TECHNISCHE UNIVERSITÄT DRESDEN

STUDIENARBEIT: ZWISCHENPRÄSENTATION

Trace-basierte Verifikation der FPGA-Implementierung eines MIPS-Prozessors

Valentin Gehrke

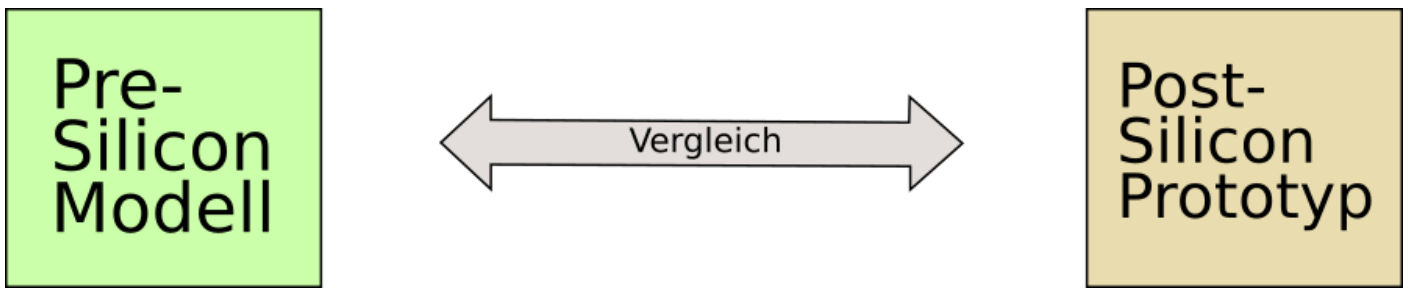
Dresden, 12.01.2017

Inhalt

01 Einleitung

Verifikation durch Vergleich

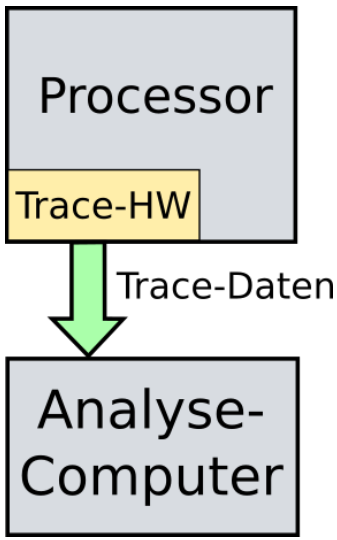
- Verhalten eines Prozessor-Prototyps mit vorherigen Modellen vergleichen
- „Post-Silicon Functional Verification“



01 Einleitung

Verhalten des Prototyps

- Moderne Mikroprozessoren besitzen eine Trace-Hardware
  - ARM Embedded Trace Macrocell (ETM)
  - Intel Processor Trace (PT)
  - u.a.
- Zeigt den Programmablauf in einem Prozessor
  - Instruktionen
  - Daten auf dem Bus
  - Daten in Registern
- Analyse von Software und Hardware



