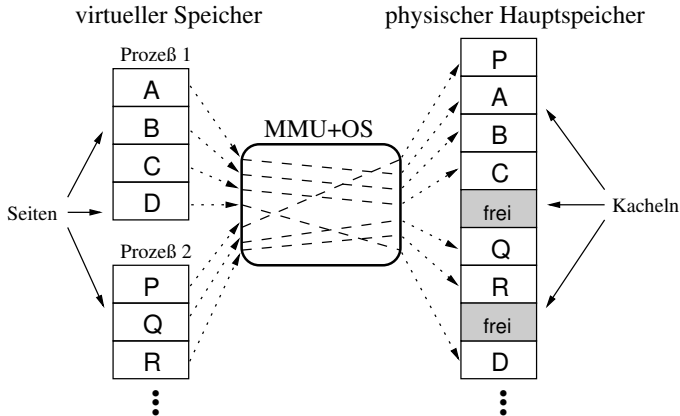
## Ziele

- Erweiterung des physischen Speichers durch Festplatte
- Schutz des Betriebssystems und der Nutzeraktivitäten voreinander
- Gestreute Speicherung (keine kontinuierlichen Speicherbereiche erforderlich)

Klassische Abstraktion: *Prozeß* = „Programm in Ausführung, jedoch in eigenem Adreßraum“

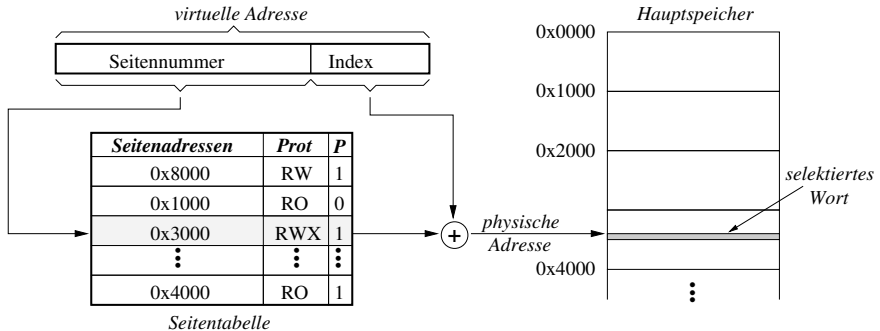
# Virtueller Speicher

## Funktionsprinzip



# Virtueller Speicher

## Umsetzung der virtuellen Adresse mittels Seitentabelle





# uClinux in a Nutshell

## uClinux

- = Linux ohne Virtuellen Speicher
- = Linux für Microcontroller ohne MMU
- wird ausgesprochen: „You-see-Linux“

# Präliminarien

- Inbetriebnahme der Platine
  - Initialkommunikation über RS232
  - Ethernet
  - Stromversorgung
- lädt Bootloader (U-Boot) und uClinux vom Flash
- Installation der Entwicklungswerkzeuge
  - Quelle: <http://blackfin.uclinux.org/gf/project/toolchain>
  - installiert nach `/opt/uClinux`

# „Hello, World!“

- ① Programmentwicklung
- ② Übersetzung für und Test im Simulator
  - bfin-uclinux-gdb
  - target sim
  - ...
- ③ Übersetzung für reale Hardware
- ④ Übetragung zum DSP (ftp, nfs)
- ⑤ Test

## Punktprodukt; C-Version

```
int dot(short *x, short *y, int len)
{
    int i, dot;

    dot = 0;
    for(i=0; i<len; i++)
        dot += x[i] * y[i];

    return dot;
}
```

## Punktprodukt; Asm-Version

```
I0 = x;  
I1 = y;  
A1 = A0 = 0;  
R0 = [I0++] || R1 = [I1++];  
LOOP dotproduct LC0 = len >> 1;  
LOOP_BEGIN dotproduct;  
    A1 += R0.H * R1.H, A0 += R0.L * R1.L  
    || R0 = [I0++]  
    || R1 = [I1++];  
LOOP_END dotproduct;  
R0 = (A0 += A1);  
dot = R0 >> 1;
```

# Punktprodukt; Ergebnisse

|             | Blackfin |     | Pentium M | Core 2 Duo |
|-------------|----------|-----|-----------|------------|
| $f_c$ [MHz] | 500      |     | 2000      | 2130       |
|             | C (-O9)  | Asm | C (-O9)   | C (-O9)    |
| Zyklen      | 405      | 53  | 684       | 384        |
| $t_e$ [ns]  | 810      | 106 | 342       | 108        |

**Tabelle:** Zyklenzahlen und Ausführungszeiten (len=100)

# uClinux vs. Linux

## Verlust des Schutzes

Alles liegt in ein- und demselben Adreßraum!

Konsequenzen:

- jede Aktivität kann das Gesamtsystem korrumpieren
- Verletzung der Integrität kaum zu erkennen (kein SIGSEV etc.)
- Ausnahme: NULL-Pointer-Dereferenz führt zu Kernel-Oops
- vgl. Kernelprogrammierung
- Demonstration: `mem-viol.c`

# uClinux vs. Linux

## Prozeßzeugung

- Semantik von `fork( )` nicht realisierbar
- liefert stets -1 (d. h. funktioniert nicht mehr)
- Nutzung von `vfork( )` stattdessen (differierende Semantik!)
- Demonstration: `fork1.c`
- Alternativen: `clone( )`, `pthread`s
- Demonstration: `fork2.c`



# uClinux vs. Linux

## Verwaltung des Kernelspeichers

- Linux: Buddy-Allocator (kmalloc())
  - schnell
  - viel Verschnitt
- uClinux: Modifikation (Kmalloc2)
  - identisch bei Anforderungen  $\leq 4$  kByte
  - sonst Aufrundung auf  $n \cdot 4$  kByte
  - kleine Anforderungen ( $\leq 8$  kByte) vom Anfang des Speichers bedient
  - große Anforderungen vom Ende
  - (noch) nicht im 2.6er uClinux

# uClinux vs. Linux

## Verwaltung des Applikationsspeichers

- kein Paging mehr  $\Rightarrow$  reale Gefahr der Fragmentierung
- keine Defragmentierung des Speichers möglich
- Stackgröße konstant! (zur Übersetzungszeit determiniert)
- `brk( )` erlaubt nur noch das *Verkleinern* des Datensegmentes (da fix)
- `malloc( )` nutzt `mmap( )`
- `mmap( )` alloziert Kernelspeicher

# Implementierung eines einfachen Treibers

Demonstration ... sofern Zeit

## Wozu das Ganze?

- Basis für Consumerelektronik mit hohem Rechenzeitbedarf
- Beispiel: Open-Source-Telefon  
(<http://www.rowetel.com/blog/?p=15>)
- einziger DSP mit (vernünftigem) Open-Source-Support

Was machen *wir* mit dem Blackfin?

- Portierung eCos
- Einbau von Schutz (Kernel  $\leftrightarrow$  User Space)
- ideale Echtzeit-Plattform
  - ADEOS und Xenomai existieren
  - sehr deterministisches Ausführungszeitverhalten
- Autumn DSP Programming School

# Einschätzung

- gegenwärtig sehr aktives Open-Source-Projekt
- jedoch klein im Vergleich zu Linux
- Qualität des Codes und der Pakete sehr hoch
- **vorbildliche** Unterstützung durch Community

→ Embedded heißt weder *lahm* noch *klein*! ←