TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut für Technische Informatik, Professur für VLSI-Entwurfssysteme, Diagnostik und Architektur

Simulative Verifikation und Evaluation des Speichermanagements einer Multi-Core-Prozessorarchitektur am Beispiel von SHAP

Christian Greth
christian.greth@mail.inf.tu-dresden.de

Dresden, 19.05.2010

Aufgabenstellung

1. Literaturstudium zu Verifikation und Evaluation von digitalen Systemen mittels Simulation und Trace.
2. Erarbeitung geeigneter Testszenarien für Verifikation der Funktionalität sowie Evaluation der Leistungsfähigkeit.
3. Identifikation von notwendigen Trace-Hardware- und Software-Komponenten unter Berücksichtigung der gegebenen Trace-Infrastruktur.
4. Implementierung und Test der identifizierten Trace-Hardware- und Software-Komponenten.
5. Durchführung und Auswertung der erarbeiteten Testszenarien.
6. Beurteilung des Speichermanagements hinsichtlich korrekter Funktion und Leistungsfähigkeit.
7. Zusammenfassung und Dokumentation der Ergebnisse.

Aufgabenstellung

1. Literaturstudium zu Verifikation und Evaluation von digitalen Systemen mittels Simulation und Trace.
2. Erarbeitung geeigneter Testszenarien für Verifikation der Funktionalität sowie Evaluation der Leistungsfähigkeit.
3. Identifikation von notwendigen Trace-Hardware- und Software-Komponenten unter Berücksichtigung der gegebenen Trace-Infrastruktur.
4. Implementierung und Test der identifizierten Trace-Hardware- und Software-Komponenten.
5. Durchführung und Auswertung der erarbeiteten Testszenarien.
6. Beurteilung des Speichermanagements hinsichtlich korrekter Funktion und Leistungsfähigkeit.
7. Zusammenfassung und Dokumentation der Ergebnisse.

Verifikation digitaler Systeme

Motivation:

- Überprüfung der Einhaltung von Designregeln.
- Fehlverhalten eines Designs finden und eliminieren.
- Die Funktion eines Designs mit der Spezifikation/Interpretation abgleichen.

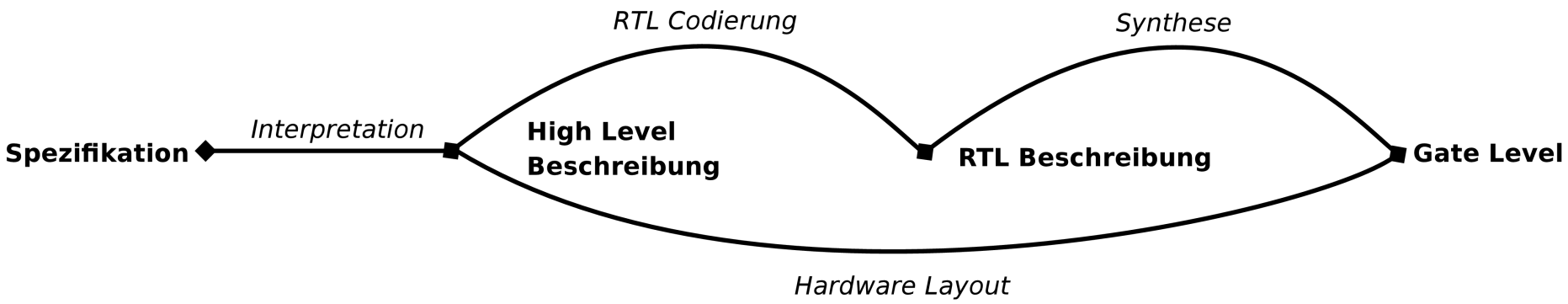
Problem [1]:

- 70% des Designprozesses werden für Verifikation/Test beansprucht [2].
- Verifikation bildet den kritischen Pfad.
- Verlust der Kontrolle über Low-Level-Funktionen durch Abstraktion.

