High-Level Decision Diagrams based Verification with PSL Assertions

(Previous: Automated Reasoning for Hardware Verification, A. Sudnitsõn)

⇒ Maksim JENIHHIN
Jaan RAIK
Alexander SUDNITSÕN
- Introduction
- Formal verification
- Simulation-based verification
- Assertion based verification (PSL, the paper)
- ~ 45,000 km²
- ~ 1,300,000 inhabitants
- capital: Tallinn (~400,000)

- Summer:
  - avg: 20°C / 70°F
  - max: (35°C / 95°F)
  - day ~ 18 hours

- Winter:
  - avg: -10°C / 15°F
  - min: (-35°C / -30°F)
  - day ~ 6 hours
Tallinn University of Technology

- The only technical university in Estonia
- 12,000 students
- 25% are at IT faculty
- IT Faculty consists of
  6 Departments:
  » Computer Engineering
  » Computer Science
  » Electronics
  » Computer Control
  » Informatics and Radio
  » Communication Engineering
32 employees: 5 professors, 6 associate professors, 10 researchers, 10 PhD students

Participation in R&D projects:
- 8 EU projects since 1993
- Bilateral research projects with Sweden, Poland, Germany

Around 80 papers published annually

Research topics:
- Hardware design (SoC, NoC, FPGA)
- Hardware verification (static / dynamic)
- Hardware testing (BIST, Boundary Scan, etc)
Verification is checking if the circuit was designed correctly.

Validation is similar to verification but it is performed on physical prototype.

Testing is checking every manufactured circuit for its correctness (absence of manufacturing defects).
- Digital circuits (i.e. ASIC, SoC)
  - not software!
  - not analog, RF, mixed-signal!
- Functional verification of a model (i.e. .vhdl)
Verification takes roughly 70-85% of design costs

Some companies have 2-4 verification engineers for every design engineer

A need to increase verification effectiveness

» Design-for-Verifiability (DFV)
Assertion-based Verification (ABV)

International Technology Roadmap for Semiconductors report
http://www.itrs.net
Introduction to hardware verification

1. Specification
   - Does it meet the specs?

2. Microarchitecture
   - Does it implement the microarchitecture?

3. RTL
   - Are they equivalent?

4. Gate netlist
   - Are they equivalent?

5. Layout

Verification

Property checking

Equivalence checking
Introduction to Hardware Verification 2

A

- specification
- design
- equivalent?
- alternative design
- design path
- verification path

B

- specification
- design
- satisfy?
- a specification expression
- property checking
- "alternative design"
Formal vs. Simulation-based verification

- **Formal verification (static)**
  - Intelligent (mathematical) proof of correctness
  - Constrained application

- **Simulation-based (dynamic)**
  - Simulation of input vectors (random or deterministic)
  - The most commonly used
Formal verification: Equivalence checking 1

- Decision Diagrams canonical form
- The idea:
  - Construct DD for the two circuits to be compared
  - Manipulate them to prove that they are equivalent
Formal verification: Equivalence checking 2

- \( d = f \oplus g \)
  - Assign the variables of boolean formula to evaluate it to TRUE (satisfiable, \( f \) and \( g \) are not equivalent)
  - OR prove that it evaluates to FALSE for all possible assignments (not satisfiable, \( f \) and \( g \) are equivalent)
\[ f(a, b, c) = (a + b + c)(\overline{a} + b + c)(a + \overline{b} + c)(\overline{a} + \overline{b} + c)(\overline{a} + b + c) \]

- **SAT** (Boolean satisfiability problem)
- 2-SAT is solvable in polynomial time
- 3-SAT is NP-complete
  - n-SAT can be reduced to 3-SAT in polynomial time
- Is \( f(a, b, c) \) solvable? \( = \) 1
- The solution is:
  - \( a = 1, b = 0, c = 0 \)
- The worst case is to try \( 2^n \) options
Model checking proves or disproves that a property (part of specification) holds for the circuit.

- Exhaustively searches the entire state space
  - In real life the space can be constrained

- Typically the properties are described in CTL (Computational Tree Logic)
  - \( \text{AG} (P \rightarrow ((\text{EX}.Q) \land \text{EX}\neg Q)) ) \]

<table>
<thead>
<tr>
<th>EG(f)</th>
<th>There is a path along which f holds at every state.</th>
</tr>
</thead>
</table>

Formal verification: Model checking
Coverage metrics

It is not feasible to simulate all possible input combinations. Usually it is not feasible to simulate all possible input stimuli (input data) covers.

