

Echtzeitsysteme (EZS) 2 – aka Echtzeitlabor

Integrierte Lehrveranstaltung, 4 SWS

Ausblick

Echtzeitsysteme - Übungen zur Vorlesung

Fabian Scheler, Peter Ulbrich, Niko Böhm

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
www4.informatik.uni-erlangen.de

8. Februar 2010

Inhalt

- ▶ ein kompletter Entwicklungszyklus für ein EZS
 1. Anforderungsanalyse
 2. Einarbeitung in die Entwicklungsumgebung
 3. Entwicklung der Komponenten
 4. Testen der Komponenten
 5. Komposition - Integrationsphase
 6. Akzeptanztest

Organisation

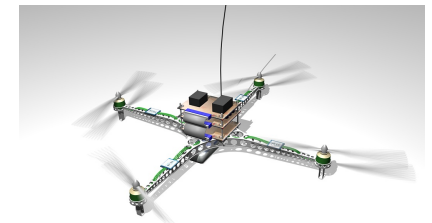
- ▶ Bearbeitung der Experimente erfolgt in 3er-Gruppen
- ▶ (benoteter) Schein
- ▶ **keine** Prüfung!

Entwicklungsumgebung

- Prozessor** ▶ Infineon TriCore 1.3 - TC1796
- Board** ▶ Infineon TriBoard
- Peripherie** ▶ ASC, SSC, GPIO, CAN, ADC, DAC, ...
- Betriebssysteme** ▶ Eigenentwicklungen: CiAO, KESO, (eCos)
▶ Industrie: ProOSEK/time, AUTOSAR OS, eCos, PXROS, FreeRTOS
- Programmiersprachen** ▶ Assembler, C, C++, Java
- Werkzeuge** ▶ SMC (*State Machine Compiler*)
▶ GNU Tools (GCC, Binutils, GDB, make)
▶ WCET-Analyse: aiT
▶ Lauterbach Trace32
▶ Oszilloskop, Funktionsgenerator

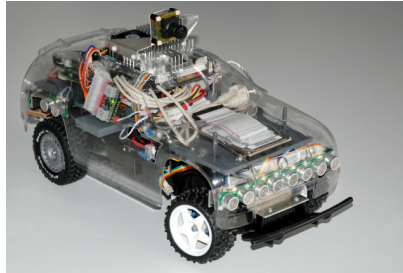
Experiment 1: Quadrocopter

- ▶ Vierrotoriges Fluggerät
 - ▶ Aerodynamisch instabil
 - ▶ Steuerung durch Variation der Drehzahl einzelner Rotoren
- ▶ Regelkreis: Lageregelung
 - ▶ Integraler Bestandteil / zeitkritisch
 - ▶ Physikalische Modell zur Lagebestimmung
 - ▶ Rückkopplung über Beschleunigungssensoren und Gyrometer
- ▶ Weitere Regelkreise
 - ▶ Orientierung Z-Achse (Kompass)
 - ▶ Höhe (Abstands- und Drucksensoren)
- ▶ Steuerung
 - ▶ WLAN oder Fernsteuerung



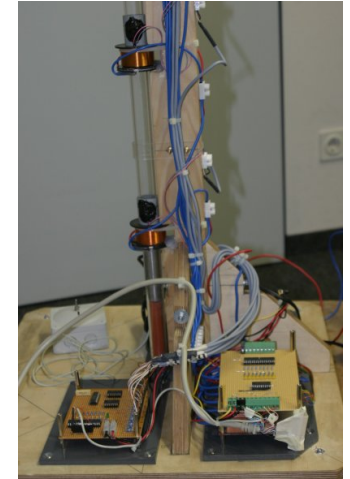
Experiment 2: Carolo-Cup

- ▶ Modellauto
 - ▶ Codierscheiben
 - ▶ Ultraschallsensoren
 - ▶ Infrarotsensoren
 - ▶ Motorsteuerung
 - ▶ Lenkung
- ▶ Fernsteuerung



