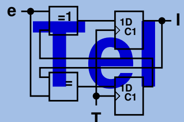
Implementierung eines compilierenden Prozessorsimulators für die Architekturbeschreibungssprache TADL

Diplomverteidigung

Frank Sakowski

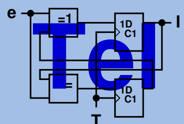
frank.sakowski@mail.inf.tu-dresden.de

Technische Universität Dresden
Institut für Technische Informatik



Gliederung

- Einleitung
- Motivation
- Vorbetrachtung
  - Begriffe, Stand der Technik, Auswahl geeigneter Methoden, Implementierungssprache
- Implementierung
  - Separierung des Simulatorkerns, Vorverarbeitung der Operationen
- Simulationsgeschwindigkeit
  - Ergebnisse, Auswertung
- Zusammenfassung
- Ausblick

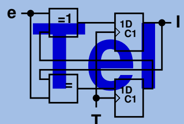


Einleitung

- Steigende Transistorendichte
- Wachsende Integration von elektronischen Systemen
- großer Entwicklungsaufwand $\Rightarrow$ Gefahr für Fehlerquellen
- Simulatoren zur Senkung von Entwurfs-/Produktionsrisiken
- bilden System funktionell ab
- Hardware-/Softwaresimulatoren
- Prozessorsimulator prosim und weitere Werkzeuge als Grundlage
- Ziel: Steigerung der Simulationsgeschwindigkeit

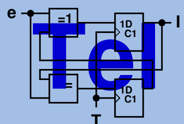
Motivation

- Geschwindigkeitsvorteil ⇒ Marktvorteil
- Simulationsgeschwindigkeit begrenzt
- Wirtsrechner – von kurzer Dauer
- Umsetzung des Simulators – zukunftsweisender, auf Zeit preisgünstiger
- Überarbeitung prosim



Begriffe 1

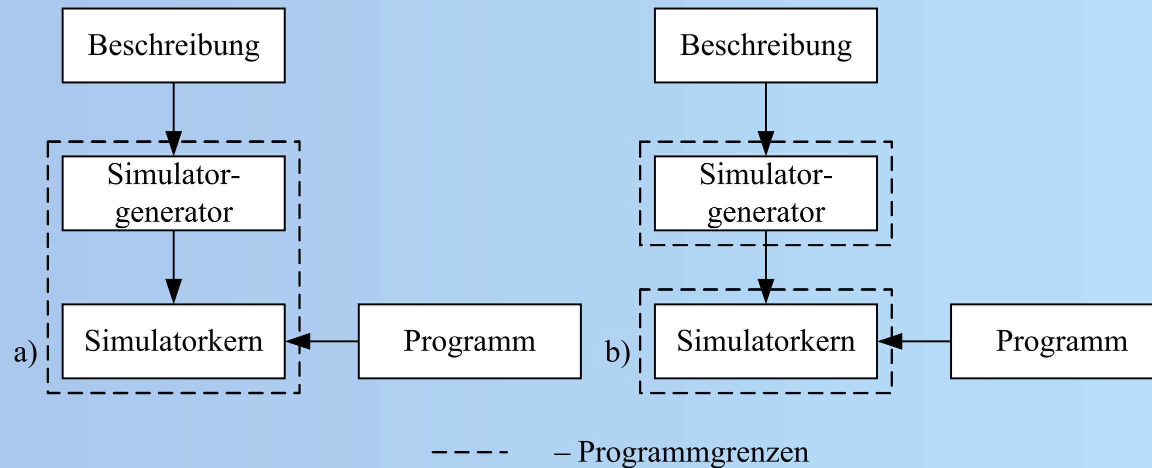
- zeitliche Abstraktion
  - befehlsgenau
  - zyklusgenau
  - phasengenau
- strukturelle Abstraktion
  - Funktion
  - Befehlssatz
  - Pipeline
  - Struktur
  - Mikroarchitektur



Frank Sakowski
Technische Universität Dresden
Institut für Technische Informatik
frank.sakowski@mail.inf.tu-dresden.de

