

Assertion-Based Verification

Marcela Šimková

Faculty of Information Technology
Brno University of Technology



LANGUAGE THEORY with APPLICATIONS 2011
December 14, 2011

Motivation

„It has been observed that verification becomes a major bottleneck in hardware design development, up to 80% of the overall development cost and time.“

-- R. Drechsler *et al*: Advanced Formal Verification

„The design and testing of an advanced microprocessor chip is among the most complex of all human endeavors.“

-- John Barton (Intel vice-president)

Verification

Verification - process that checks whether a system satisfies a given correctness specification.

- ◉ Verification of computer systems:
 - finding as many errors as possible in earlier phases of design,
 - prevent failures of systems in a real application.
- ◉ Consequences of errors:
 - loss of money (1994 Intel: floating-point error in Pentium processor and loss of 480 mil. \$),
 - loss of human lives (failure in military, aerospace, automobile, security systems).

Verification problems

- ◉ Verification requires more effort than the actual design itself.
- ◉ Duration of the verification process when verifying really complex systems is a **problem** !
 - Use advance verification techniques to localize problems faster, e.g. **Assertion-Based Verification**.

Assertion-Based Verification

Assertion-Based Verification (ABV) – a methodology used to formally express intended system behaviour, internal synchronization, expected operations using assertions.

Assertions – temporal logic formulas that must hold at all times during verification.

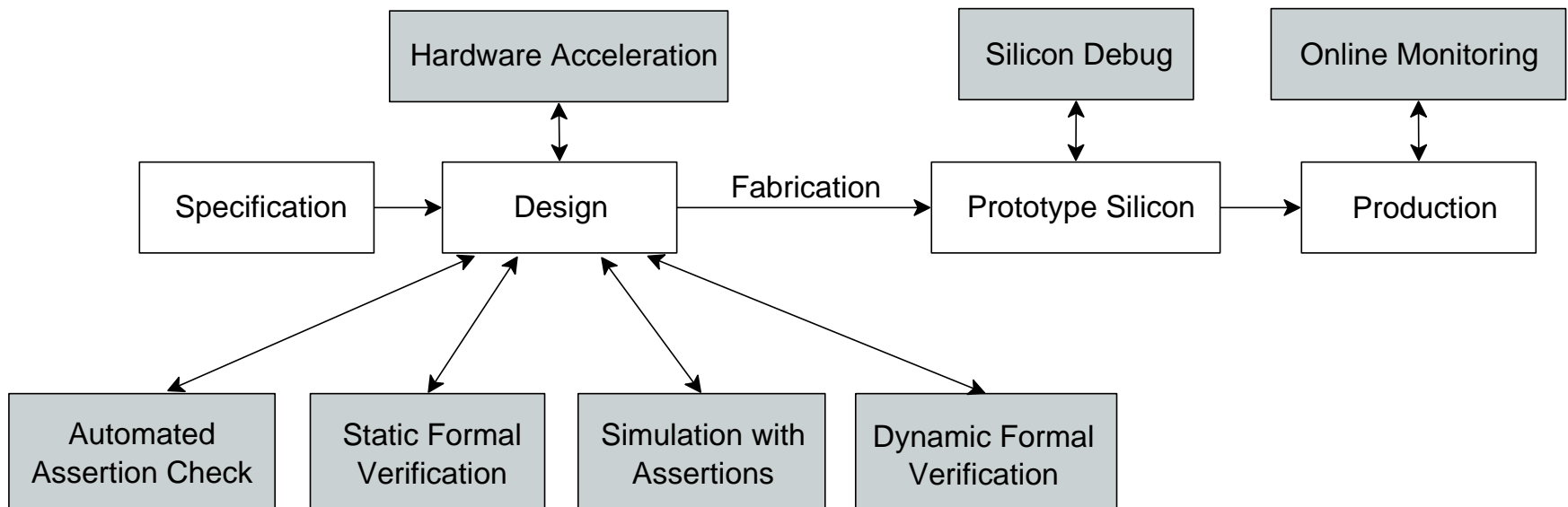
- ⦿ Fast detection and localization of problems.
- ⦿ Assertions can be expressed at different levels of a verified system:
 - internal and external interfaces,
 - cross-domain crossings,
 - directly inside a system.