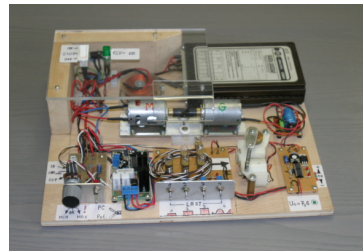
Experiment 3: Hau den Lukas

- ▶ Eisenprojektil in einer Plexiglasröhre
- ▶ wird von Elektromagneten
 - ▶ beschleunigt
 - ▶ gebremst
- ▶ Elektromagneten werden gesteuert
- ▶ Lichtschranken *beobachten* das Projektil
- ▶ verschiedene *Spielarten*
 - ▶ kontinuierlich/schrittweise
 - ▶ anheben/fallen/pendeln
- ▶ mit/ohne Bedienpult



Experiment 4: Generator

- ▶ Regelkreis
 - ▶ Motor treibt Generator an
 - ▶ erzeugte Spannung soll möglichst konstant sein
- ▶ verschiedene Lasten werden zugeschaltet
 - ▶ konstante Lasten
 - ▶ variable Lasten
- ▶ Regler muss die Spannung *nachregeln*
 - ▶ und zwar **rechtzeitig**



AKSS: Cloud Computing

Hauptseminar

Lehrform Hauptseminar 2 SWS

Leistung

- ▶ Vortrag & Ausarbeitung

Vorbesprechung

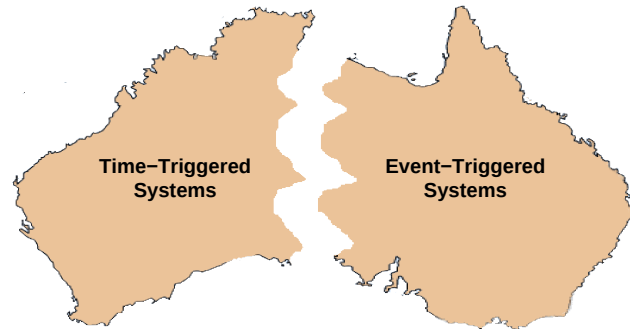
- ▶ Termin: Donnerstag, 11.02.2010, 16:00 - 17:00
- ▶ Raum: 0.035 (Besprechungsraum Informatik 4)

weitere Information

- ▶ http://www4.informatik.uni-erlangen.de/Lehre/SS10/HS_AKSS

Atomic Basic Blocks (ABBs)

Abhängigkeiten in Echtzeitsystemen



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

Atomic Basic Blocks (ABBs)

Folgen

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!**

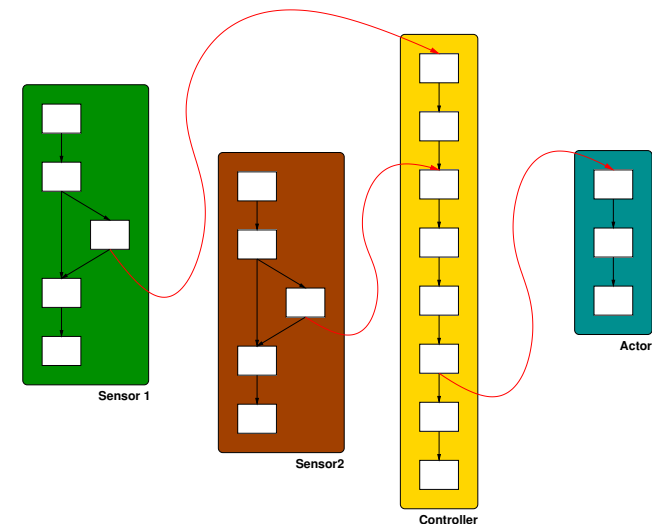
Atomic Basic Blocks (ABBs)

