

Echtzeitsysteme

Ausblick

Lehrstuhl Informatik 4

07. Februar 2013

Gliederung

1 Sommersemester

- Vorlesungen
- Seminar

2 Aktuelle Forschungsarbeiten

- Aspektorientierte Echtzeitsystemarchitekturen
- Verlässliche eingebettete Systeme: DanceOS und CoRed
- Abschlussarbeiten/Masterprojekte
 - AORTA
 - DanceOS/CoRed

Vorlesung: Verlässliche Echtzeitsysteme

Nicht mehr die Rechtzeitigkeit, sondern die korrekte Funktion steht im Vordergrund

Echtzeitsysteme sind in der Regel sicherheitskritische Systeme

☞ das Wohlergehen von Leib und Leben hat oberste Priorität

↪ Fehlfunktionen sind zu vermeiden

Standards & Normen Was schreibt der Gesetzgeber vor?

- Wie muss die Softwareentwicklung aufgebaut sein?
- Was und wie wird begutachtet begutachtet?

↪ einen „groben Überblick“ vermitteln

Softwareentwicklung ohne „Fehler einzubauen“

- Wie zeigt man die korrekte Funktionsweise von Software?
- ↪ statische Analyse, Model Checking, dynamisches Testen, ...

Fehlertoleranz zur Laufzeit

- Wie geht man mit „Bit-Kippen“ um?

↪ **Fehlertoleranztechniken**

Vorlesung: Verlässliche Echtzeitsysteme (Forts.)

Organisatorisches

Vorlesung

Dozent Fabian Scheler

Wochenstunden 2 Semesterwochenstunden

Raum 01.255-128

Uhrzeit Montag, 12:15 - 13:45, Start: 15.04.2012

Übung

Dozenten Martin Hoffmann,
Florian Franzmann,
Isabella Stalkerich

Wochenstunden 2 + 2 Semesterwochenstunden

Raum und Zeit siehe UnivIS, nach Vereinbarung

Energiegewahre Systemsoftware

Ausgewählte Kapitel der Systemsoftwaretechnik (AKSS)

Gliederung

1 Sommersemester

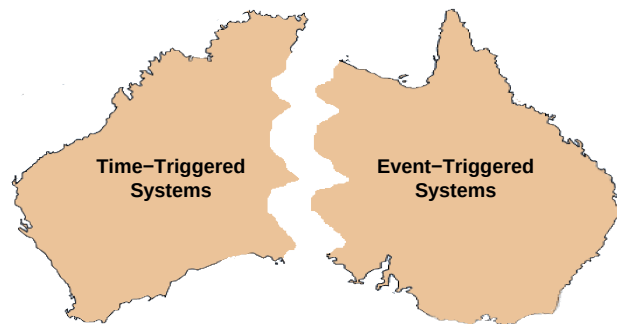
- Vorlesungen
- Seminar

2 Aktuelle Forschungsarbeiten

- Aspektorientierte Echtzeitsystemarchitekturen
- Verlässliche eingebettete Systeme: DanceOS und CoRed
- Abschlussarbeiten/Masterprojekte
 - AORTA
 - DanceOS/CoRed

Aspektorientierte Echtzeitsystemarchitekturen

Zeit- und ereignisgesteuerte Echtzeitsysteme sind grundverschieden



... wenn es um die Implementierung von Abhängigkeiten geht

- **implizit** sichergestellt
 - statische Ablaufplanung
- **explizit** sichergestellt
 - Schlossvariablen, Semaphore
 - Nachrichten
 - ...

Aspektorientierte Echtzeitsystemarchitekturen (Forts.)

☞ Das hat auch Auswirkungen auf die Implementierung von Echtzeitanwendungen!