01 Einleitung

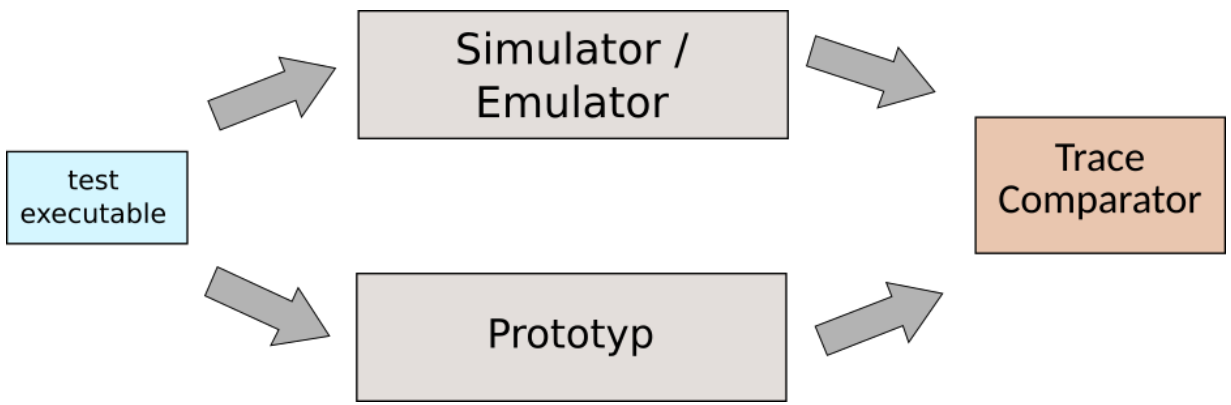
Verifikation mittels Trace

- Funktionale Korrektheit des Prozessors überprüfen
- **Annahme:** Programm ist korrekt; Prozessor enthält Fehler
- Prozessor Traces aufzeichnen
  - Program Trace
  - Bus Trace
- Verifikation des Traces
  - manuell
  - automatisiert über Regeln basierend auf der Spezifikation
  - Vergleich mit Referenz-Traces

01 Einleitung

Verifikation mittels Referenz-Trace

- Referenz-Trace aus anderem Modell
  - RTL-Simulation
  - Befehlssatzsimulator
- Gleiches Test-Programm im Prototyp und Simulator/Emulator
- Vergleichen der Traces
- Abweichungen können Fehler im Prototypen zeigen