Assertion Languages and Libraries

- ◉ Language-based assertions:
 - SystemVerilog Assertions (SVA),
 - Property Specification Language (PSL).
- ◉ Library-based Assertions (commonly used assertions):
 - Open Verification Library (OVL),
 - SystemVerilog Assertion Library,
 - CheckerWare Library.

Assertions in Development Cycle

- Assertions can be added at any stage of the design process and in any quantity.
- Assertions can be reused across different stages as well as in different designs.

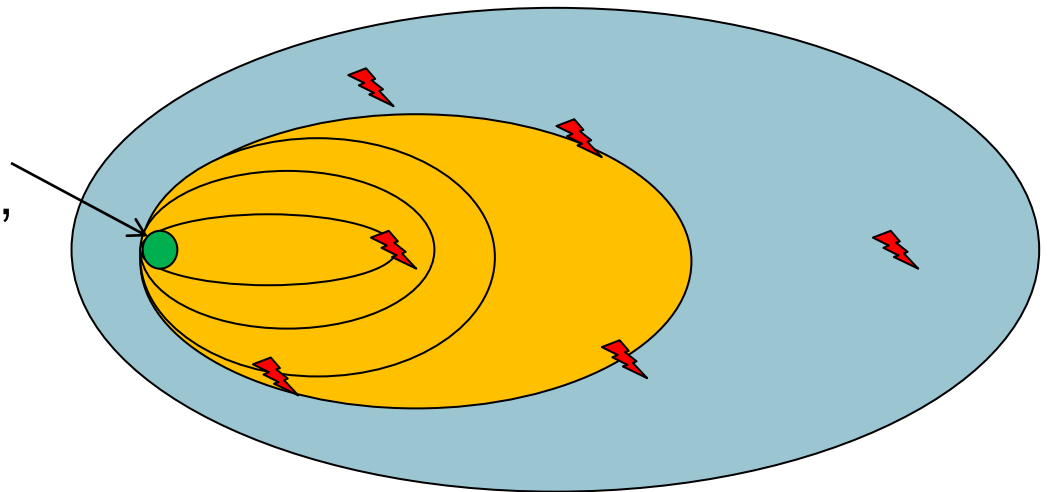


Automatic Assertion Check

- ⦿ Applies a set of pre-defined assertion rules to the RTL code of the design → fully automated process.
- ⦿ Synthesizes and formally analyzes the **internal structures** of the design.
- ⦿ Detected errors: overflow, case error, deadlock, unreachable states, blocks, incorrect cross-domain crossing, dead code, ...