  → Abstraktion reduziert jedoch die Dauer der Verifikation.

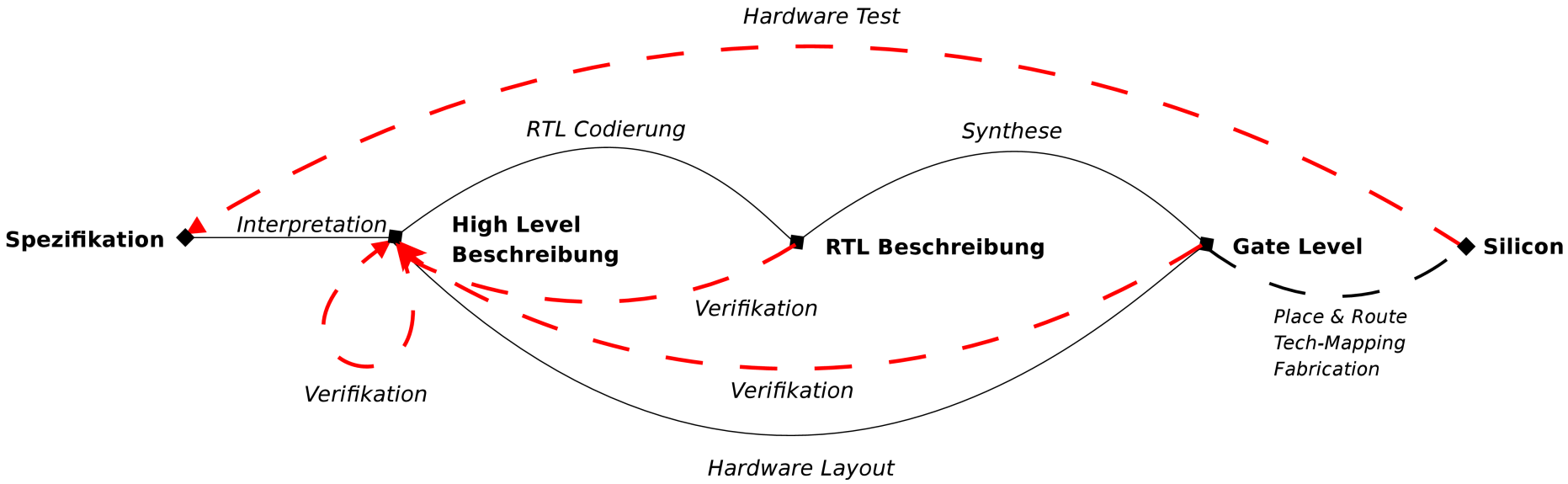
„Was soll verifiziert werden?“

Verifikation digitaler Systeme



→ Es existieren mehrere Ebenen um eine Verifikation durchzuführen.

Verifikation digitaler Systeme



- Verifikation ungleich Test [1]
- Verifikation gegen die Interpretation, nicht gegen die Spezifikation [1]

Verifikation digitaler Systeme

Formale Verifikation

- Nutzung mathematischer Methoden zum Beweis der Funktionalität.
- Formale Verifikation kann Fehlerfreiheit eines Entwurfs beweisen.
- Eine formale Beschreibung des zu verifizierenden Systems ist nötig.

Funktionale Verifikation (Validation)