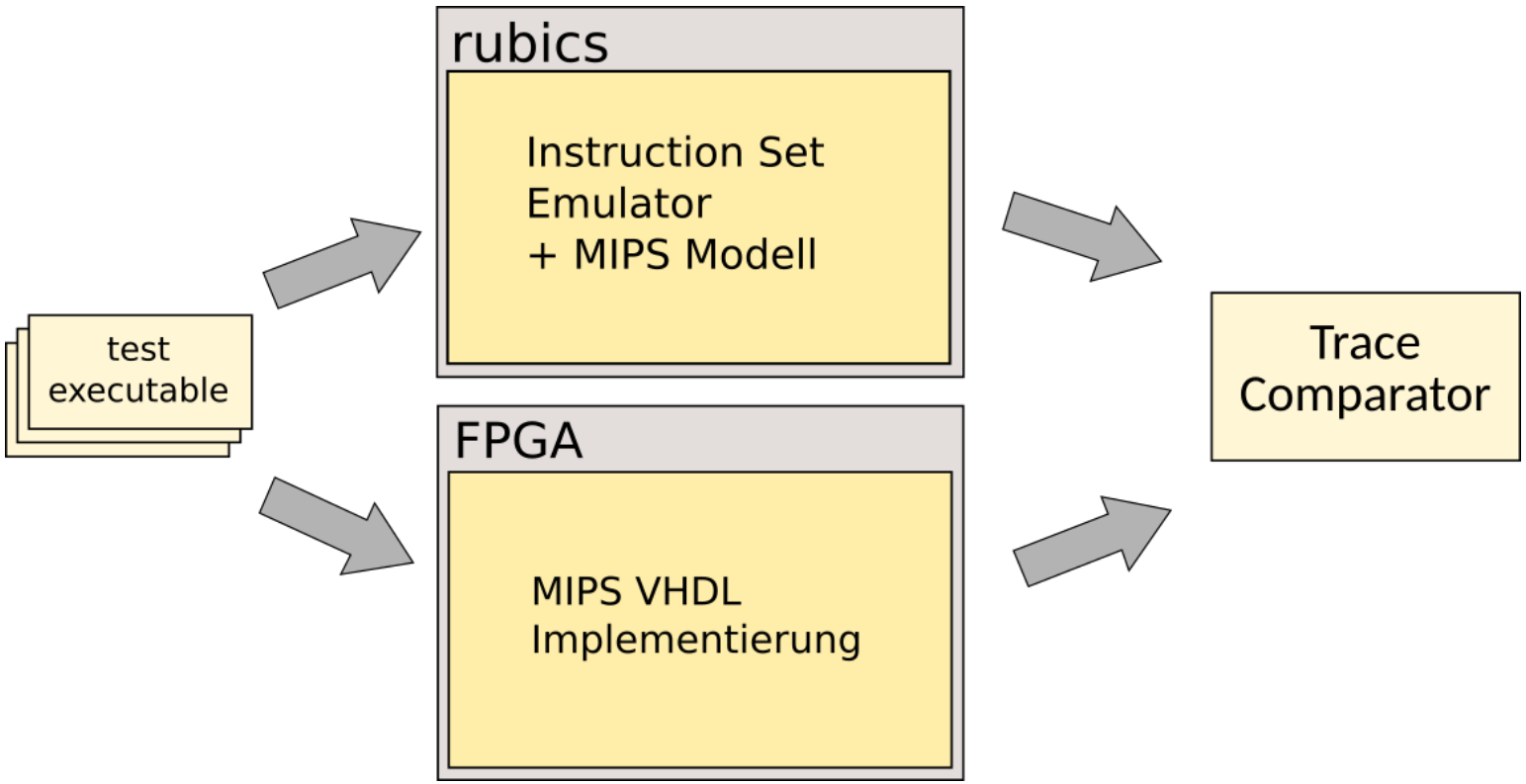


02 Thema
Das Thema dieser Studienarbeit

„Trace-basierte Verifikation der FPGA-Implementierung eines MIPS-Mikroprozessors“

- Funktionale Verifikation mittels Trace
- MIPS-Prozessor mit reduziertem Befehlssatz
- Befehlssatzemulator „rubics“ für Referenz-Traces
- Automatisierter Vergleich der Traces
- **Ziel:** Welche Vor- und Nachteile bietet „rubics“ gegenüber RTL-Simulationen in diesem Szenario.

02 Thema
Aufbau

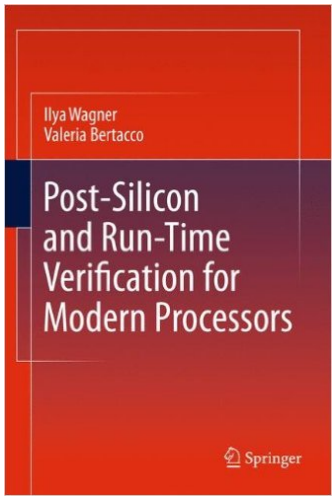


02 Thema
MIPS-Prozessor und Umgebung

- basierend auf MIPS-Prozessor R4000
- 32-Bit RISC
- Single-Core
- 31 Arbeitsregister
- reduzierter Befehlssatz (36 Befehle)
- keine Interrupts/Exceptions
- simpler Wishbone-Controller
  - Arbeitsspeicher
  - 7-Segment-Displays (des FPGA Boards)
  - Timer für Delays

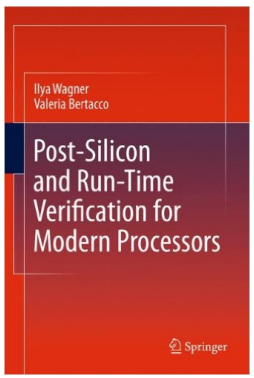
03 Literatur

„Post-Silicon and Runtime Verification for Modern Processors“



- Ilya Wagner (Intel Corp.), Valeria Bertacco (University of Michigan)
- Übersicht der Verifikation von Intel/AMD Prozessoren
- 2.2.2 Functional Post-Silicon Verification
  - Verifikation durch direkte und zufällige Tests
  - Simulatoren werden zum Erstellen der korrekten Ergebnisse benutzt
- Ansatz für Test Programme: Reversi Test Generation System