Begriffe 2

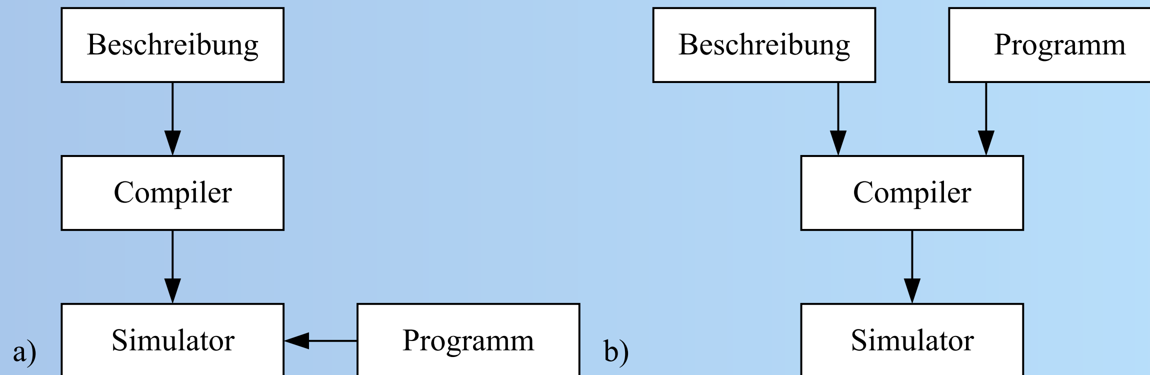
- architekturinterpretierend/-compilierend



Frank Sakowski
Technische Universität Dresden
Institut für Technische Informatik
frank.sakowski@mail.inf.tu-dresden.de

Begriffe 3

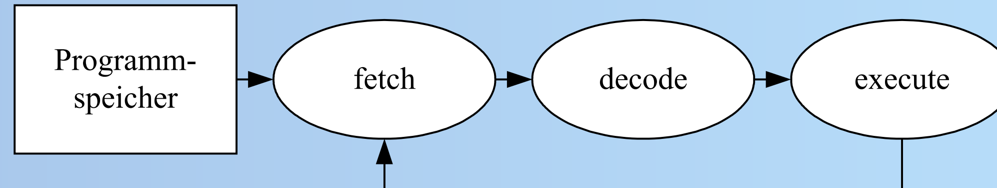
- programminterpretierend/-compilierend



Frank Sakowski
Technische Universität Dresden
Institut für Technische Informatik
frank.sakowski@mail.inf.tu-dresden.de

Stand der Technik 1

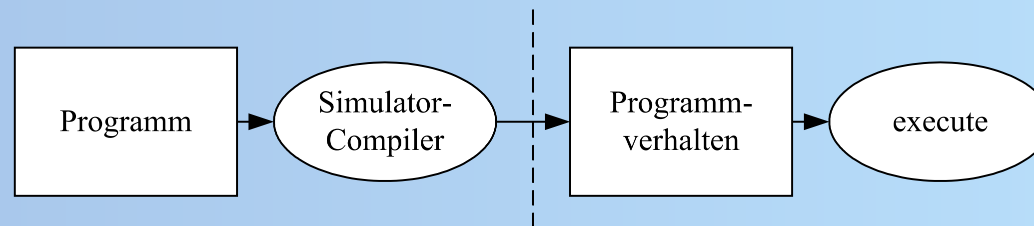
- Simulator Teil einer Entwicklungsumgebung
- mittels Architekturbeschreibungssprache retargierbar
- alle architekturcompilierend
- programminterpretierend am häufigsten, hohe Genauigkeit, flexibel



Frank Sakowski
Technische Universität Dresden
Institut für Technische Informatik
frank.sakowski@mail.inf.tu-dresden.de