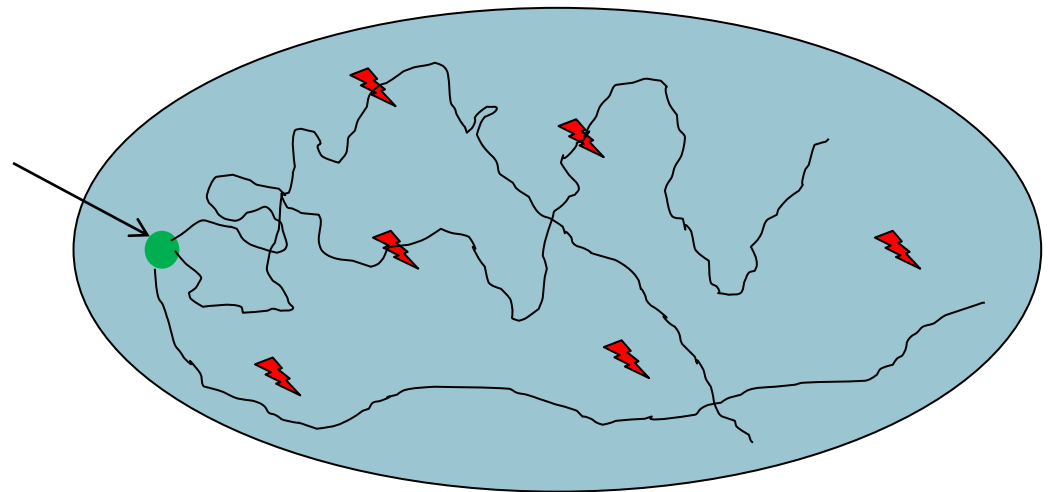
Static Formal Verification

- ◉ **Block level** verification using user-specified assertions.
- ◉ Analysis by **systematic state space exploration** (from starting state) using model checkers and theorem provers.
- ◉ Assertions can be determined as **true, false** or **indeterminate**.
- ◉ The scenarios in which the violation of these assertions happen are shown (**counter-example**).
- ◉ Detected errors: arbitration, resource sharing, allocation, access to buffers and memories, control logic, ...



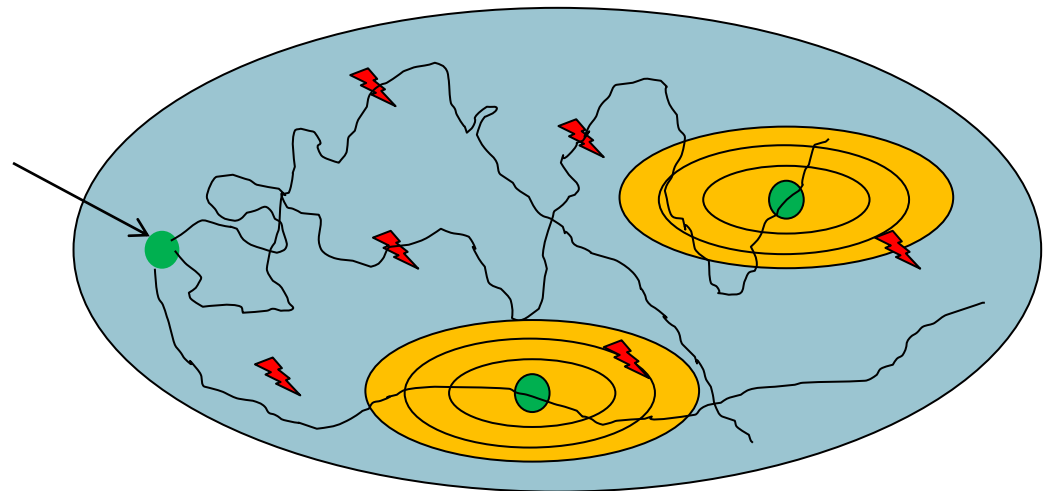
Simulation with Assertions

- Assertions are **simulated** with the design and the test environment in some simulator.
- Assertions passively monitor activities inside the design and on interfaces.
- Detected errors: inter-module communication, interface protocol violations.



Dynamic Formal Verification

- ◉ Runs concurrently with simulation and leverages all simulation vectors.
- ◉ Identifies states close to the assertions and performs local formal analysis about them.
- ◉ Explores hard to reach corner cases and logic.
- ◉ Detected errors: designs with long latency and serial interfaces, ...



Hardware acceleration

- ⊙ **Simulation** of inherently parallel hardware system **is extremely slow** when compared to the speed of real hardware.
 - Acceleration of verification using hardware accelerators or emulators.
- ⊙ Verified system together with some components of the verification environment are moved and executed in reprogrammable hardware, e.g. FPGA.
- ⊙ Assertion-based verification **is not implicitly supported** in hardware accelerators !
 - Synthesis of assertions into special *circuit-level checkers*, which ensure ABV also during hardware accelerated verification runs.

Assertion Checkers

- ⊙ Assumptions about Assertion Checkers:
 1. They should require few hardware resources and should be fast to allow high clock speeds.
 2. They should continually report errors in real time as the design is executed.
- ⊙ Assertions are written in high-level languages and are **not suitable** for direct implementation in circuit form.
- ⊙ **Transformation** into the form of hardware assertion checkers expressed in a Hardware Description Language (HDL):

Assertion \Rightarrow Büchi automaton \Rightarrow Finite-state machine

Büchi automata

A (non-deterministic) **Büchi automaton** β is a tuple $\beta = (Q, \Sigma, \delta, Q_0, F)$ where:

Q is a finite set of states,

Σ is a finite alphabet,

$\delta \subseteq Q \times \Sigma \times Q$ is the transition relation,

$Q_0 \subseteq Q$ is the set of initial states,

$F \subseteq Q$ is the set of accepting states.