3 types of coverage metrics in hardware verification:

- Code coverage
- Parameter coverage (next slide)
- Functional coverage

- Depends on implementation, used for parameters
- Depends not on implementation but on specifications
- Difficult to measure
Code coverage metric

- How good code entities are stimulated by simulations
- Depends on implementation
  - It is possible to have 100% code coverage on completely wrong implementation
- Easy to calculate

Statement coverage

| line 1: always @(posedge clock) |
| line 2: begin |
| line 3: a = b + c; |
| line 4: x = (y << 4); |
| line 5: if (a > x) |
| line 6: begin |
| line 7: y = b & a; |
| line 8: x = (x >> 2); |
| line 9: end |
| line 10: else |
| line 11: b = b ^ c; |
| line 12: if (x == y) |
| line 13: c = y; |
| line 14: else |
| line 15: y = a; |
| line 16: end // end of always |

Block coverage

| line 1: always @(posedge clock) |
| line 2: begin |
| line 3: a = b + c; |
| line 4: x = (y << 4); |
| line 5: if (a > x) |
| line 6: begin |
| line 7: y = b & a; |
| line 8: x = (x >> 2); |
| line 9: end |
| line 10: else |
| line 11: b = b ^ c; |
| line 12: if (x == y) |
| line 13: c = y; |
| line 14: else |
| line 15: y = a; |
| line 16: end // end of always |

Path coverage
ABV benefits:

» **Dynamic** – better observability
detecting bugs earlier and closer to their origin

» **Static** – better controllability
direct verification to the area of interest
Assertion-based Verification

- Completeness problem
  - Who/what and when should specify assertions?
  - When is it enough?

- In practice design engineer writes them for VHS (Verification Hot Spots). Such spot:
  - contains a great number of sequential states;
  - deeply hidden in the design, making it difficult to control from the inputs
  - has many interactions with other state machines and external agents
  - has a combination of these properties
PSL = Property Specification Language

- Based on IBM’s Sugar, developed by Accellera
- IEEE 1850 Standard in 2005

Flavors:
- VHDL, Verilog, SystemVerilog, GDL, SystemC

4 layers:

- **Boolean layer** – boolean expressions in HLD: \((a && (b || c))\)
- **Temporal later** – sequences of boolean expressions over multiple clock cycles, supports SERE: \({A[*3];B}\)\(\rightarrow\){C}
- **Verification layer** - directives for verification tool telling what to do with specified properties
- **Modelling layer** – models environment
reqack: assert always (req -> next ack);

Label

When to check

Verification directive

Property to be checked

\begin{align*}
\text{req} & \quad \text{ack} \\
\text{ack} & \quad + \\
\text{req} & \quad \text{req} \\
\text{ack} & \quad \text{ack}
\end{align*}
Proposed and developed in TUT

HLDDs are proved to speed-up simulation

» By up to factor 10 compared to commercial simulators
European Commission

6th Framework Programme Research Project

» ST Microelectronics (coordinator)
» Aerielogic and TransEDA
» Universities form
  Tallinn (Estonia)
  Linköping (Sweden)
  Southampton (UK)
  Verona (Italy)

2006 - 2008
Work-in-progress research at TUT

- HLDD-based verification flow

Validation patterns

- Stimuli & Testbench (VHDL)

Design model

- TLM (SystemC)
- RTL (VHDL)

Properties, assertions

- PSL (simple subset)

System representation

- HLDD Interface
- HLDD Model
- Properties, assertions
- HLDD Properties

Dynamic Verification:
- HLDD Simulation
- HLDD Coverage Analysis & Test Generation
- Dynamic Verification: Assertion Coverage
- Property Checker
- Assertions automatic translation is a complex process employing:
  - Nondeterministic Finite Automaton
  - Deterministic Finite Automaton
- Only subset of PSL properties is translatable
- An option to TUT internal solution is external translator like FoCs form IBM
p: assert always ({a; [*2] ;b} |=> {c});

```vhdl
PROCESS (clk)
BEGIN
  IF ( ( clk = '1' ) ) THEN
    focs_ok <= (focs_vout(4) OR NOT(c));
  ELSE
    focs_ok <= '1';
  END IF;
END PROCESS;
PROCESS
  VARIABLE focs_vout : std_logic_vector(4 DOWNTO 0);
BEGIN
  WAIT UNTIL (clk'EVENT AND clk = '1');
  focs_vout(4 DOWNTO 0) := reverse((focs_v(0) AND a) & (focs_v(1) AND '1') & (focs_v(2) AND '1') & (focs_v(3) AND b) & (focs_v(4) AND NOT(c)));
END PROCESS;
END PROCESS;  
```

**Translation using FoCs from IBM**

**FoCs vhdl checker**

**HLDD checker**
This presentation has given a brief overview of *hardware verification*

Work-in-progress of Ph.D. research was presented

» PSL assertions are used as HLDD simulation checkers in verification flow
Bibliography and contact

- **Book:**
  

- **Papers:**
  
  High-Level Decision Diagrams (HLDD) and DECIDER by Jaan Raik and Prof. Raimund Ubar from IEEEExplore

- **Contact:**
  
  papers on PSL and assertions
  
  **Maksim Jenihhin** – maksim@pld.ttu.ee
  
  Tallinn University of Technology, ESTONIA
Thank You!