Stand der Technik 2

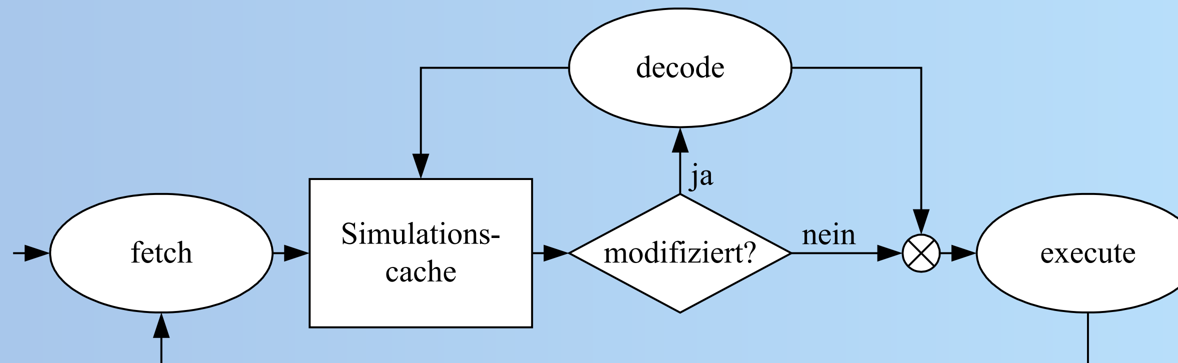
- programmcompilierend zwei/drei Größenordnungen schneller, kein dynamischer Code, viele nur befehlsgenau



Frank Sakowski
Technische Universität Dresden
Institut für Technische Informatik
frank.sakowski@mail.inf.tu-dresden.de

Stand der Technik 3

- hybride Ansätze, Simulationscache

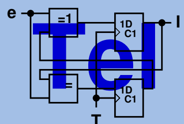


- Optimierung Compilierungszeit, Speicherverbrauch
- Registerprojektion

Frank Sakowski
Technische Universität Dresden
Institut für Technische Informatik
frank.sakowski@mail.inf.tu-dresden.de

Auswahl geeigneter Methoden

- architekturcompilierend – zur Beseitigung von Programmlast
- programmcompilierend – zu abstrakt
- Vorverarbeitung – interessant bei Emulation
- Simulationscache – zu evaluieren
- Registerprojektion – zu aufwändig
- Senkung der Abstraktion – nicht Bestandteil der Aufgabe



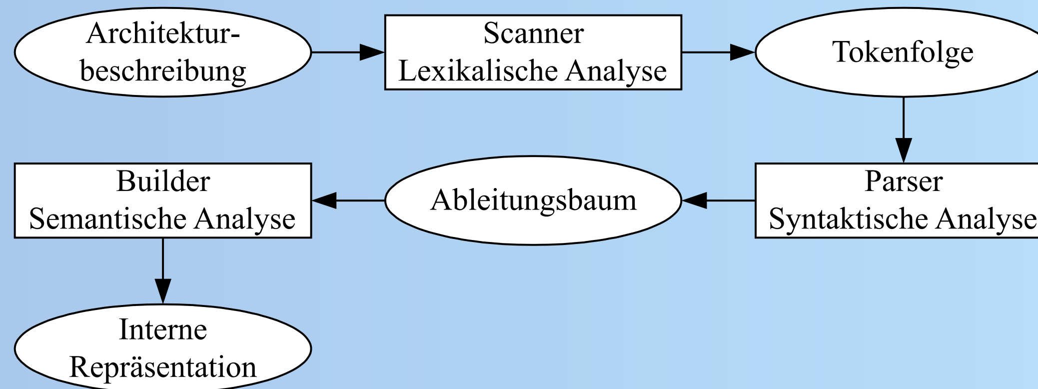
Frank Sakowski
Technische Universität Dresden
Institut für Technische Informatik
frank.sakowski@mail.inf.tu-dresden.de

Implementierungssprache

- C++ – vorgegeben, weite Verbreitung, modular erweiterbar, gute Unterstützung durch Standardwerkzeuge
- SystemC – empfohlen, zusätzliche Datentypen, Takt, Ereignissteuerung
- prosim hat jene Elemente bereits implementiert
- komplette Überarbeitung von prosim ⇒ keine Kompatibilität

Separierung des Simulatorkerns 1

- Generator von prosim



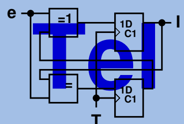
- Builder besteht aus ArchBuilder, ArchInstrBuilder und ArchOpBuilder

Frank Sakowski
Technische Universität Dresden
Institut für Technische Informatik
frank.sakowski@mail.inf.tu-dresden.de