- ◉ Büchi automata represent **languages of infinite words** → they accept them by looping through accepting states.
- ◉ Provided $(q_1, a, q_2) \in \delta$, we often write $q_1 \xrightarrow{a} q_2$.
- ◉ The **language** of β is defined as
 $L(\beta) = \{w \in \Sigma^\omega \mid \text{there is an accepting run of } \beta \text{ over } w\}$

Assertion Checker Example

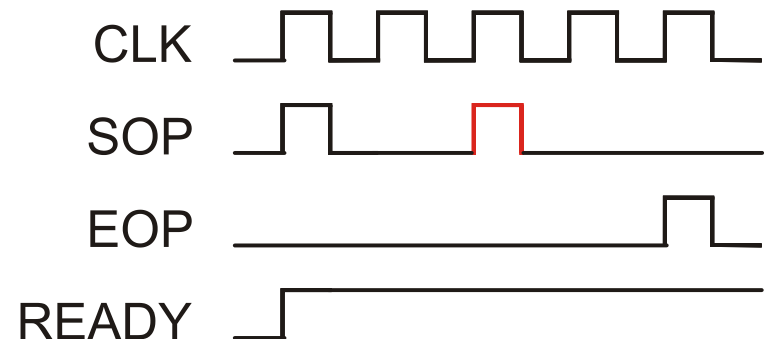
◎ Assertion:

```
// -- Matching EOP after SOP --  
// Each SOP must be, after some time, followed by EOP.
```

```
sequence eop_seq;  
  ##[0:$] EOP && READY;  
endsequence
```

```
property EOPMatchSOP;  
  @(posedge CLK) disable iff (RESET)  
  SOP && (!EOP) && READY |=>  
    (!(SOP && READY)) throughout eop_seq;  
endproperty
```

```
assert property (EOPMatchSOP)  
  else $error("SOP was not followed by  
    matching EOP.");
```



◎ LTL formula: $\varphi = (SOP \wedge \neg EOP \wedge READY) \rightarrow X (\neg (SOP \wedge READY) \cup (EOP \wedge READY))$

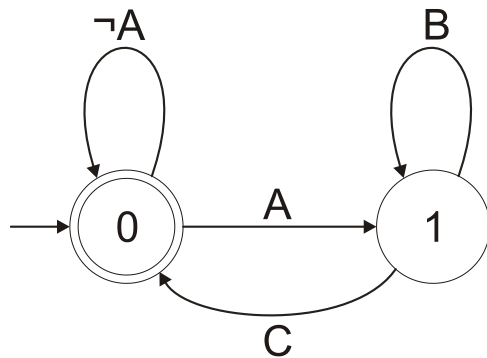
Assertion Checker Example

- LTL formula:

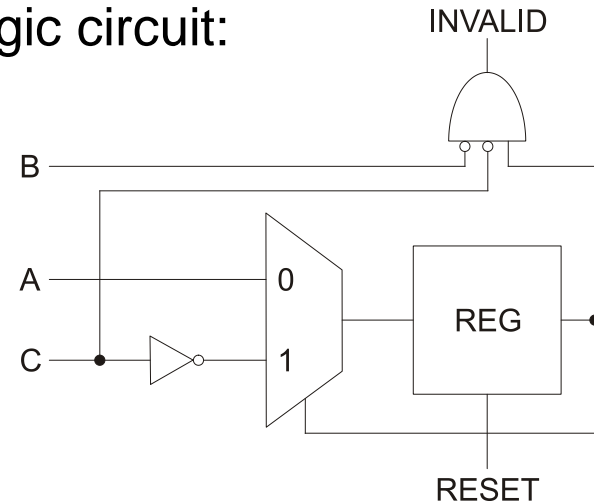
$$\varphi = \underbrace{(SOP \wedge \neg EOP \wedge READY)}_A \rightarrow X \underbrace{(\neg (SOP \wedge READY))}_B \text{ U } \underbrace{(EOP \wedge READY)}_C$$

$$\varphi = A \rightarrow X(B \text{ U } C)$$

- Büchi automaton:

