# Linux auf dem Nios II Softcore Prozessor

Tobias Klauser <klto@zhaw.ch>

Institute of Embedded Systems
Zürcher Hochschule für Angewandte Wissenschaften

30. August 2011

Zürcher Hochschule
für Angewandte Wissenschaften
School of Engineering
InES Institute of Embedded Systems

1. Agenda
2. Institute of Embedded Systems ZHAW
3. Übersicht FPGA und Softcore Prozessoren
4. Übersicht Linux
5. Linux auf dem Nios II
6. Anwendungsbeispiele

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InES Institute of Embedded Systems

# Institute of Embedded Systems

- F&E Institut der ZHAW
- Rund 45 Mitarbeiter
- Schwerpunkte
  - Industrielle Kommunikation
  - Zeitsynchronisation und Hochverfügbarkeit
  - System on Chip Design
  - Wireless Communication, Low Power
  - Entwicklungsmethoden
- Projekte in Zusammenarbeit mit Industriepartnern
- Für mehr Informationen: `http://ines.zhaw.ch`

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InES Institute of Embedded Systems

Institute of Embedded Systems

- F&E Institut der ZHAW
- Rund 45 Mitarbeiter
- Schwerpunkte
  - Industrielle Kommunikation
  - Zeitsynchronisation und Hochverfügbarkeit
  - System on Chip Design
  - Wireless Communication, Low Power
  - Entwicklungsmethoden
- Projekte in Zusammenarbeit mit Industriepartnern
- Für mehr Informationen: `http://ines.zhaw.ch`

1. RT Ethernet: Profinet, Powerlink
2. Präzise Zeitsynchronisierung: IEEE1588 (PTP), Redundanzprotokolle: PRP, HSR

# FPGA und Softcore

- Field Programmable Gate Array
  - Schaltung wird in einer Beschreibungssprache (VHDL, Verilog) codiert
  - Code wird synthetisiert und FPGA damit programmiert
  - Änderungen am Design relativ einfach, Custom Systems
  - Parallelität
- Softcore
  - Prozessor innerhalb des FPGA implementiert
  - Fast immer Bestandteil eines System-on-a-Chip
  - Konfigurierbar (z.B. MMU), erweiterbar (z.B. Custom Instructions)
  - Proprietär, herstellerabhängig: Nios II (Altera), MicroBlaze (Xilinx), Cortex M1 (ARM)
  - Open-Source Softcores: Gaisler LEON, OpenRISC, LatticeMico32

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InES Institute of Embedded Systems

FPGA und Softcore

- Field Programmable Gate Array
  - Schaltung wird in einer Beschreibungssprache (VHDL, Verilog) codiert
  - Code wird synthetisiert und FPGA damit programmiert
  - Änderungen am Design relativ einfach, Custom Systems
  - Parallelität
- Softcore
  - Prozessor innerhalb des FPGA implementiert
  - Fast immer Bestandteil eines System-on-a-Chip
  - Konfigurierbar (z.B. MMU), erweiterbar (z.B. Custom Instructions)
  - Proprietär, herstellerabhängig: Nios II (Altera), MicroBlaze (Xilinx), Cortex M1 (ARM)
  - Open-Source Softcores: Gaisler LEON, OpenRISC, LatticeMico32

1. FPGA - ASIC, Rapid Prototyping
2. schnelle Signalverarbeitung in FPGA möglich
3. Viele Schaltungen benötigen heutzutage uC, Softcore einfache Lösung für Projekte mit FPGA
4. Softcore für Prototypen aber auch für Endprodukt möglich

# Nios II

- 32-bit RISC Softcore-Prozessorarchitektur für Altera FPGAs
- 3 Basisvarianten:
  - Economy (600-700 LEs)
  - Standard (1200-1400 LEs)
  - Fast (2600-3000 LEs)
- Konfigurierbar mit oder ohne Memory Management Unit (MMU)
- Optionale Hardware MUL/DIV Instruktionen, Custom Instructions
- Peripherie (Timer, UART, Ethernet MAC, Flash Controller, ...) als IP über SOPC System konfigurierbar

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InES Institute of Embedded Systems