Separierung des Simulatorkerns 2

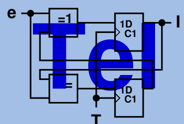
- modifizierter Builder erzeugt Code für Simulatorkern und stößt Compilierung an

```
$ ./csim -a xyz.arch
  setzt compiledsimulation auf true;
  übergibt Architekturbeschreibung an Simulator
    übergibt Architekturbeschreibung an ArchBuilder
      stellt Programmcode zusammen
      File-Streams: csim -> csim.cpp
                    csimTmp -> zweiter Teil von csim.cpp
                    cmake -> arch/makefile.inc
      schreibt Code in Streams
      übergibt Architekturbeschreibung an Parser
        loadLib(), buildBus(), buildRegFile()
          csimTmp
```



Separierung des Simulatorkerns 3

```
{
  //=================================
  // Build register file "PC"

  OwningRef<RegFile> reg(new RegFile(*arch, Ident(arch->strings, "PC"), 1, 32,
    1, 1, 0));

  {
    //=================================
    // Register aliases

    arch->aliaser.define("PC0", Location(reg.get(), BitVector(sizeof(unsigned)
      << 3, "0")));
    arch->aliaser.define("PC", Location(reg.get(), BitVector(4, "0")));
  }

  //=================================
  // Hold on to the regfile

  arch->clock.addListener(reg.get());
  arch->clock.addEventSource(reg.get());
  arch->regfiles.add(reg);
}
```



Frank Sakowski
Technische Universität Dresden
Institut für Technische Informatik
frank.sakowski@mail.inf.tu-dresden.de

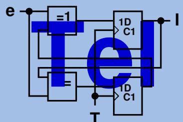
Separierung des Simulatorkerns 4

```
  buildPipe()
    csim, csimTmp, cmake
    Streams: hFile -> xyzPipeStage.hpp
             cFile -> xyzPipeStage.cpp
             cFileTmp -> zweiter Teil von xyzPipeStage.cpp
    übergibt Operation an ArchOpBuilder
    hängt cFileTmp an cFile
  buildIsa()
    csimTmp
    übergibt Befehlsliste an ArchInstrBuilder
      buildInstruction() -> !Instructor - Zeiger statt Objekte
        csim, csimTmp, cmake
        Streams: hFile -> xyzInstruction.hpp
                 cFile -> xyzInstruction.cpp
                 cFileTmp -> zweiter Teil von xyzInstruction.cpp
        übergibt Operation an ArchOpBuilder
        hängt cFileTmp an cFile
hängt csimTmp an csim
compiliert Simulatorkern
```

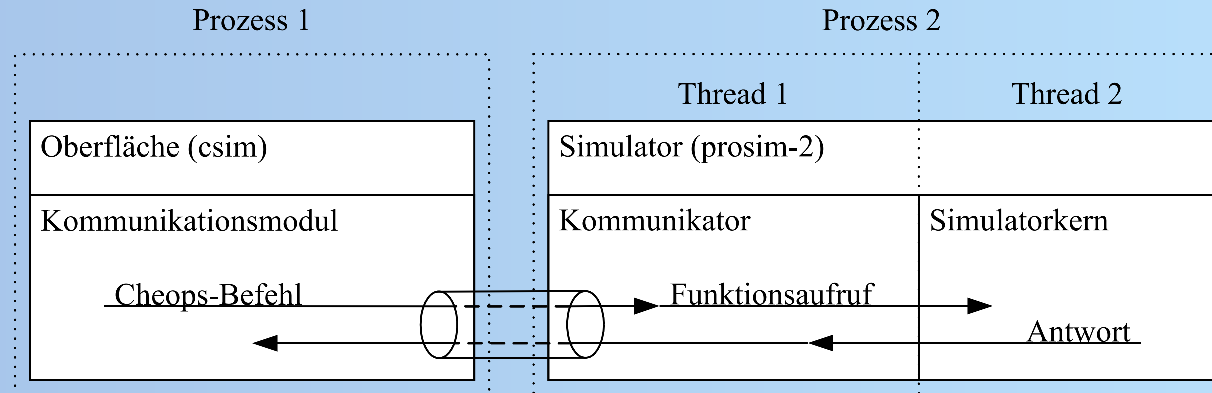
Separierung des Simulatorkerns 5

- Kern im eigenen Prozess
- Kommunikation mittels cheops
- Steuerung durch NDO nicht möglich ⇒ angepasste Testoberfläche

```
erzeugt neuen Prozess
  startet Simulatorkern im Kindprozess
    initialisiert cheops-Server
    initialisiert Architektur -> !main() friend von Arch
    wartet auf Verbindung
    startet Kommunikator -> !Communicator angepasst
    übergibt Architektur an Kommunikator
      übergibt Architektur an Simulator -> !Simulator angepasst
    verknüpft Kommunikator mit Socket
  startet Benutzeroberfläche im Elternprozess
    initialisiert cheops-Client
    nimmt Verbindung auf
    fungiert als Kommunikationsmodul
```