- Prototyping (FPGA-based prototyping)
- Emulation
- Simulation

Funktionale Verifikation

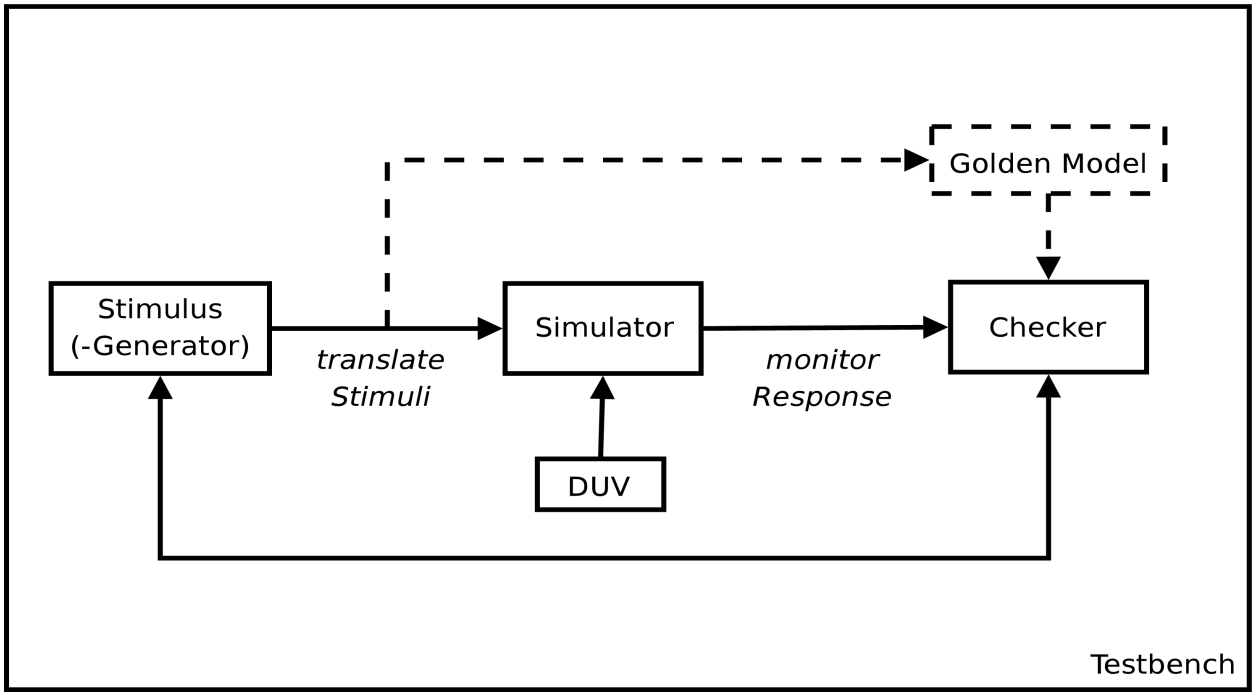
Prototyping (FPGA-based prototyping)

- schnell und vergleichsweise günstig
- Zeitanspruch für Implementation eventuell sehr hoch
- nur wenig Möglichkeiten für Debugging

Emulation

- Design wird in Hardware-Beschleuniger (Emulator) gemapped
- „Live“-Stimuli können injiziert werden
- Emulatoren sind schnell aber teuer
- Zeitanspruch für Implementation gering

Funktionale Verifikation



Simulation

- Stimulus (-Generator), Simulator und Checker bilden die Simulationsumgebung.

Funktionale Verifikation

Simulation

- sequentielle Ausführung „paralleler Hardware“
- Breakpoints und Start/Stop/Resume der Simulation
- Signale können zu jeder Zeit beobachtet werden

Stimuli-Erzeugung

- direkt durch verifizierende Person,
- automatisch (zufällig) durch Generator oder
- automatisch mit Bedingung durch Generator und verifizierende Person.

Funktionale Verifikation

Checker

- Visuelle Überprüfung der Ergebnisse durch verifizierende Person,
  → Waveform oder ASCII-Auszug
- Vergleich mit dem GoldenModel oder
- Vergleich mit vorher erzeugten Ergebnis-Vektoren.

