

# ***Compilerbau***

Wintersemester 2009 / 2010

Dr. Heiko Falk


Technische Universität Dortmund

Lehrstuhl Informatik 12

Entwurfsautomatisierung für Eingebettete Systeme

# Organisatorisches (1)

## Vorlesung (2V)

- Dienstags, 13:00 – 14:30 Uhr  
INF/E009
- Dienstags, 14:50 – 16:20 Uhr  
INF/E009
- Alle zwei Wochen! Konkrete Termine:  Folie 4

## Materialien

- Foliensätze online:  
<http://ls12-www.cs.tu-dortmund.de/teaching/courses/ws0910/cb>
- Bücher & Originalartikel

# Organisatorisches (2)

## Wünschenswerte Voraussetzungen

- Programmiersprache C, Assembler
- Rechnerarchitektur
- Endliche Automaten, Kellermaschinen

## Prüfungen

- Schriftliche Klausur von 90 Minuten Länge
- *Datum, Zeit & Ort werden noch bekannt gegeben*

## Closed Notebook Policy

- Private Laptops bleiben in der Vorlesung bitte geschlossen
- ☞ *Deutliche Verbesserung der Arbeitsatmosphäre!*

# Termine

## **Dienstags, 13.00 – 14.30 und 14.50 – 16.20 Uhr**

- 13. Oktober 2009
- 27. Oktober 2009
- 10. November 2009
- 24. November 2009
- 8. Dezember 2009
- 5. Januar 2010
- 19. Januar 2010
- 2. Februar 2010 *(nur von 13.00 – 14.30 Uhr!)*

# ***Kapitel 0***

## ***Einordnung & Motivation der Vorlesung***

# Worum geht's beim Compilerbau?



## Definition laut Wikipedia:

*„Ein **Compiler** [...] ist ein Computerprogramm, das ein in einer Quellsprache geschriebenes Programm – genannt Quellprogramm – in ein semantisch äquivalentes Programm einer Zielsprache (Zielprogramm) umwandelt. Üblicherweise handelt es sich dabei um die Übersetzung eines von einem Programmierer in einer Programmiersprache geschriebenen Quelltextes in Assemblersprache, Bytecode oder Maschinensprache.“*

*„Der **Compilerbau**, also die Programmierung eines Compilers, ist eine eigenständige Disziplin innerhalb der Informatik. Er gilt als das älteste Gebiet der praktischen Informatik.“*

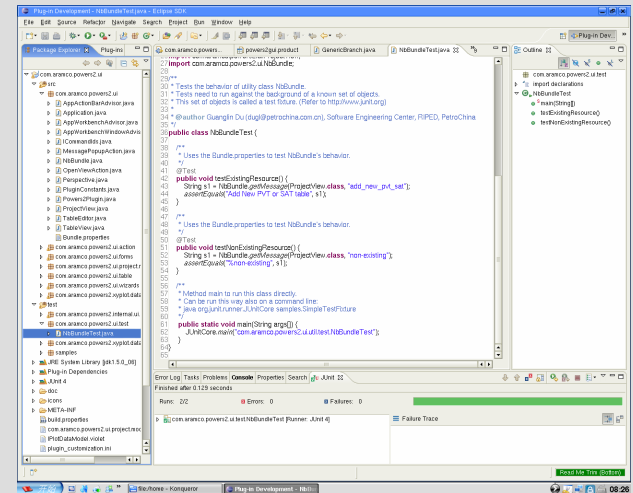
**Die gute Nachricht:** Compilerbau ist eine Disziplin der praktischen Informatik!



# Nutzung von Compiler-Technologie (1)

## Beispiel: Übersetzung von Programmiersprachen

- Programmierung heute zumeist in objektorientierter Hochsprache (z.B. Java, C++).
- Programmierung geschieht üblicherweise in komfortablen Entwicklungsumgebungen (z.B. eclipse)
- Hochsprachliche Programme müssen in ausführbaren Code übersetzt werden (Java: Bytecode, C++: Maschinencode).
- Compiler wird i.d.R. durch einfachen Knopfdruck in der Entwicklungsumgebung aufgerufen.



