Universität Bremen

dfki Deutsches Forschungszentrum für Künstliche Intelligenz
German Research Center for Artificial Intelligence

Systems of High Safety and Security

Lecture 12 from 21.01.2026: Hardware Verification

Winter term 2025/26

Christoph Lüth

Organisatorisches

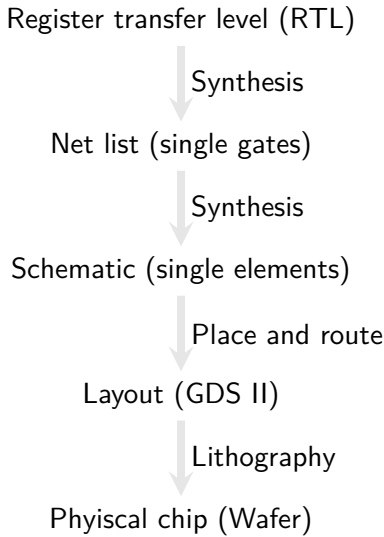
- Die Vorlesung am 28.01. **fällt aus!**
- Bitte an der stud.ip-Evaluation teilnehmen.

Roadmap

- Introduction
- Legal Requirements - Norms and Standards
- The Development Process
- Hazard Analysis
- The Big Picture: Hybrid Systems
- Temporal Logic with LTL and CTL
- Operational Semantics
- Axiomatic Semantics - Specifying Correctness
- Floyd-Hoare Logic - Proving Correctness
- A Simple Compiler and its Correctness
- TinyRV32I
- Hardware Verification & Conclusions

Why Hardware Verification?

- Hardware Manufacturing is **expensive**.
- Hence errors must **avoided**.
- Intel Pentium FDIV bug cost $ 475 mio (in 1994).
- Different verification methods at different levels.
- We will focus on functional correctness at RT-Level.



Combinational vs. Sequential Circuits

- **Combinational** circuits are functions: output determined **only** by input.
  - No hidden internal state
  - Examples: adders, multipliers, multiplexer
- **Sequential** circuits are **stateful**: output determined by input and internal state
  - On circuit level: feedback (“loop”)
  - Examples: latches, flip-flops, mealy automata, CPU cores

Languages

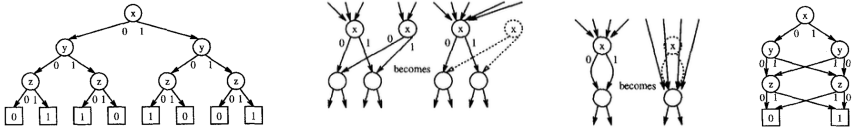
- Circuits are described by **hardware description languages**
  - VHDL, SystemVerilog — de-facto standard for lower layers
  - SpinalHDL/Chisl, Clash
- Properties are specified in PSL or SVA
  - Variations/Fragments of temporal logic

Verification Methods

- **Simulation:** Given input I, compute output O using the RTL model
  - Easy to use, only for small circuits
  - Corresponds to unit tests
- **Emulation:** Run on custum hardware (e.g. FPGA)
  - Can be costly, not for large circuity
- Extending the **coverage**: fuzzing, metamorphic test, ATPG
- **Formal Verificiation**: mathematical proof

Formal Verification: BDD

- Binary Decision Diagrams: Efficient represenation of boolean formulae
- Graph with variables (and constants) as nodes, values as edges
- Constructing BDDS: Example $x \oplus y \oplus z$



- **Canonical**, compact representation
- Used for efficient evaluation and **equivalence check**.

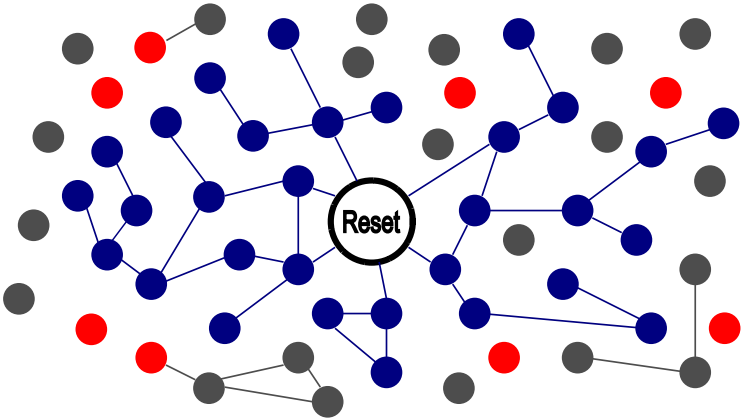
Formal Verification: SAT and SMT

- Basic question: can a formula ϕ be true
- SAT (Satisfiability): open propositonal logic
  - SAT Solvers take formula in conjunctive normal form (standard format: DIMACS), construct counterexample
  - Work for **huge** formulae (up to 10^7 variables)
  - Tools: chaff, MiniSAT, CP-SAT
  - Problem: very inexpressive logic
- SMT (Satisfiability modulo theory): first-order with natural numbers, linear inequalities, Presburger arithmetic
- Standard format (`smt-lib`)
- Tools: Z3, Yices, CVC