Nios II

- 32-bit RISC Softcore-Prozessorarchitektur für Altera FPGAs
- 3 Basisvarianten:
  - Economy (600-700 LEs)
  - Standard (1200-1400 LEs)
  - Fast (2600-3000 LEs)
- Konfigurierbar mit oder ohne Memory Management Unit (MMU)
- Optionale Hardware MUL/DIV Instruktionen, Custom Instructions
- Peripherie (Timer, UART, Ethernet MAC, Flash Controller, ...) als IP über SOPC System konfigurierbar

1. Flexible Anpassung an Bedürfnisse im Laufe des Projekts
2. SOPC = System on Programmable Chip: System von Altera zur Konfiguration von vorgefertigten IP-Komponenten

# Wieso ein Betriebssystem?

- Multi-Tasking, Multi-User
- Memory Management, Memory Protection (MMU benötigt)
- TCP/IP Stack
- Filesysteme
- Bestehende Software und Treiber

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InES Institute of Embedded Systems

## Wieso ein Betriebssystem?

- Multi-Tasking, Multi-User
- Memory Management, Memory Protection (MMU benötigt)
- TCP/IP Stack
- Filesysteme
- Bestehende Software und Treiber

1. mehrere (quasi) nebenläufige Prozesse bzw. Benutzer
2. Jeder Prozess (und Kernel) in eigenem virtuellem Adressraum, Schutz vor fehlerhaften Speicherzugriffen in schlecht programmierter Software
3. Mit MMU ist zwingend ein OS nötig bzw. macht einen MMU ohne OS keinen Sinn
4. Viele Hersteller bieten heutzutage ihre Software bzw. Treiber für Linux an, bestehende Software im Haus

# Wieso (Embedded) Linux?

- Skalierbarkeit: Von "Big Iron" bis zum Embedded System
  - Starke Verbreitung im Bereich Embedded Systems (z.B. Android auf Smartphones)
- Umfangreiches Software-Ökosystem, grosse Community
- Einheitliche Entwicklungsumgebung (Unix-artig, POSIX-Standard)
- Portierungen auf zahlreiche ($\sim$ 30) Prozessor-Architekturen
- Niedrige Kosten, freie Lizenzen

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InES Institute of Embedded Systems

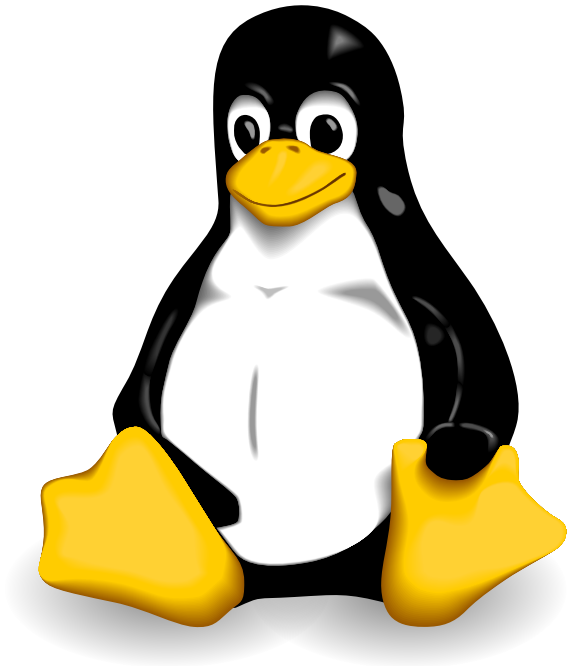
## Wieso (Embedded) Linux?

- Skalierbarkeit: Von "Big Iron" bis zum Embedded System
  - Starke Verbreitung im Bereich Embedded Systems (z.B. Android auf Smartphones)
- Umfangreiches Software-Ökosystem, grosse Community
- Einheitliche Entwicklungsumgebung (Unix-artig, POSIX-Standard)
- Portierungen auf zahlreiche ($\sim 30$) Prozessor-Architekturen
- Niedrige Kosten, freie Lizenzen

1. Supercomputer (Anteil von über 90% bei Top500), Cluster, Mainframe (IBM zSeries), Server, PC, Netzwerk-Komponenten, Unterhaltungselektronik, Smartphones, ...
2. Community bietet Support, Austausch unter Nutzern, grosse Auswahl bzw. Alternativen an Software
3. Entwicklungssystem $=$ Zielsystem
4. teilweise exotische Architekturen, Linux als einziges unterstütztes OS
5. Source Code verfügbar

# Embedded Linux

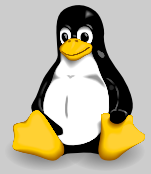
- Linux Kernel als Kernkomponente eines Linux Systems
- Libraries und Applikationen werden zusätzlich benötigt, um ein komplettes System zu erhalten
- Minimale Voraussetzungen:
  - 32-bit Prozessor (mit oder ohne MMU)
  - unterstützte Toolchain (GCC, binutils, ...)
  - libc (uClibc, eglibc, ...)



Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InES Institute of Embedded Systems

## Embedded Linux

- Linux Kernel als Kernkomponente eines Linux Systems
- Libraries und Applikationen werden zusätzlich benötigt, um ein komplettes System zu erhalten
- Minimale Voraussetzungen:
  - 32-bit Prozessor (mit oder ohne MMU)
  - unterstützte Toolchain (GCC, binutils, ...)
  - libc (uClibc, eglibc, ...)



1. Embedded Linux: Sammelbegriff für Embedded Systems mit Linux Kernel
2. Applikationen und Libraries werden üblicherweise durch Distribution gesammelt
3. libc = C Library

# Linux/uClinux auf dem Nios II

- Linux unterstützt Nios II mit *und* ohne MMU
  - Erster Port von Microtronix für Nios II NOMMU (Kernel 2.6.11)
  - Separater Port von Wind River für Nios II MMU (Kernel 2.6.21)
- Open Source Nios II Linux (alterawiki.com/wiki/Linux)
  - MMU und NOMMU Ports zusammengeführt
  - Zahlreiche Verbesserungen, Anpassungen an aktuelle Linux Kernel Versionen
  - Integration in Mainline (kernel.org) geplant
  - Verwendet uClinux-dist als Distribution
- Kommerzielle Anbieter
  - Auf Basis von Open Source Nios II, eigene Distribution
  - Wind River, Timesys, SLS

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InES Institute of Embedded Systems

## Linux/uClinux auf dem Nios II

- Linux unterstützt Nios II mit *und* ohne MMU
  - Erster Port von Microtronix für Nios II NOMMU (Kernel 2.6.11)
  - Separater Port von Wind River für Nios II MMU (Kernel 2.6.21)
- Open Source Nios II Linux (alterawiki.com/wiki/Linux)
  - MMU und NOMMU Ports zusammengeführt
  - Zahlreiche Verbesserungen, Anpassungen an aktuelle Linux Kernel Versionen
  - Integration in Mainline (kernel.org) geplant
  - Verwendet uClinux-dist als Distribution
- Kommerzielle Anbieter
  - Auf Basis von Open Source Nios II, eigene Distribution
  - Wind River, Timesys, SLS

1. NOMMU 2004, MMU 2009, im Auftrag von Altera
2. Port momentan noch ausserhalb von offiziellem Linux Kernel
3. Zusätzliche Treiber
4. Version momentan bei 3.0 bzw. 3.1-rc4

# Nios II mit oder ohne MMU?

| **MMU** | **NOMMU** |
|---|---|
| *mehr* LEs im FPGA | *weniger* LEs im FPGA |
| Virtual Memory, separater Adressraum für jeden Prozess | ein globaler Adressraum für alle Prozesse und Kernel |
| Shared Libraries | Libraries werden statisch zu jedem Binary gelinkt |
| ELF Fileformat | Flat Fileformat |
| keine Änderungen an Userspace-Software notwendig | u.U. Anpassungen an Software notwendig (z.B. kein `fork()`) |
| langsamer | schneller |