HDL-Code-Überprüfung durch Coverage-Verfahren

- Auffinden ungenutzter Code-Zeilen oder Code-Fragmenten.
- Überprüfung von Bedingungen und Verzweigungen.
- Condition-, Expression-, Branch-, Statement-Coverage

Leistungsevaluation digitaler Systeme

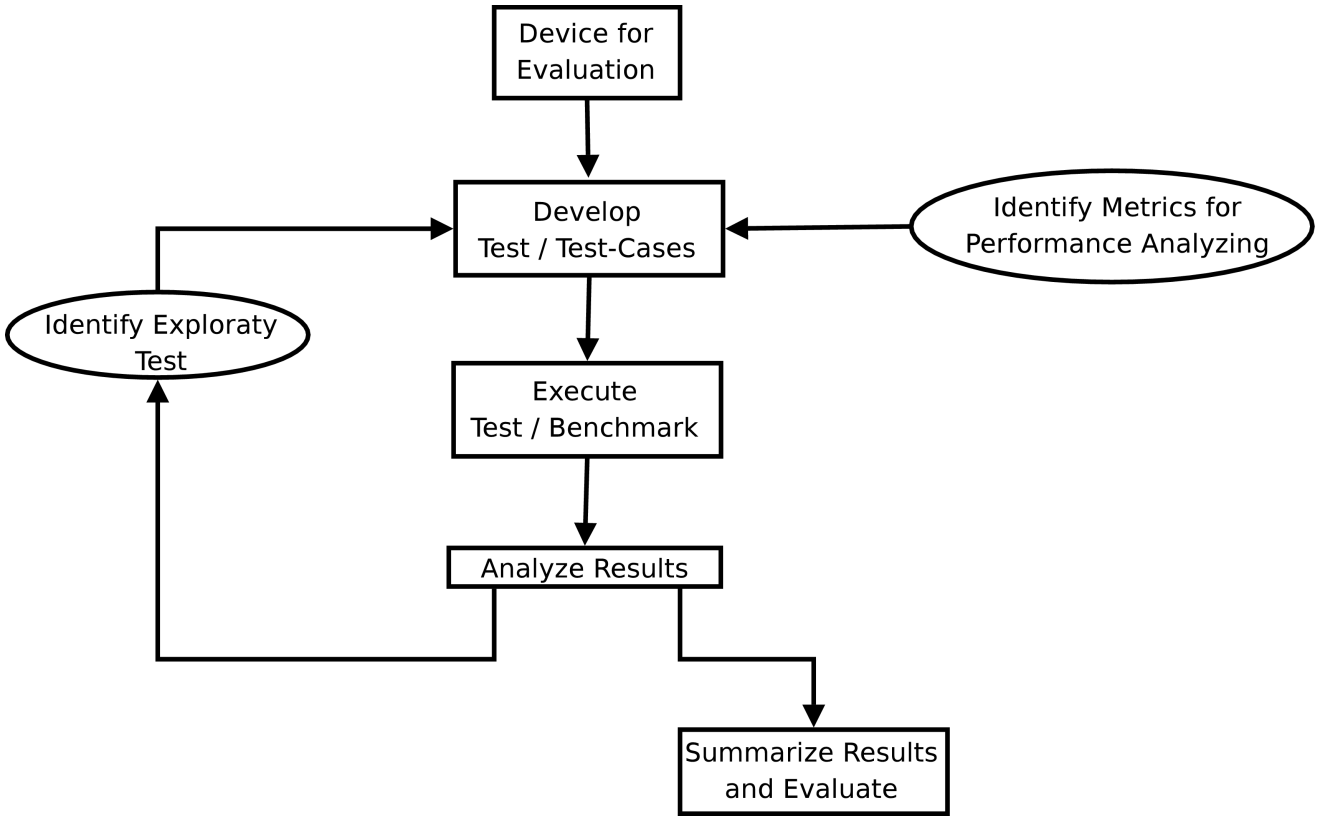
Motivation

- Leistungsfähigkeit eines Systems ermitteln/zeigen und anschließendes Bewerten.
- Vergleichsmöglichkeiten zu anderen/ähnlichen Systemen schaffen.
- Eventuelle Engpässe (bottlenecks) finden und beheben.

Leistungsevaluation durch auf die Hardware zugeschnittene Testszenarien oder durch allgemein gültige Benchmarks.