Bounded and Symbolic Model Checking

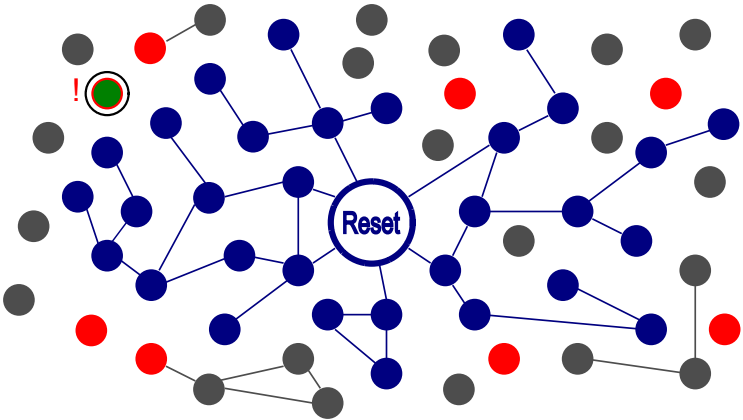
- Proving properties of **finite state machines**.
- Checking all states is **too expensive**.
- Only check **reachable** states

Symbolic Model Checking



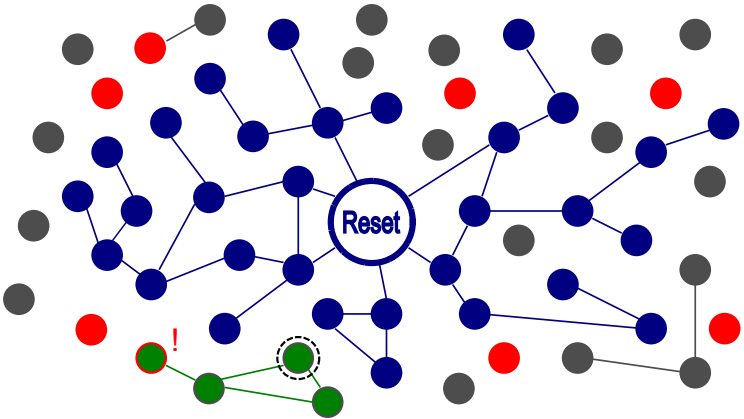
- Starting from initial state, explore all reachable states.
- **Correctness:** can we reach error states?
- Problem:
  - Exploring all states usually too expensive

Symbolic Model Checking



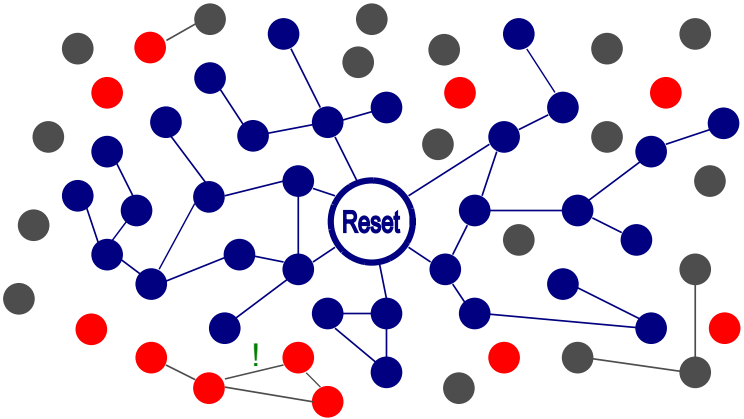
- Transitions given **symbolically**
  - Translate state transition into **first-order formula**
  - Translate correctness question into SMT problem
- **Bounded** model checking: up to depth N (here $N = 4$)
- Problem:
  - May miss errors

Symbolic Model Checking



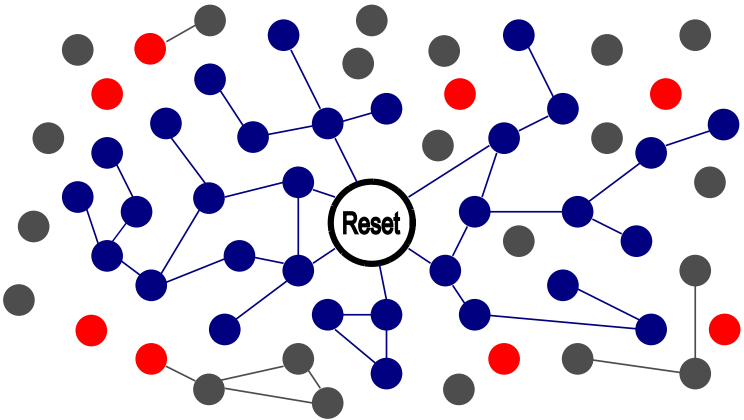
- To explore more states: **indeterminate** starting state
- Problem:
  - May generate **false positives**
  - Need to restrict start states
  - How to define suitable start states?