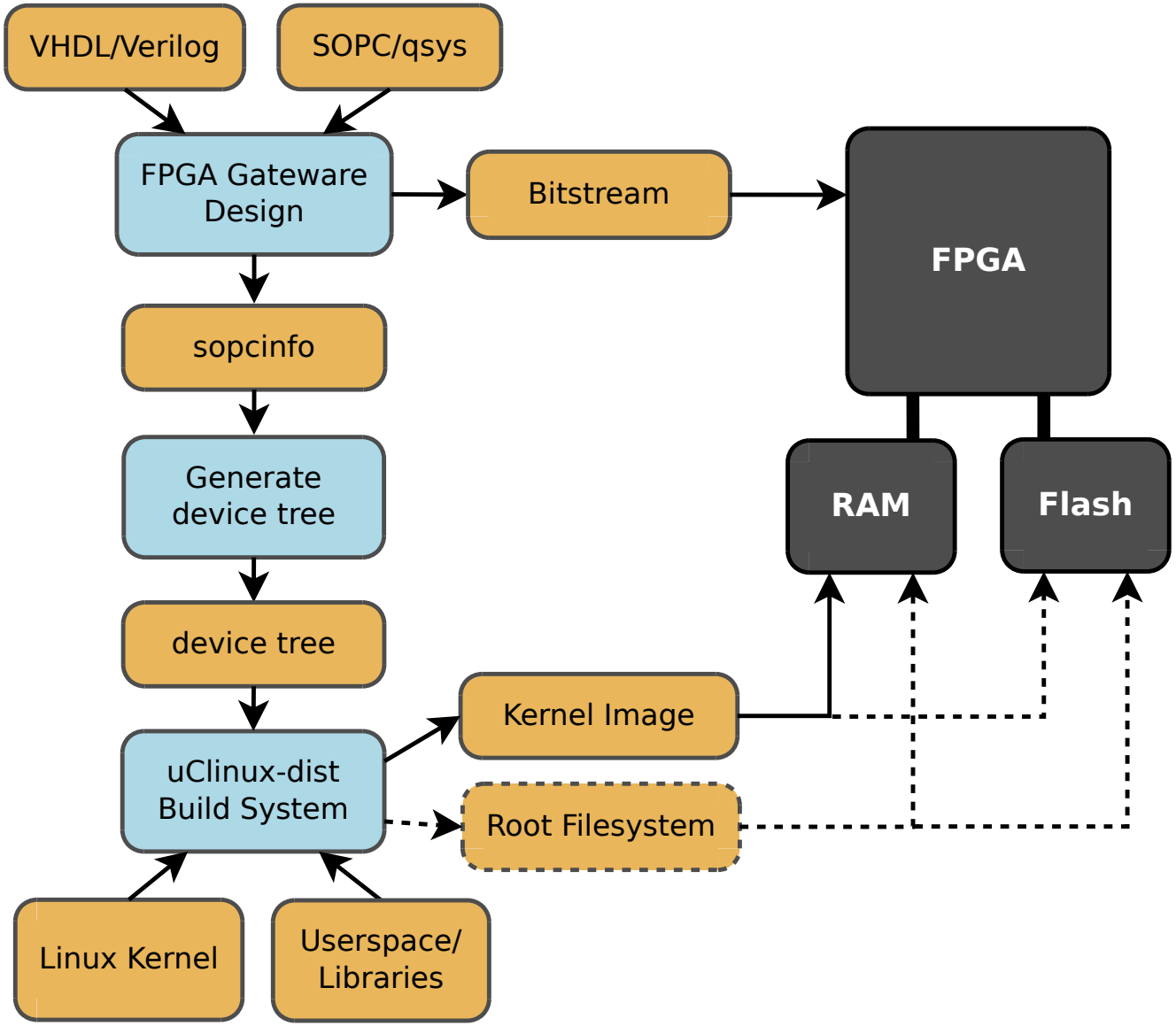
Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InES Institute of Embedded Systems

## Nios II mit oder ohne MMU?

| MMU | NOMMU |
|---|---|
| *mehr* LEs im FPGA | *weniger* LEs im FPGA |
| Virtual Memory, separater Adressraum für jeden Prozess | ein globaler Adressraum für alle Prozesse und Kernel |
| Shared Libraries | Libraries werden statisch zu jedem Binary gelinkt |
| ELF Fileformat | Flat Fileformat |
| keine Änderungen an Userspace-Software notwendig | u.U. Anpassungen an Software notwendig (z.B. kein `fork()`) |
| langsamer | schneller |

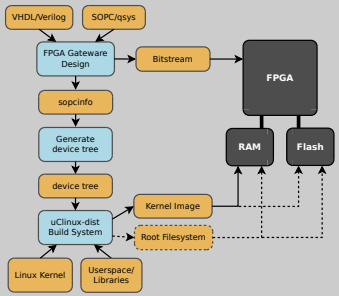
1. LE = Logic Elements, Logikzellen
2. Prozesse können bei unsauberer Programmierung in Speicher von Kernel oder anderen Prozessen schreiben bzw. lesen
3. durch statisches Linken werden Binaries grösser
4. Context Switches werde mit MMU aufwändiger

# Development Workflow




Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InES Institute of Embedded Systems

Development Workflow




1.