# Nutzung von Compiler-Technologie (2)

## Beispiel: Interpretierung von Programmiersprachen

- Viele kleine „Hilfsprogrammchen“ werden oft nicht wie auf voriger Folie beschrieben programmiert.
- Statt dessen: Ad-hoc Realisierung mit Skriptsprachen (z.B. sh, perl).
- Solche Skripte werden zur Laufzeit interpretiert, d.h. Befehl für Befehl in Maschineninstruktionen übersetzt, die dann unmittelbar ausgeführt werden.
- Zusätzlich: Auch klassische Programmiersprachen, die keine Skriptsprachen sind, beruhen oft auf Interpretern (z.B. Java, dessen vor-compilierter Bytecode zur Laufzeit interpretiert wird).

```

#!/bin/bash

set -u
trap hook_exit HUP QUIT KILL TERM
trap 'EXIT_CODE=1; hook_exit' INT

if [ "$#" -eq "0" ]; then
    echo "Can not be invoked directly."
    exit 1
fi

source test.lib
  
```



# Nutzung von Compiler-Technologie (3)

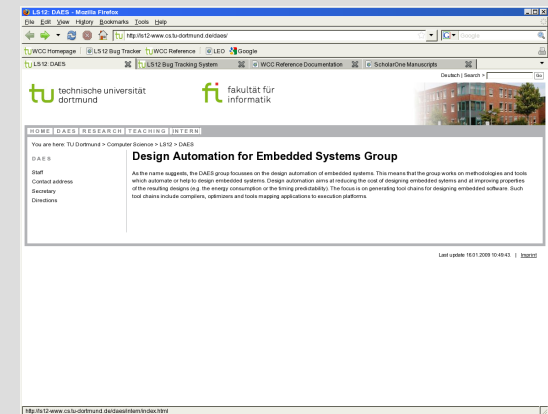
## Beispiel: Web-Browser

- Web-Server liefern an Browser HTML-Dateien aus, die eine Spezialisierung der XML-Notation sind.
- Ein Browser muss HTML-Dateien analysieren (parsen) und in eine grafische Darstellung im Browser-Fenster übersetzen.

```

Source of: http://is12-www.cs.tu-dortmund.de/daes/ - Mozilla Firefox
File Edit View Help
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="content-type" content="text/html, charset=ISO-8859-1">
<meta name="keywords" content="" />
<meta name="description" content="" />
<meta name="http-equiv" content="" />
<meta name="robots" content="index, follow">
<meta name="revisit-after" content="1 days">

<link rel="stylesheet" href="/stylesheets/screen.css" type="text/css"
media="screen" title="Styleheed" />
<link rel="stylesheet" href="/stylesheets/print.css" type="text/css"
media="print" />
<link rel="stylesheet" href="/stylesheets/opera.css" type="
"text/Opera.charset=utf-8" media="screen" title="opera" />
<link rel="stylesheet" href="/stylesheets/custom_links.css" type="
"text/css" media="screen" /><!--[[[ IE]]]-->
<link rel="stylesheet" href="/stylesheets/ie.css" type="text/css" media="screen" t
<link rel="stylesheet" href="/stylesheets/custom_links_ie.css" type="text/css" med
<!--[[[ end]]]-->
<link rel="SHORTCUT ICON" href="/favicon.ico" type="image/x-icon">
<link rel="stylesheet" href="/stylesheets/extended.css" type="text/css"
media="screen"/>
    
```



# Nutzung von Compiler-Technologie (4)

## Beispiel: Professioneller Textsatz

- Wissenschaftliche Texte werden bevorzugt in LaTeX gesetzt, einer „Programmiersprache“ für Manuskripte in Buchdruckqualität.
- LaTeX-Kommandos, die Struktur und Format des Dokuments beschreiben, werden in TeX-Befehle übersetzt, die den Textsatz beschreiben.
- Aus den TeX-Befehlen wird dann eine lesbare Ausgabe z.B. in Postscript- oder PDF-Format erzeugt.

```

\documentclass{svjour3}

\smartqed
\usepackage{array,epsfig,graphicx}
\journalname{Springer Real-Time Sys.}

\begin{document}
  \title{A Compiler Framework for...}
  \author{H. Falk \and ...}

  ...
\end{document}
  
```

A Compiler Framework for the Minimization of Worst-Case Execution Times

Heiko Falk · Paul Lokuciejewski

Received: date / Accepted: date

**Abstract** The current practice in designing software for real-time systems is tedious. There is almost no tool support assisting the designer in automatically deriving safe bounds of the *worst-case execution time (WCET)* of a system during code generation and in systematically optimizing code in order to minimize WCETs.

This article presents concepts and infrastructures for WCET-aware code generation and optimization techniques for WCET minimization. All together, they help in obtaining code explicitly optimized for its worst-case timing, in automating large parts of the real-time software design flow, and in reducing costs of a real-time system by allowing to use tailored hardware.