Fadenabstraktion

- taktgesteuerte Systemen: **einfach Ereignisbehandlungen**
- vorranggesteuerte Systemen: **komplexe Ereignisbehandlungen**

Portabilität

- Fadenabstraktionen sind mit der Anwendung verwoben
- Fäden ...
 - sperren Schlossvariablen
 - versenden Nachrichten
 - warten auf Signale anderer Fäden
- oder laufen einfach nur durch (engl. *run-to-completion*)

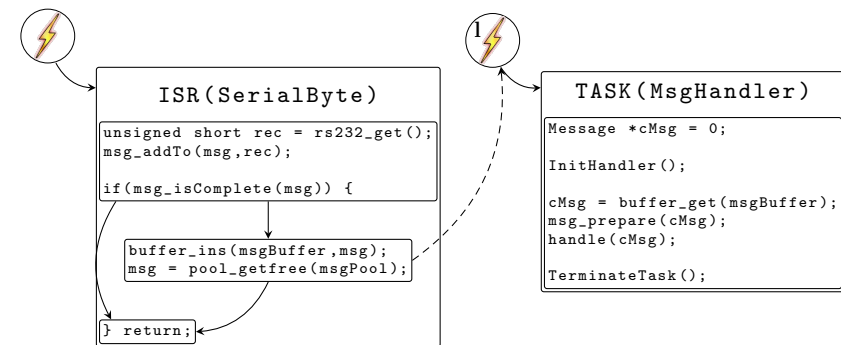
☞ Portierung zwischen Takt-/Vorrangsteuerung ist **sehr schwierig!**

Atomic Basic Blocks (ABBs)

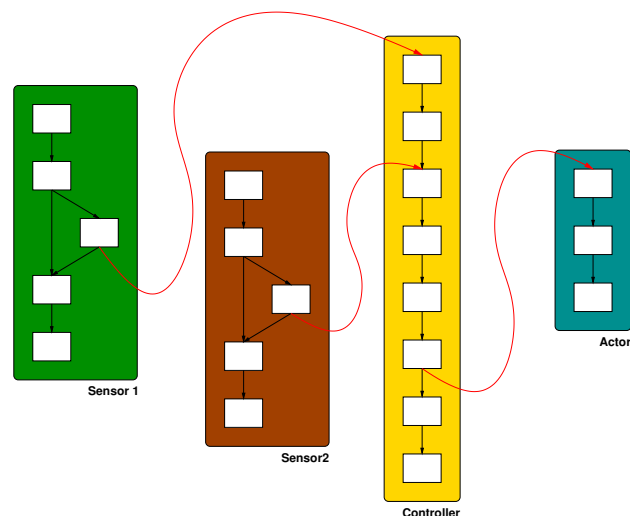
Lösungsidee: Abstrahiere von den Eigenheiten dieser Echtzeitsystemarchitekturen

- stelle Abhängigkeiten **unabhängig** von der Fadenabstraktion dar
- Abbildung auf
 - taktgesteuerte Systeme oder
 - vorranggesteuerte Systeme
- Grundlage: Basisblöcke eines CFGs $\leadsto$ **Atomic Basic Blocks**
 - mehrere Grundblöcke werden zu einem ABB zusammengefasst
 - „Basic Blocks“ ist der erste Namensteil
 - **Abhängigkeiten** verbinden ABBs $\leadsto$ ABB-Graphen
 - Datenabhängigkeiten, gerichtete und ungerichtete Abhängigkeiten
 - prinzipiell mithilfe der Echtzeitsystemarchitektur implementiert
 - ABB-Graphen überspannen **mehrere** Kontrollflüsse
 - im ABB: **keine** Abhängigkeiten zu anderen Kontrollflüssen
 - das macht sie aus Sicht anderer Kontrollflüsse „atomar“
 - $\leadsto$ das ist der zweite Teil des Namens