# Linux Image & Root Filesystem

- Kernel Image als komprimiertes ELF-Binary
  - Kann direkt in RAM geladen und gestartet werden
  - Wird u-boot verwendet, muss das Image konvertiert werden
- Root Filesystem in einem `initramfs` an Kernel angehängt
- Keine Persistenz über Reboot hinweg
  - Alternative: Root FS separat in Flash speichern
  - NFS (Network Filesystem) verwenden

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InES Institute of Embedded Systems

## Linux Image & Root Filesystem

- Kernel Image als komprimiertes ELF-Binary
  - Kann direkt in RAM geladen und gestartet werden
  - Wird u-boot verwendet, muss das Image konvertiert werden
- Root Filesystem in einem `initramfs` an Kernel angehängt
- Keine Persistenz über Reboot hinweg
  - Alternative: Root FS separat in Flash speichern
  - NFS (Network Filesystem) verwenden

1. Eigener Mini-Bootloader innerhalb des Kernel-Images
2. u-boot kann aus Flash oder via Netzwerk (TFTP) laden
3. oder nur Teile von RootFS (z.B. Conffiles) in Flash
4. NFS während Entwicklung

# Device Tree (devicetree.org)

- Strukturierte Beschreibung der Hardware
- Integration mit Linux:
  - Als Bestandteil des Kernel Images
  - Im Flash (nur mit u-boot)
  - Laden über TFTP (nur mit u-boot)
- Linux extrahiert Device Tree, lädt Treiber entsprechend
- Von zahlreichen Linux-Ports eingesetzt (ARM, PowerPC, MicroBlaze, OpenRISC)

```
cpu_0: cpu@0x0 {
  compatible = "ALTR,nios2-9.1";
  clock-frequency = <50000000>;
  dcache-size = <2048>;
  icache-size = <4096>;
  ALTR,implementation = "fast";
  ALTR,has-div;
  ALTR,has-mul;
  ALTR,reset-addr = <0xc4000000>;
  ...
};

uart_0: serial@0x6000000 {
  compatible = "ALTR,uart-9.1";
  reg = <0x6000000 0x20>;
  interrupt-parent = <&cpu_0>;
  interrupts = <2>;
  current-speed = <115200>;
  clock-frequency = <50000000>;
};
```

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InES Institute of Embedded Systems

## Device Tree (devicetree.org)

- Strukturierte Beschreibung der Hardware
- Integration mit Linux:
  - Als Bestandteil des Kernel Images
  - Im Flash (nur mit u-boot)
  - Laden über TFTP (nur mit u-boot)
- Linux extrahiert Device Tree, lädt Treiber entsprechend
- Von zahlreichen Linux-Ports eingesetzt (ARM, PowerPC, MicroBlaze, OpenRISC)

```
cpu_0: cpu@0x0 {
  compatible = "ALTR,nios2-9.1";
  clock-frequency = <50000000>;
  dcache-size = <2048>;
  icache-size = <4096>;
  ALTR,implementation = "fast";
  ALTR,has-div;
  ALTR,has-mul;
  ALTR,reset-addr = <0xc4000000>;
  ...
};

uart_0: serial@0x6000000 {
  compatible = "ALTR,uart-9.1";
  reg = <0x6000000 0x20>;
  interrupt-parent = <&cpu_0>;
  interrupts = <2>;
  current-speed = <115200>;
  clock-frequency = <50000000>;
};
```

1. Urspünge im Opefirmware-Projekt (Bootumgebung für OS, analog BIOS)
2. Hardware-Details wie Adressen, Interrupt-Nummern etc. müssen nicht im OS hart codiert werden, OS wird unabhängiger von Hardware
3. Flattened Device Tree -¿ Kompilierung in Binärformat
4. Im Idealfall muss Linux bei geringfügiger Anpassung der HW (z.B. neue IRQ einer Komponente) nicht neu kompilert werden