Es ist ein System zum Beobachten/Aufzeichnen der Aktivitäten nötig. → Trace

Leistungsevaluation digitaler Systeme



Leistungsevaluation digitaler Systeme

Metriken und Methoden

- Latenz
- Durchsatz
  - → Objektallokationen
  - → Objektzugriffe (Lese-/Schreibzugriffe)
- Stress-Test
- Corner-Test

Bewertungskriterien

- Maximum und Minimum
- Durchschnitt
- Worst-/Best-Case

Testszenarien

Voraussetzung

- VHDL-Quellcode von SHAP
- HDL-Simulator ModelSim mit Unterstützung für Code-Coverage
- Xilinx ISE für Synthese

- FPGA für Prototyp-Test
- SHAP mit integrierter Trace-Hardware

→ Verifikation und Evaluation durch Simulation und Trace

Testszenarien - Simulation

Simulation mit Testbenches

- Erzeugung von Testbenches um die Funktionalität der einzelnen FSM nachzuweisen.
- Ausführen der Testbenches in ModelSim.
- Zuhilfenahme der Coverage-Methoden in ModelSim um den VHDL-Quellcode hinsichtlich der Effizienz zu evaluieren.

Testszenarien - Trace

Wesentliche Funktionen des SHAP-Speichermanagements

- Speicher allokieren (Objektallokationen)
- Zugriffe auf den Speicher verwalten (Objektzugriffe)
- Speicher freigeben (Gargabe Collector)

Funktionen der Trace-Hardware

- Memory-Tracer zur Aufzeichnung der Speicherzugriffe
  → Adressen und Daten, Quelle des Speicherzugriffs, R/W-Zugriff
- Garbage Collection
  → Aufzeichnungen der Objektreferenzen

Testszenarien - Trace

Objektallokation

- Objekte gleicher/unterschiedlicher Größe allokieren
  → Latenz und Durchsatz
- Fragmentierung der Segmente des Heaps durch unterschiedliche Objektgrößen
- Objektallokation an Grenzbereichen
  → „voller“ Heap

Testszenarien - Trace

Objektzugriffe

- Objekte gleicher/unterschiedlicher Größe allokieren
- Objektreferenzen halten
- „zufälliger“ Zugriff auf Felder der angelegten Objekte
  → Durchsatz und Latenz
- Objektallokation und Objektzugriffe beobachten

- Objektzugriffe aus verschiedenen Threads auf unterschiedlichen Cores
  → „lock and release“

Testszenarien - Trace

Garbage Collector - GC

- Objektreferenzen freigeben und Verhalten des GC beobachten
- Mark and Sweep:
  → Werden alle freigegeben Objektreferenzen vom GC gefunden?
  → Werden alle markierten Objektreferenzen freigegeben?
- Compact:
  → Werden alle Objektreferenzen aktualisiert?
- Zeiten zwischen „Mark“ und „Sweep“ bzw. „Compact“ messen
  → Latenz um „neuen“ Speicher belegen zu können

Testszenarien - Trace

Zusammenspiel der 3 Testszenarien

- Kombination aus Objektallokation und Objektzugriff sowie Garbage Collection
- Zusammenspiel beobachten

Problem der Zufälligkeit:

- Zufälligkeit findet die meisten Fehler
- Wie soll Zufälligkeit erzeugt werden?
  → Einbringen von Zufallszahlen von Außen
  → Zufallszahlengenerator in Hardware

Zusammenfassung

Verifikation digitaler Systeme

- mehrere Ebenen für Verifikation
- formale Verifikation und Validation
- Validation durch FPGA-based prototyping, Emulation oder Simulation
- Validation kann Fehlerfreiheit nicht garantieren

Leistungsevaluation digitaler Systeme

- verschiedene Metriken für Messung und Kriterien zur Bewertung
- System zum Aufzeichnen/Beobachten nötig

Quellenangaben

[1] Bergeron, J.: Writing Testbenches. Functional Verification of HDL Models. Boston [u.a.]: Kluwer Academic, 2000.

[2] Wiemann, A.: Standardized Functional Verification. New York: Springer, 2008.