Lösungsidee

- ▶ stelle Abhängigkeiten **unabhängig** von der Fadenabstraktion dar
- ▶ Abbildung auf
 - ▶ taktgesteuerte Systeme oder
 - ▶ vorranggesteuerte Systeme
- ▶ Basisblöcke eines CFGs $\leadsto$ **Atomic Basic Blocks**
 - ▶ **Abhängigkeiten** verbinden ABBs
 - ▶ Kontrollflussgraphen, Datenflussgraphen, gegenseitiger Ausschluss
 - ▶ ABB-Graphen überspannen **mehrere** Kontrollflüsse
 - ▶ in einem ABB: **keine** Abhängigkeiten zu anderen Kontrollflüssen

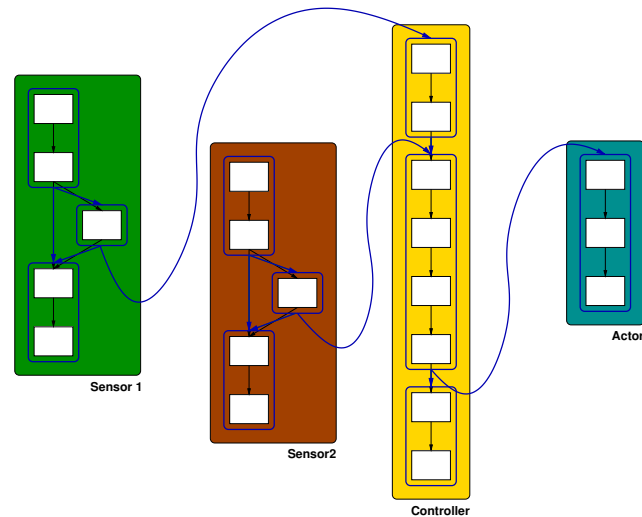
Atomic Basic Blocks (ABBs)

Beispiel



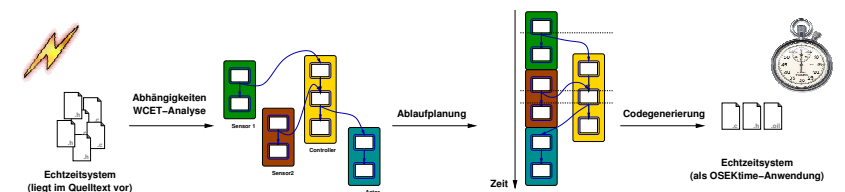
Atomic Basic Blocks (ABBs)

Beispiel

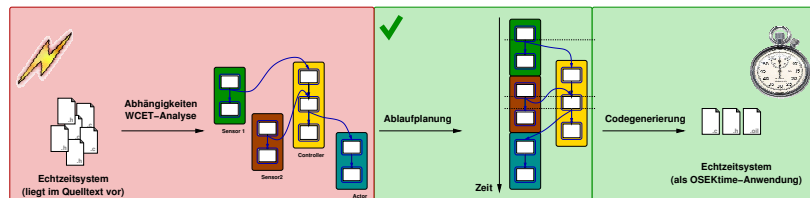


Der Real-Time Systems Compiler (RTSC)

- ▶ 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



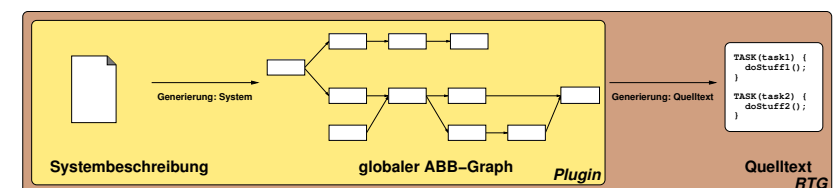
RTSC: OSEK/AUTOSAR OS Front-End



- ▶ existierendes Front-End: synthetische, minimale API
- ▶ Portierung auf AUTOSAR OS
 - ▶ Welche Möglichkeiten für Abhängigkeiten gibt es?
 - ▶ Aufbereitung der Abhängigkeiten für das Middle-End
 - ▶ synthetische API sehr ähnlich zur AUTOSAR OS API

RTSC Testsystem Generator (RTG)