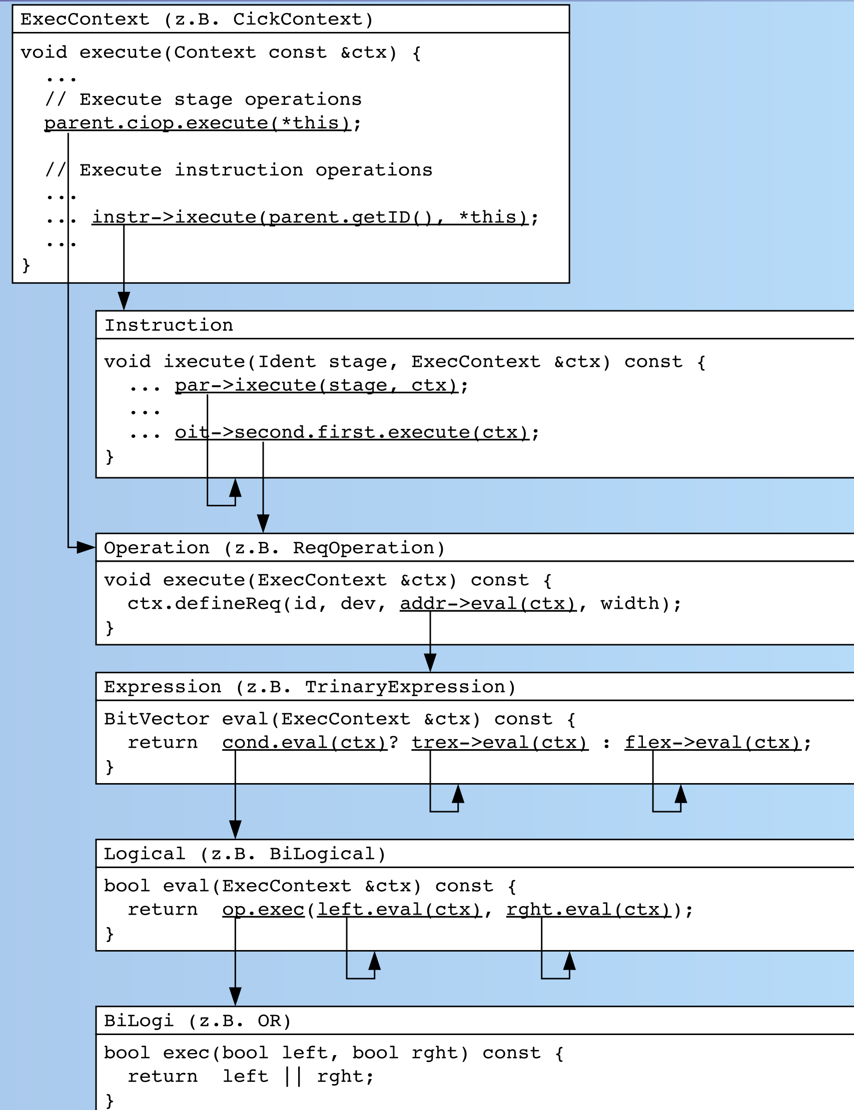


Separierung des Simulatorkerns 6



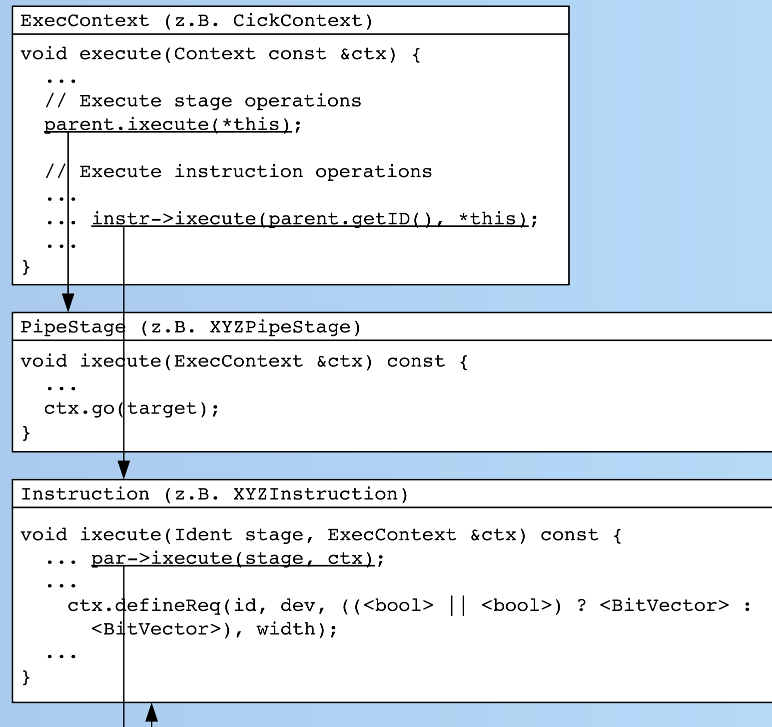
Frank Sakowski
Technische Universität Dresden
Institut für Technische Informatik
frank.sakowski@mail.inf.tu-dresden.de

Vorverarbeitung der Operationen 1

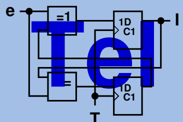


Frank Sakowski
Technische Universität Dresden
Institut für Technische Informatik
frank.sakowski@mail.inf.tu-dresden.de

Vorverarbeitung der Operationen 2



- Einsparung an Funktionsaufrufen bei Ausführung

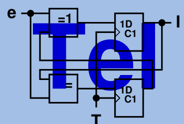


Frank Sakowski
Technische Universität Dresden
Institut für Technische Informatik
frank.sakowski@mail.inf.tu-dresden.de

Vorverarbeitung der Operationen 3

- keine expliziten Operationen, arithmetischen und logischen Ausdrücke
- Ausführungsfunktionen für jeden Befehl/jede Pipelinestufe individuell erstellt

```
ArchOpBuilder erhält Operation
  buildXyzOperation()
    hFile, cFile, cFileTmp
    übergibt Operationen an ArchOpBuilder
    übergibt arithmetische Ausdrücke an MyExprBuilder
      buildXyzExpression()
        hFile, cFile, cFileTmp
        übergibt arithmetische Ausdrücke an MyExprBuilder