**Keywords** Real-Time, WCET, Compiler, Code Generation, Optimization

# Nutzung von Compiler-Technologie (5)

## Beispiel: Verwendung von Compiler-Analysen

- Bevor ein Compiler eine Übersetzung durchführt, muss er die zu übersetzenden Daten analysieren (z.B. auf Syntax, Struktur oder Semantik).
- ☞ Compiler bestehen daher aus einer *Analysephase* gefolgt von der *Synthesephase*, die die eigentliche Übersetzung durchführt.
- Techniken der Analysephase findet man überall dort wieder, wo Daten eingelesen und strukturiert zur weiteren Verarbeitung in Datenstrukturen repräsentiert werden müssen.
- Beispiele aus dem Bereich Eingebetteter Systeme:
  - MP3-Player: Nutzen Meta-Informationen in ID3-Tags
  - Digitale Kameras: Nutzen Meta-Informationen in EXIF-Tags

# Stand der Forschung im Compilerbau (1)

## Analysephase:

- Eines der am besten erforschten Gebiete der Informatik, mit sehr starken theoretischen Grundlagen.
- Die Praxis zeigt, dass alle gängigen Programmiersprachen in die Klasse der sog. LR(1)-Sprachen fallen (☞ *Kapitel 4*).
- Die starke theoretische Basis für LR(1)-Sprachen ist in Werkzeuge gemündet (z.B. FLEX, BISON), die dem Compilerbauer das Realisieren von LR(1)-Analysatoren sehr leicht machen.
- ☞ Aktuelle Forschungsaktivitäten im Bereich der Analyse (*meiner Meinung nach*) recht begrenzt.

# Stand der Forschung im Compilerbau (2)

## Synthesephase:

- Programmsynthese hängt stark von der Prozessorarchitektur ab.
  - In letzten Jahrzehnten: Prozessoren sind immer komplexer, mächtiger und auf Spezialzwecke ausgerichtet geworden.
  - Moderne Befehlssätze sind für Spezialzwecke erweitert worden, bspw. um Gleitkomma-Befehle oder zur Beschleunigung von Multimedia-Anwendungen (z.B. Intel's MMX). (☞ Kapitel 1)
  - Compiler-Synthese muss solche spezialisierten und hocheffizienten Befehle ausnutzen und erzeugen können! Gerade im Bereich *Eingebetteter Systeme* ist dies extrem wichtig.
- ☞ *Vielfältige Forschungsaktivitäten auf dem Gebiet der Synthese; sehr lebendiger, aktiver und spannender Bereich der Informatik!*

# Eingebettete Systeme

**Definition:** Eingebettete Systeme (ES) sind

- informationsverarbeitende Systeme,
- die in ein größeres Produkt eingebettet sind.

- Informationsverarbeitung Eingebetteter Systeme nicht ausschlaggebend für Kauf
- Statt dessen: Nutzen des übergeordneten Produkts beeinflusst Kaufentscheidung

[P. Marwedel, *Eingebettete Systeme*, Springer, 2007]

# Anwendungsbereiche Eingebetteter Systeme

- **Konsumgüter**



- **Multimedia**



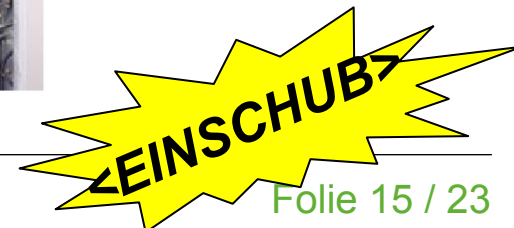
- **Transportmittel**



- **Telekommunikation**



- **Gebäudeautomation, Robotik, ...**



# Bedeutung Eingebetteter Systeme

- **Smartphones** 113 Mio Geräte 2007 → 25,6% Steigerung p. A.  
365 Mio Geräte 2012
- **UMTS** 402 Mio Kunden weltweit 2008  
30 Mio Neukunden pro Quartal
- **Energieverbrauch mobiler Breitband-Infrastruktur**  
42,8 Mrd KWh 2005 → 124,4 Mrd KWh 2011
- **Breitband-Internet**  
567 Mio Kunden 2011 → 100% Steigerung i. Vgl. z. 2007
- **US Konsum-Elektronik**  
Ø Haushalt: 25 Geräte, Ø Erwachsener: 1.200\$ p.A.

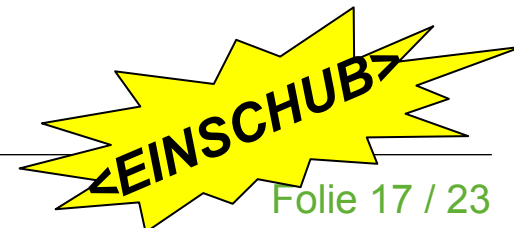
[[www.itfacts.biz](http://www.itfacts.biz)]



# Anforderungen an Eingebettete Systeme (1)

## ■ Effizienz

- Laufzeit-Effizienz
- Energieverbrauch
- Codegröße
- Physikalische Größe / Gewicht
- Kosten



# Anforderungen an Eingebettete Systeme (2)

## ■ Realzeit-Fähigkeit

Für Eingabe  $x$  berechne ein System  $f(x)$ .