Symbolic Model Checking



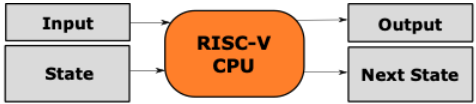
- To explore more states: **indeterminate** starting state
- Problem:
  - May generate **false positives**
  - Need to restrict start states
  - How to define suitable start states?

Symbolic Model Checking: Induction



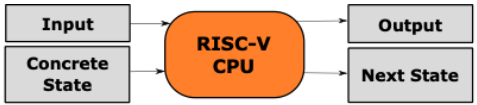
- Induction:
  - Show all start states are error-free
  - Show if state S is error-free and $S \rightarrow S'$ then S' is error-free
- Problem:
  - Needs symbolic description of error states
  - More complex properties may require transitions of greater depth

Case Study: Verification of a RISC-V CPU



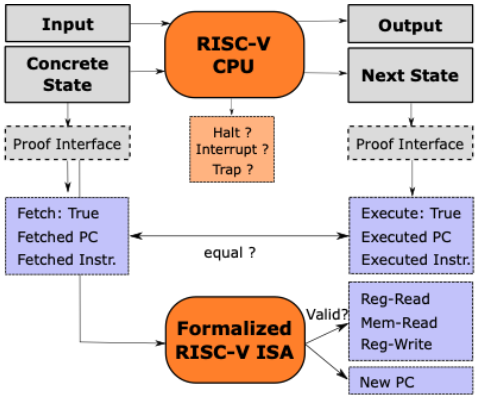
- What to verify?
- CPU core satisfies RISC-V ISA (RV32I)
- Core: VexRiscV (pipelined RISC-V core), written in SpinalHDL

Case Study: Verification of a RISC-V CPU



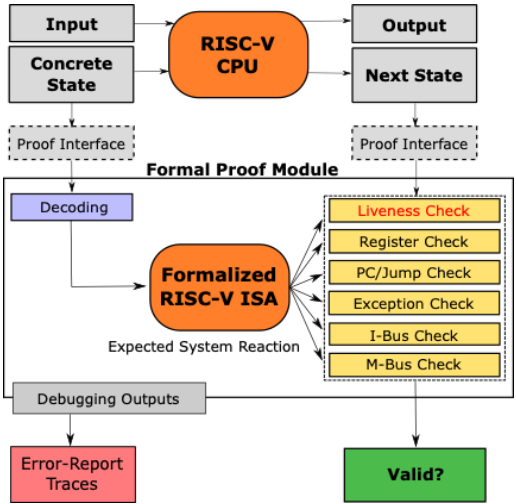
- What to verify?
- CPU core satisfies RISC-V ISA (RV32I)
- Core: VexRiscV (pipelined RISC-V core), written in SpinalHDL

Case Study: Verification of a RISC-V CPU



- What to verify?
- CPU core satisfies RISC-V ISA (RV32I)
- Core: VexRiscV (pipelined RISC-V core), written in SpinalHDL

Case Study: Verification of a RISC-V CPU



- What to verify?
- CPU core satisfies RISC-V ISA (RV32I)
- Core: VexRiscV (pipelined RISC-V core), written in SpinalHDL

- Proof results:
  - Illegal instructions recognized
  - Execution of 32 base instructions
  - Arithmetic in ALU
  - Reading/writing register file
  - PC behaviour
  - Memory interaction

Tools

- Commercial tools: e.g. Cadence Jasper Gold, OneSpin
- Open-Source:
  - abc — synthesis and verification
  - Symbiyosys — formal verification via SMT

Conclusions

- Hardware verification: slightly more **tractable** then software.
- Hardware verification is **industrially relevant**.
- Many success stories, most of them **confidential**.

Prüfungen

Prüfungen

- **Was** wird geprüft?
  - Verständnis des Stoffes
  - Fallbeispiel
- **Wann** wird geprüft?
  - Do, 05.02.2026, 9-14
  - Mi 25.03.2026 (ggf. zusätzlich Do 26.03.), 9-14