03 Literatur
Reversi Test Generation System



- Wert $\rightarrow$ Operation $\rightarrow$ Umkehroperation $\rightarrow$ gleicher Wert
- Operanden werden zufällig gewählt.
- Reversible Operationen
  - Addition $\Leftrightarrow$ Subtraktion
- Nicht-reversible Operationen
  - AND, OR $\rightarrow$ XOR $\Leftrightarrow (\overline{\mathrm{a}} \wedge \mathrm{b}) \vee (\mathrm{a} \wedge \overline{\mathrm{b}})$
- Speicher-Operationen
  - Mehrmals Datensatz im Speicher kopieren
  - mit XOR-Hash Daten überprüfen
- Mehrere Tests generieren und interleaved in ein Programm zusammenführen

03 Literatur

„Automatic Verification of Microprocessor designs using Random Simulation“

- Uppsala Universitet, Schweden
- OpenRISC 1200
- Zufällige Testprogramme
  - zufällige Befehle aus der Befehlsliste
  - zufällige Parameter
- Vergleich der Traces von RTL-Simulation und Befehlssatzsimulator
- Ergebnis: Fehler in 4 Befehlen in der RTL-Implementation gefunden

03 Literatur

„Architectural Trace-Based Functional Coverage for Multiprocessor Verification“

- University of Michigan, ARM Ltd.
- Vergleich von Traces
- verschiedene Simulationen (RTL, ISS, gem5)
- Traces der Cores in einen Trace kombinieren
- Probleme mit der Reihenfolge
  - Core-Interaktionen (z.B. Semaphoren)
- „Since all single-processor variants of an instruction set architecture must guarantee a unique correct execution flow of a program, their architectural traces must be identical.“

03 Literatur

„Functional Verification of a 32-bit RISC Processor Core“

- Tampere University of Technology, Finnland
- Verifikation des COFFEE RISC Cores
- 4.3 Usage of Random Input
  - Zufällige Befehlssequenz (linear feedback shift register)
  - Trace aus RTL-Simulation
  - Referenz-Trace aus Befehlssatzsimulator
- „The consept [sic] of directed random input together with high level models proved to be very valuable and efficient way of finding errors from the design“

03 Literatur
Weiteres

- „Functional Verification of a Multiple-issue, Out-Of-Order, Superscalar Alpha Processor - The DEC Alpha 21264 Microprocessor“ - Digital Equipment Corporation

  Verifikation des RTL-Modells durch Vergleich mit einem Befehlssatzsimulator mithilfe von direkten und pseudo-zufälligen Tests
- „SEQ.OPEN: A Tool for Efficient Trace-Based Verification“ - Garavel; Mateescu

  Trace wird in LTS konvertiert und mit regulären Ausdrücken und μ-calculus mit dem CADP Toolset verifiziert.
- „Evaluating and Comparing Simulation Verification vs. Formale Verification approach On Block Level Design“

  eingeschränkt zufällige Befehle werden dem Prozessor zugeführt und mit Regeln basierend auf der Spezifikation in jedem Takt überprüft

03 Literatur
Weiteres

- „Methodology for Processor implementation verification“ - Daniel Lewin
  Generierung der Tests basierend auf Zustandsmaschinen der Module des Prozessors
- „Trace-Based Post-Silicon Validation for VLSI Circuits“ - X. Liu ; Q. Xu
  Einführung in Verifikation mit Trace; Methoden zum Finden von funktionalen und strukturellen Fehlern
- „MIPS R4000 Microprocessor User’s Manual“
  Architektur und Befehlsliste des MIPS R4000

04 Aufgaben

Aufgaben in der Studienarbeit

1. MIPS-Befehlssatzmodell für „rubics“ entwerfen.
2. Trace-Plugin für „rubics“ entwickeln.
3. Testen beider Erweiterungen mit selbst erstellten und vorgegebenen Testprogrammen.
4. Programm zum automatisierten Vergleich der Traces entwickeln.
5. Traces von rubics und FPGA aufzeichnen.
6. Fehler in FPGA-Implementierung erzeugen und finden.