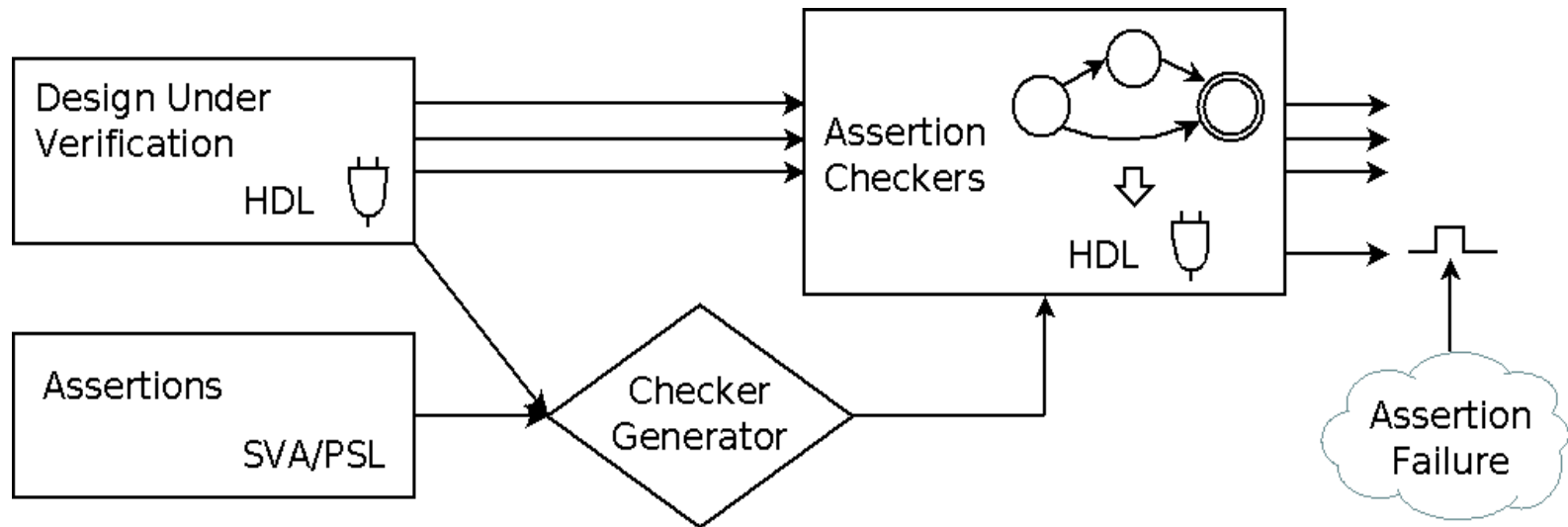


- Logic circuit:

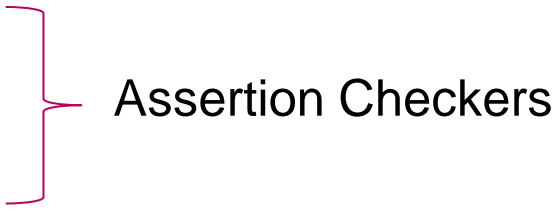


Generation of Assertion Checkers

- Automatic generation of checkers is much more advantageous than designing checkers by hand!



Conclusion

- ◉ It has been published that it is possible to achieve 50% reduction in debugging time when applying ABV.
 - ◉ ABV can be applied at different levels of design process:
 - specification,
 - static verification,
 - simulation,
 - dynamic verification,
 - hardware acceleration,
 - post-fabrication silicon debugging,
 - online-monitoring.
- 
- Assertion Checkers

Questions?