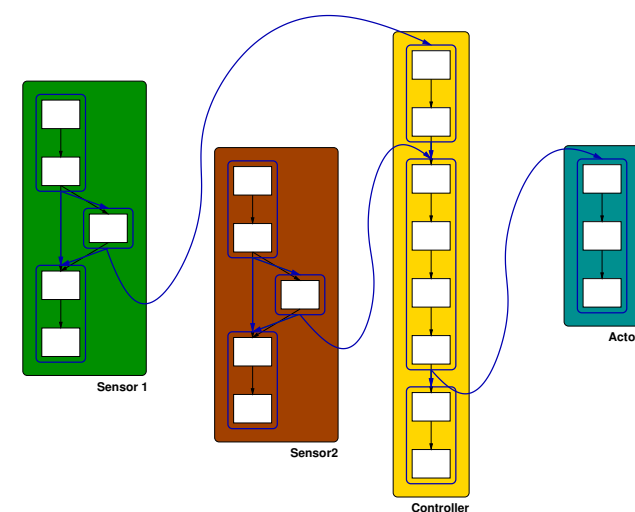
Atomic Basic Blocks (ABBs) — Beispiel



Atomic Basic Blocks (ABBs) — noch ein Beispiel



Atomic Basic Blocks (ABBs) — noch ein Beispiel



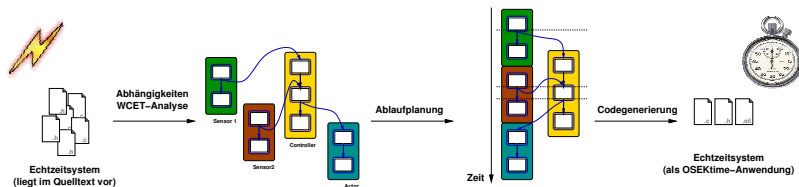
Der Real-Time Systems Compiler (RTSC)

- betriebssystemgewahrer Übersetzer
- vermittelt zwischen zeit- und ereignisgesteuerten Systemen
 - und verwendet dabei ABBs als Zwischendarstellung
- basiert auf der LLVM (Low-Level Virtual Machine)
- Gliederung wie in klassischen Compilern

Front-End Abhängigkeitsgraph erzeugen, WCET-Analyse

Middle-End Transformationen des ABB-Graphen, Ablaufplanung

Back-End Codegenerierung für ein bestimmtes Betriebssystem



Forschungsziele im AORTA-Projekt

Echtzeitbetriebssysteme — Wie sieht die „perfekte“ Schnittstelle aus?

- Semaphore, Mutex, Messages, Flags, Events, ... ~ **riesige Auswahl**
 - jedes Betriebssystem bringt seine ganz spezielle Lösung mit
- Welche Abhängigkeiten implementiert man eigentlich damit?
 - Welche Eigenschaften haben die erzeugten Abhängigkeiten?

Ereignisgesteuerte Systeme als Zielplattform — der Weg zurück

- bisher werden nur zeitgesteuerte Zielsysteme unterstützt

Verteilte Systeme bzw. Mehrkernsysteme statt Monoprozessoren

- Verteilung/Ablaufplanung auf mehreren Rechenknoten/-kernen
- Behandlung des Kommunikationssystems

Optimierung übergeordneter Eigenschaften

- z.B. Blockadezeiten durch blockierende Synchronisation

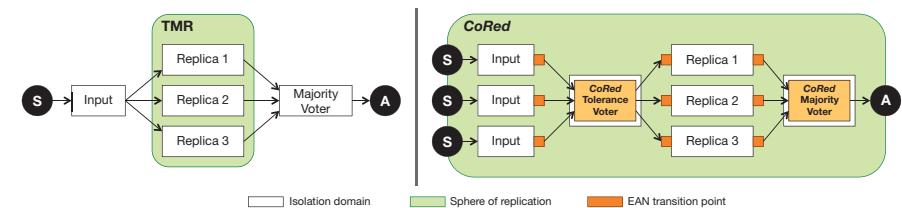
Verlässliche eingebettete Systeme



Problem: Moderne Hardware wird immer unzuverlässiger

- ~ Transiente Hardwarefehler (Bitkipper)
- ~ In sicherheitskritischen Anwendungen nicht tolerierbar

Verlässliche eingebettete Systeme



CoRed

- Absicherung auf der Ebene der Anwendung
- Selektiver Schutz sicherheitskritischer Anwendungsteile

DanceOS

- Absicherung auf der Ebene des Betriebssystems
- Gezielte Absicherung kritischer Datenstrukturen und Operationen

Die „perfekte RTOS-Schnittstelle“

Ausgangspunkt Schnittstellen verschiedener Echtzeitbetriebssysteme

- POSIX (QNX, VxWorks, ...), eCos, Windows CE, AUTOSAR, ...

Fokus: Kontrollflussabstraktion $\leadsto$ Fäden, Unterbrechungen, ...