        übergibt logische Ausdrücke an MyLogiBuilder
    übergibt logische Ausdrücke an MyLogiBuilder
      buildXyzLogical()
        hFile, cFile, cFileTmp
        übergibt arithmetische Ausdrücke an MyExprBuilder
        übergibt logische Ausdrücke an MyLogiBuilder
```



Frank Sakowski
Technische Universität Dresden
Institut für Technische Informatik
frank.sakowski@mail.inf.tu-dresden.de

Vorverarbeitung der Operationen 4

```
buildRangeExpression() -> !rangeExpressionEval() in Instruction/PipeStage
buildSplitOperation() -> !ExecContext/PipeStage angepasst
```

Frank Sakowski
Technische Universität Dresden
Institut für Technische Informatik
frank.sakowski@mail.inf.tu-dresden.de

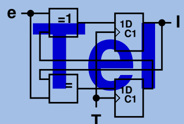
Vorverarbeitung der Operationen 5

- initialisieren der Variablen durch Konstruktor
- Problem: Zeiger auf Pipelinestufen ⇒ Laufzeitfehler

```
... ID_<name>_<nr>(accs.strings(), <name>), ... {
                                     -> !Accessible in Instruction/PipeStage
                                     -> !Accessible aus Arch genommen
... PS_<name>_<nr>(accs.stages().get(Ident(accs.strings(), <name>))), ... {
                                     -> Auflösung zur Laufzeit
```

