Linux in Atronic's Electronic Gaming Machine (EGM)

Jörg Faschingbauer
jfaschingbauer@atronic.com

Intro: Über Atronic, 1

- Casino Slot Machines
- Seit 1993
- 700-800 Beschäftigte
- Produkte aktiv in 91 Ländern
- > 200 Spiellizenzen (incl. Nevada)

Intro: Über Atronic, 2

Intro: Über Atronic, 3

Intro: Über mich

- Technische Mathematik, TU Graz bis 1993
- Assistent ebendort, bis 1997
- Hyperwave, Salomon, Atronic
- Unix (SunOS, HP-UX) seit ca. 1990, Linux seit ca. 1995, Gentoo seit 2003

Motivation

- “Embedded”: alles, wo man nicht sieht, was drin ist.
- Kernel 2.6: absolut geeignet
  - Hardwareunterstützung
  - Realtimeunterstützung
  - Unendlich Konfigurierbar
- Mobiltelefone
- Home-Router
- Server-Farmen

Agenda

- Hardwareüberblick
- “Board Support Package”
  - Beispiel: zwei typische Driver
- Portierung der Applikation
- Applikationsentwicklung
- Erfahrungen
- Soziologie

Gegenwärtiges Produkt

- Hardware
  - 2 Boards: Hauptlogik + Kommunikation
  - Motorola 68332
  - Etwas angestaubt
- Software
  - ENEA OSE 4 (Embedded Realtime)

Hardware, 1

- Überprüfbarkeit und Sicherheit
  - Gesetzlich strengstens geregelt
  - Non-Volatile RAM
  - “Power Down” Türüberwachung durch Microcontroller
- “Parallele” Devices
  - Buttons, Lampen, LEDs, mechanical Meters
  - Münzeinwurf mit Münzerkennung und Lichtschranken
  - Münzauszahlung und andere Motoren

Hardware, 2

- Serielle Devices
  - Ticketprinter
  - Geldschein- und Ticketerkennung
  - Card Reader
  - USB: Patent mit fraglichem Hintergrund
- Diverse serielle Protokolle
  - Jackpotcontroller
  - Accounting

Hardware, 3

- Prozessor
  - AMCC PPC440EP. “System on Chip”
  - DDR SDRAM Controller
  - PCI Controller
  - 2 x 100 Mbit Ethernet
  - USB
  - 4 x UART
  - und, und, und ...
  - ansonsten irrelevant – Linux funktioniert

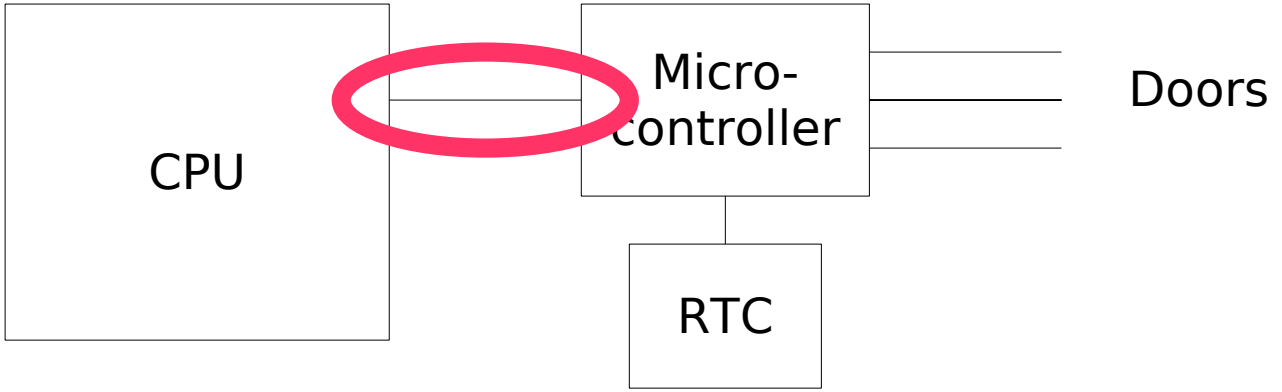
Linux Technische Daten

- Kernel
  - Beginn mit 2.6.15 (BlueCat Linux)
  - Mittlerweile Vanilla 2.6.24.2
- “Board Support Package” (BSP)
  - Externe Auftragsarbeit
  - Firma RST, incl. Subcontractor (unser Herr Ocker, www.reccoware.de)
  - Basis Hardwareanbindung
  - Drivers

Drivers im Allgemeinen

- Meistens für Hardware-Interaktion (no na)
  - /dev/zero z.B. nicht
- Interrupt handling, Device Kommunikation
- Interaktion mit User Space über Device Nodes und System Calls (`open()`, `read()`, `write()`, `close()`, `select()`, `ioctl()`)
- Interfaces sind Geschmacksfrage!