- ▶ **mein Problem:**
 - ▶ um den RTSC zu testen/evaluieren braucht man Anwendungen
 - ▶ reale Anwendungen bekommt man praktisch nicht
 - ▶ es sei denn man entwickelt sie selbst ~ Aufwand :-)
 - ▶ die Industrie ist hier recht wenig kooperativ
 - ▶ **meine Hoffnung:**
 - ▶ eigentlich reichen ja auch **synthetische Anwendungen**
- ~ ein entsprechender **Generator** wäre toll :-)
- ~ der **RTSC Testsystem Generator (RTG)**



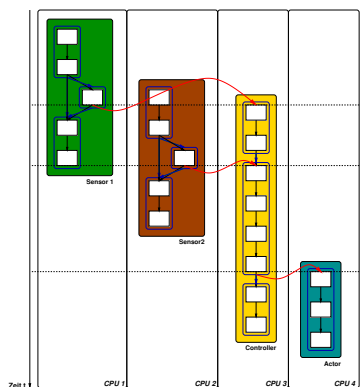
RTSC Testsystem Generator (RTG) (Forts.)

- ▶ Eingabe: **Systembeschreibung**
 - ▶ x % periodische Ereignisse, y % nicht-periodische Ereignisse
 - ▶ maximale Systemauslastung: z %
 - ▶ strukturelle Vorgaben
 - ▶ System enthält zyklische Abhängigkeiten
 - ▶ es gibt Produzenten-Konsumenten-Beziehungen
 - ▶ Verwendung globaler und lokaler Variablen
 - ▶ ...
- ▶ Variante: **Plugin**
 - ▶ wird vom RTSC angerufen
 - ▶ erstellt, einen reinen ABB-Graphen $\leadsto$ Quelltext ist nicht relevant
 - $\leadsto$ RTSC muss an einigen Stellen angepasst werden
 - $\leadsto$ echte Code-Transformationen sind weniger sinnvoll
- ▶ Variante: **RTG**
 - ▶ Erzeugung von Quelltext
 - ▶ **Herausforderung:** Einhaltung der WCET
 - $\leadsto$ **explizite** Einhaltung notwendig

RTSC: Multicore Scheduling

- ▶ Automobilindustrie: 70 - 100 Steuergeräte je Premium-KFZ
 - $\leadsto$ **Problem:**
 - viele Controller $\mapsto$ viele Busse $\mapsto$ viel Kupfer $\mapsto$ viel Gewicht $\mapsto$ **viel Verbrauch**
 - $\leadsto$ **Lösung:** Konsolidierung z.B. durch Mehrkernprozessoren, **aber:**
 - (1) Hardwareanforderungen – I/O
 - (2) Programmierung von Mehrkernprozessoren ist schwierig
- ▶ Idee: zumindest bei (2) könnte der RTSC helfen :-)
- ▶ automatisch Abbildung von ABB-Graphen auf
 - ▶ Mehrkernsysteme und
 - ▶ verteilte Systeme
- ▶ Implementierung eines existierenden Algorithmus
 - ▶ Peng, Shin und Abdelzaher (1997)
 - ▶ statische Allokation von *Modulen* ($\approx$ ABBs) auf Rechenknoten

RTSC: Multicore Scheduling (Forts.)

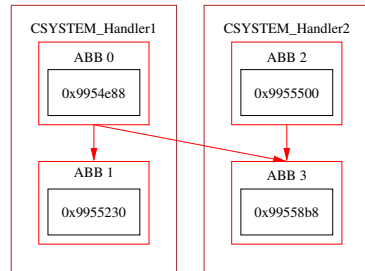


- ▶ Abbildung auf verschiedene Knoten
 - ▶ Annahme einer globalen Zeitbasis
- ▶ Berücksichtigung von Kommunikation
 - ▶ über gemeinsamen Speicher
 - ▶ gleicher Knoten $\leadsto$ Variablen
 - ▶ über Nachrichten
 - ▶ entfernter Knoten $\leadsto$ TTCAN
- ▶ Unterstützung von TTCAN
 - ▶ frühere Studienarbeit

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
}
  
```

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

Forschungs- und Entwicklungsprojekte: Universität, Forschungseinrichtungen, Industrie

weitere Themen im Internet/UnivIS:

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