Ergebnisse

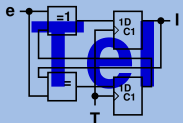
Architektur – Programm	Anzahl Takte	prosim	prosim-2	Veränderung
DLX – Registeraddition	851.967	51,8 s	54,2 s	+4,6 %
ARM7TDMI				
– FFT (Compiler)	176.613	10,9 s	10,4 s	-4,6 %
– FFT (Handoptimiert)	131.811	8,3 s	8,0 s	-4,0 %
– Primzahlensieb	15.234.723	1.020,0 s	975,0 s	-4,4 %
ARM9E-S				
– FFT (Compiler)	133.540	14,8 s	14,6 s	-1,4 %
– FFT (Handoptimiert)	116.657	12,6 s	12,7 s	+0,8 %



Frank Sakowski
Technische Universität Dresden
Institut für Technische Informatik
frank.sakowski@mail.inf.tu-dresden.de

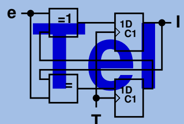
Auswertung

- generell Beschleunigung
- gegenläufige Entwicklung auf Vorverarbeitung zurückzuführen
- Ursache: Operationen Go/Flush, arithmetischer Ausdruck Queued
- Problem: Zeiger auf Pipelinestufen $\Rightarrow$ rückläufige Entwicklung
- Ergebnis abhängig vom Verhältnis Pipelineoperationen/-ausdrücke zu Nicht-Pipelineoperationen/-ausdrücke
- mit Behebung: Geschwindigkeit konstant und größer
- prosim/prosim-2 programmiertechnisch zu ähnlich für größere Geschwindigkeitsdifferenzen
- nur Neuentwurf bringt mehr Leistung
- hohe Zeitersparnis durch abstraktere Simulation $\Rightarrow$ Befehlssatzniveau



Zusammenfassung

- von architekturinterpretierend zu architekturcompilierend
- Operationsabarbeitung verändert
- Anbindung an NDO nicht umgesetzt ⇒ minimalistische Oberfläche
- geringfügige Beschleunigung



Frank Sakowski
Technische Universität Dresden
Institut für Technische Informatik
frank.sakowski@mail.inf.tu-dresden.de

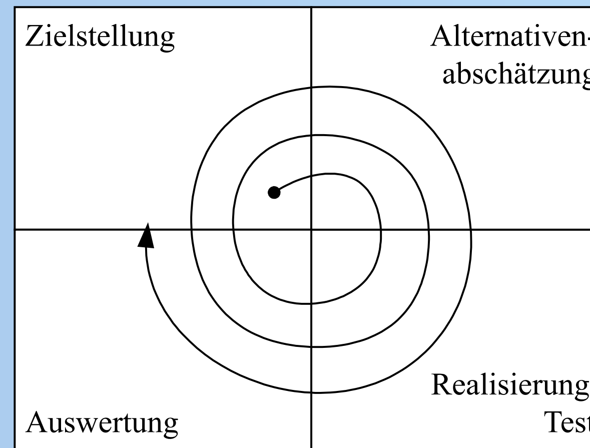
Ausblick

- Überarbeitung NDO, Änderung der Anzeigepolitik
- Simulationscache - für prosim und prosim-2 nutzbar
- Beschreibungen auf Befehlssatzniveau
- weiterer Verwendungszweck von DITO:
  - Coverifikation für Vergleich strukturell und funktionell
  - programmcompilierend für Software-Codesign, hybrider Ansatz
  - Hardware-Codesign: Simulationsschnittstellen, Hardwaresimulationsmodelle

Frank Sakowski
Technische Universität Dresden
Institut für Technische Informatik
frank.sakowski@mail.inf.tu-dresden.de

Funktionsüberprüfung

- Entwicklung nach Spiralmodell von Boehm



- Fehlerfreiheit/Korrektheit von prosim vorausgesetzt
- reine Funktionstests, keine Überprüfung der Simulatorinterna
- verschiedene Architekturen und Programme

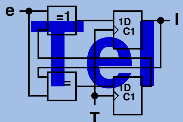
Frank Sakowski
Technische Universität Dresden
Institut für Technische Informatik
frank.sakowski@mail.inf.tu-dresden.de

Programm

- kurze Demonstration von prosim-2

Testumgebung

- Modifizierte Benchmarkfunktion von prosim
- Simulation ohne Benutzeroberfläche
- Simulationsabbruch mit Hilfe von Watchpoints
- Ergebnis: Anzahl der Taktzyklen und Simulationszeit



Frank Sakowski
Technische Universität Dresden
Institut für Technische Informatik
frank.sakowski@mail.inf.tu-dresden.de