# Bootprozess

1. Laden des FPGA-Bitstreams
2. Laden des Linux Images inkl. Reset des Nios
3. Nios beginnt mit Ausführung des Linux Kernels
4. Linux Kernel erkennt Hardware anhand des Device Tree und lädt entsprechende Treiber
5. `initramfs` wird durch den Kernel ins RAM entpackt und gemountet
6. Init-Prozess (`/sbin/init`) wird ausgeführt
7. Optional: Mounten von von Flash- bzw. Netzwerk-Filesystemen
8. Weitere Userspace-Prozesse (Daemons etc.) werden durch Init gestartet

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InES Institute of Embedded Systems

Bootprozess

1. Laden des FPGA-Bitstreams
2. Laden des Linux Images inkl. Reset des Nios
3. Nios beginnt mit Ausführung des Linux Kernels
4. Linux Kernel erkennt Hardware anhand des Device Tree und lädt entsprechende Treiber
5. `initramfs` wird durch den Kernel ins RAM entpackt und gemountet
6. Init-Prozess (`/sbin/init`) wird ausgeführt
7. Optional: Mounten von von Flash- bzw. Netzwerk-Filesystemen
8. Weitere Userspace-Prozesse (Daemons etc.) werden durch Init gestartet

1. ohne u-boot
2. Laden von FPGA-Bitstream und Linux Image via JTAG oder aus dem Flash
3. Prozessor-spefisische Initialisierung, MM, Scheduler
4. Device Tree einkompiliert
5. Was nach init passiert, wird durch initskripts festgelegt
6. z.B. könnte auch RootFS aus Flash oder anderem Festspeicher gemountet werden

# Anwendungsbeispiel - Redundantes Kommunikationsnetzwerk

- Nios II mit MMU (anfangs ohne MMU)
- Ethernet-basiert, optional SHDSL
- Parallel Redundancy Protocol
- Software-Komponenten (COTS und Eigenentwicklung) u.a. für SHDSL, Webserver, Remote Update, SNMP, NTP
- Treiber für eigene IP-Komponenten

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InES Institute of Embedded Systems

## Anwendungsbeispiel - Redundantes Kommunikationsnetzwerk

- Nios II mit MMU (anfangs ohne MMU)
- Ethernet-basiert, optional SHDSL
- Parallel Redundancy Protocol
- Software-Komponenten (COTS und Eigenentwicklung) u.a. für SHDSL, Webserver, Remote Update, SNMP, NTP
- Treiber für eigene IP-Komponenten

1. Distribution von SLS
2. Portierung NOMMU - MMU relativ einfach, nur geringfügige Anpassungen an Treibern (aber nur, weil diese die Linux-Schnittstellen nicht korrekt benutzten)
3. SD Card IP

# Anwendungsbeispiel - Übertragung von HD-Videodaten

- Nios II ohne MMU
- Gigabit-Ethernet
- USB Host- und Device-Controller
- Software-Komponenten (COTS und Eigenentwicklung) u.a. für Session-Setup, Konfiguration, Webserver
- Treiber für eigene IP-Komponenten und Hardware

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InES Institute of Embedded Systems

## Anwendungsbeispiel - Übertragung von HD-Videodaten

- Nios II ohne MMU
- Gigabit-Ethernet
- USB Host- und Device-Controller
- Software-Komponenten (COTS und Eigenentwicklung) u.a. für Session-Setup, Konfiguration, Webserver
- Treiber für eigene IP-Komponenten und Hardware

1. TSE MAC von Altera
2. USB/IP
3. Bestehende Treiber für USB konnten verwendet werden

# Zusammenfassung

- Nios II als flexibel konfigurierbarer Softcore-Prozessor innerhalb eines System-on-a-Chip
- Linux bietet umfangreiche Hardware- und Software-Unterstützung
- Softcore-System und Linux sind nachträglich einfach anpass- und erweiterbar
- Integration von bestehender Linux/Unix-Software mit kleinen oder geringen Anpassungen möglich

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InES Institute of Embedded Systems

# Danke für Ihre Aufmerksamkeit

Tobias Klauser
Institute of Embedded Systems
Zürcher Hochschule für Angewandte Wissenschaften
klto@zhaw.ch

Weitere Informationen zu Nios II Linux:
`http://www.alterawiki.com/wiki/Linux`

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InES Institute of Embedded Systems