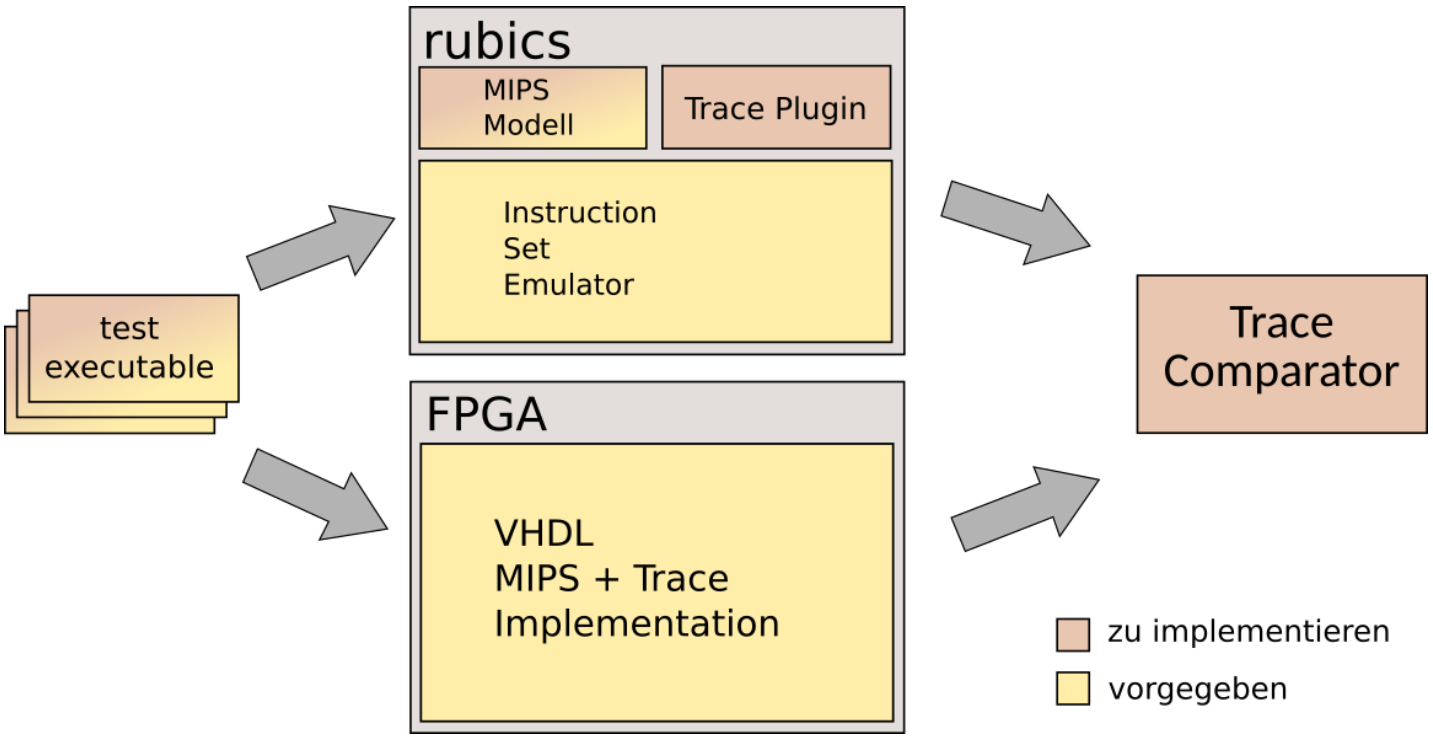
04 Aufgaben

Eigene Testprogramme

- direkte Tests
  - kleine Testfälle, einzelne Befehle testen
  - Sortieralgorithmen: Heap-Sort, Insertion-Sort, etc.
  - Suchalgorithmen: Lineares Suchen, Binäres Suchen
  - Fibonacci-Sequenz, Euklidischer Algorithmus
- zufällige Tests
  - ausgewählte Befehle mit zufälligen Parametern
  - Reversi-Prinzip
  - Generierung mit kontextfreier Grammatik
    - Register zufällig initialisieren
    - zufällige Befehle
    - Konstrukte (z.B. Zählschleifen, Bedingte Blöcke, u.a.)

04 Aufgaben

Aufgaben in der Studienarbeit



Vielen Dank für Ihr Aufmerksamkeit