Atronic Drivers: Microcontroller

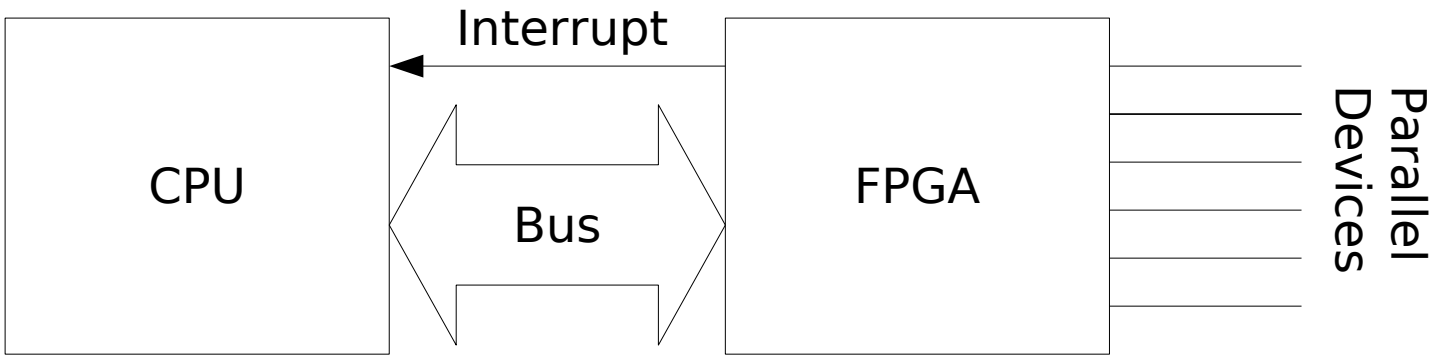


- Requests: “Get door log”, “Get current time”
- Request/Response für spezifische Devicefunktionalität, fixe Parameter
  - `ioctl()` am besten geeignet
- Intern zwei Leitungen: Clock und Daten

Atronic Drivers: Parallele Devices – FPGA, 1

- Viele Pins zyklisch nach Änderungen überprüft (“gepollt”)
- Historisch in Software, vom Prozessor
- Heute: FPGA (“Field Programmable Gate Array”)
  - www.opencores.org
- Polling in Hardware/Firmware, Interrupt bei Änderung

Atronic Drivers: Parallele Devices – FPGA, 2



- Interrupt == Event
- Driver muss Event dem Userspace mitteilen
- System Calls `read()` und `select()`

Atronic Drivers: Parallele Devices – FPGA, 3

- Testen einer Button-Bank

```
$ cat /dev/qspi/2
r 7fff 7ffb f7ca
r 7ffb 7fff fc0e
...
```

- Synchrones Lesen eines Buttons

```
$ cat /sys/devices/platform/pcsfpga.0/qspi/rx2
7fff
```

- Einschalten einer Lampe

```
$ echo 7eff > \
/sys/devices/platform/pcsfpga.0/qspi/tx2
```

Applikation, 1

- Historisch: EPROMS
  - Realtime OS: ENEA OSE
  - Kernel und Applikation statisch gelinkt
  - “Relativ” monolithisch
- Unklare Trennung OS/Applikation
  - Teilweise Logik in Interrupts
  - Direkter Hardwarezugriff
- Windows-Port für Entwicklung

Applikation, 2

- Zertrümmerung des Monolithen
  - Bootprozess nicht Aufgabe der Applikation
  - Diverse Services ebenso nicht
    - Watchdog
    - Logging, Log Rotation
    - Integrity Checks
    - Firmware Download
    - ...

Applikation, 3

- Realtime
  - Fixe Prioritäten
  - Alle Gefahren, die man sich vorstellen kann (Priority Inversion!)
  - Ressourcensparend, weil einfach
- Message Passing Architektur
  - Ca. 100 Threads
  - Message Queues
  - Pthread

Applikation, 4

- Realtime unter Linux
  - Mit Kernel 2.6 kennt Linux Realtime
  - ... wenn auch nur “Soft”
  - Wichtig: “If a process with a higher static priority gets ready to run, the current process will be preempted.”
  - Applikation dahingehend schwer zu ändern: “Priority Based Synchronization”
  - Aber: Vanilla Soft Realtime reicht allemal!

Applikationsentwicklung, 1

- ~80% Windows, Visual Studio
- Mittlerweile auch Linux/PC Port
- Build: generiert aus allgemeinem Format für:
  - omake (Clearcase make)
  - M$ Visual Studio
  - GNU Make (Windows, Linux/x86)

Applikationsentwicklung, 2

- Debugging
  - strace
  - valgrind
    - noch nicht für PowerPC, aber ansonsten genial
  - gdb
    - lokal, remote, grafisch, ... (wie man's braucht)
  - procfs, sysfs
    - Man sollte drin stöbern und es sich auf der Zunge zergehen lassen

Erfahrungen, 1

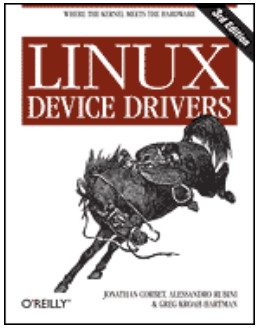
- Vanilla Kernel
  - Außer, wenn man Hard Realtime braucht
  - Zwei, drei Bücher lesen
  - Geld ist besser in eigenen Code investiert
- Kernel Hacking
  - Extern, zumindest für den Anfang
  - JTAG Debugger? Auch ein RT Embedded Spleen.
  - Version Control? Wenn, dann Git. Sonst kernel.tar.bz2 und gendiff.

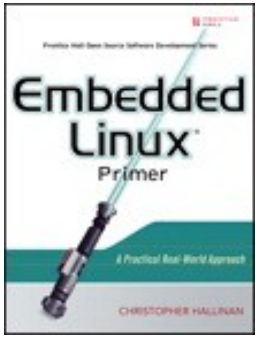
Erfahrungen, 2

- “Realtime”-Schmerzen
  - TTY Layer: Bottom Half, aber `low_latency` konfigurierbar
  - Big Kernel Lock: TTY vs. NFS-Logging (Argh!)
  - Alles andere hausgemacht

Empfohlene Bücher (Kernel)

- Linux Kernel Development (Robert Love)
- Linux Device Drivers (Jonathan Corbet et al.)
- Embedded Linux Primer (Christopher Hallinan)





Empfohlene Bücher (Userspace)

- Advanced Programming in the UNIX Environment (W.Richard Stevens, Stephen Rago)
- Programming with POSIX Threads (David Butenhof)