- Ein ***Nicht-Realzeit-System*** heißt korrekt, wenn  $f(x)$  korrekt berechnet wird.
- Ein ***Realzeit-System*** heißt korrekt, wenn zusätzlich  $f(x)$  innerhalb von außen vorgegebener Zeit berechnet wird.

☞ Eine zu späte Berechnung von  $f(x)$  durch ein Realzeit-System ist gleich einer falschen Berechnung.

## Anforderungen an Eingebettete Systeme (3)

### **„Hartes“ Realzeit-System:**

Zu späte Berechnung von  $f(x) \rightarrow$  Katastrophe  
(Verlust menschlichen Lebens, Umweltschäden, ...)

#### *Beispiel Airbag-Steuerung:*

Befehl zum Zünden der Airbags: 15ms

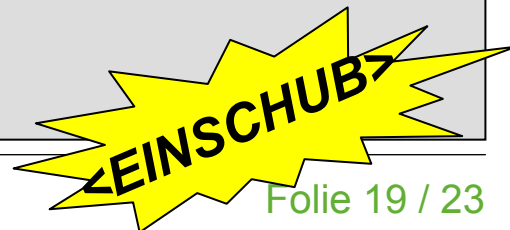
Zu späte Entscheidung: Verletzungsgefahr für Insassen  
und/oder Retter. Daher: Airbags nicht zünden

### **„Weiches“ Realzeit-System:** Keine katastrophalen Folgen

#### *Beispiel DVD-Player:*

Zu späte Frame-Dekodierung: Frame-Drop

Unschön, aber (i.d.R.) nicht katastrophal



# Anforderungen an Eingebettete Systeme (4)

## ■ Zuverlässigkeit / Sicherheit

- Lebensdauer Eingebetteter Systeme: Einige Jahre
- Während der gesamten Lebensdauer: Keine Ausfälle

*Beispiel Drosselklappen-Steuerung:*

Produktionsvolumen: 2 Mio. Einheiten pro Jahr

Erlaubte Fehlerquote: 1 Einheit pro Jahr

## ■ Wartbarkeit, (begrenzte Erweiterbarkeit)

- Fehlersuche, Diagnose, Rekonfiguration zur Laufzeit, ...

## ■ Unterstützende Entwurfswerkzeuge ( $\rightarrow$ Time-to-Market)

- Spezifikation, Synthese, Code-Generierung, ...

# Vorlesung „Compilerbau“

## Über die Vorlesung:

- Praxisnahe und algorithmische Einführung in alle Teilbereiche des Compilerbaus.
- Im Gegensatz zu klassischen Compilerbau-Vorlesungen: Fokus nicht nur auf Analysephase.
- Vielmehr: Behandlung forschungsnaher Fragen aus dem Bereich der Synthesephase, insbes. Instruktionsauswahl, Register-Allokation und Code-Optimierung.
- Wegen der enormen Relevanz Eingebetteter Systeme werden aktuelle Forschungsfragen in der Vorlesung anhand von Problemen aus dem Bereich Eingebetteter Systeme motiviert.
- *Schwerpunkt*: Optimierungen während der Synthesephase.

# Gliederung der Vorlesung

- **Kapitel 1: Compiler – Abhängigkeiten und Anforderungen**
- **Kapitel 2: Interner Aufbau von Compilern**
- **Kapitel 3: Lexikalische Analyse (Scanner)**
- **Kapitel 4: Syntaktische Analyse (Parser)**
- **Kapitel 5: Semantische Analyse**
- **Kapitel 6: Instruktionsauswahl**
- **Kapitel 7: Register-Allokation**
- **Kapitel 8: Code-Optimierung**
- **Kapitel 9: Ausblick**

# Allgemeine Literatur

## Compilerbau

- Reinhard Wilhelm, Dieter Maurer, *Übersetzerbau*, 2. Auflage, Springer, 1997.  
ISBN 3-540-61692-6
- Steven S. Muchnick, *Advanced Compiler Design & Implementation*, Morgan Kaufmann, 1997.  
ISBN 1-55860-320-4
- Andrew W. Appel, *Modern compiler implementation in C*, Cambridge University Press, 1998.  
ISBN 0-521-58390-X

## Eingebettete Systeme

- Peter Marwedel, *Eingebettete Systeme*, Springer, 2007.  
ISBN 978-3-540-34048-5