- Welche Kontrollflussabstraktionen werden angeboten?
- Wie werden sie aktiviert, wie implementieren sie Abhängigkeiten?
 - gerichtete/ungerichtete, synchrone/asynchrone Aufrufsemantik
 - Welche Informationen werden übertragen — Daten, Zeit, Signale

Ergebnis ist eine Studie:

- Welche Abhängigkeiten kann man überall implementieren?
- Gibt es Abhängigkeiten, die nicht implementiert werden können?
- Sind die Schnittstellen orthogonal oder gibt es mehrere Möglichkeiten dieselbe Abhängigkeit zu implementieren?

☞ Untermauernde **Experimente/Beispiele** sind wünschenswert!

RTSC: Next Generation RTSC

- ABBs im RTSC: Aggregation von Basisblöcken

- implementiert als Menge von Zeigern

$\leadsto$ **Problem:** passt nicht so ganz zu Übersetzern

- **Annahme:** jede Analyse kann bei Bedarf neu berechnet werden
 - notwendig Information steckt ja im Quellcode :-)

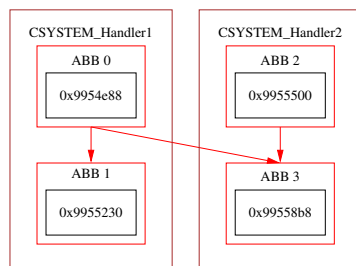
$\leadsto$ bei ABBs ist das aber nicht der Fall :-)

- ABBs müssen immer manuell aktualisiert werden
- zyklische Abfolgen von Transformationen sind nicht möglich
- ...

$\leadsto$ **Lösung:** packe ABBs in den Quelltext :-)

- implementiert als Aufrufe von *Magic Functions*
- *Magic Functions* sind keine echten Funktionen
- werden aber vom RTSC interpretiert

RTSC: Next Generation RTSC (Forts.)



```

def void @CSYSTEM_Handler1() {
    entry:
    call void @RTSC_ABB_begin(0)
    ...
    call void @RTSC_ABB_succ(1)
    call void @RTSC_ABB_succ(3)
    call void @RTSC_ABB_end(0)
    split:
    call void @RTSC_ABB_begin(1)
    ...
    call void @RTSC_ABB_end(1)
    ret void
}
  
```

Zuverlässigkeit nachweisen

Fehlerräumenanalyse

- Welche Bitkipper wirken sich wirklich aus?
- Einschränkung des möglichen Fehlerräumen

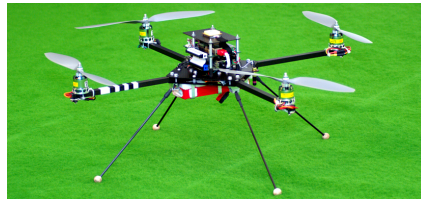
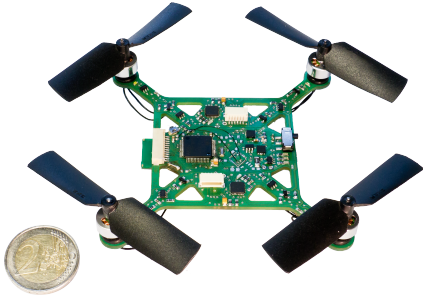
Zuverlässigkeitsanalyse mittels symbolischer Ausführung

- Mathematischer Nachweis ist schwierig und wenig praxisrelevant
- Stattdessen: Nutzung neuer Techniken wie der *symbolischen Ausführung*

Fehlerinjektion im realen/virtuellen System

- Zuverlässigkeitsanalyse hat ihre Grenzen
- Experimentelle Fehlerinjektion

Zuverlässigkeit implementieren



Weitere Themen

- Sind im Umfeld von DanceOS/CoRed/AORTA möglich
- Auch für Projektarbeiten (z.B., Masterprojekt)

Studien-/Diplom-/Bachelor-/Master- ... Doktorarbeiten

Forschungs- und Entwicklungsprojekte: Universität, Forschungseinrichtungen, Industrie

weitere Themen im Internet/UnivIS:

<http://www4.informatik.uni-erlangen.de/Theses/>

