Efficient Verification of Verilog Cell Libraries

Matthias Raffelsieper

TU/e

HWVW 2010

Motivation

Valichip project: Formal verification of cell libraries

- Cooperation between TU/Eindhoven and industrial partners Fenix Design Automation and NXP

Goal: Check that different functional descriptions are equivalent

Contributions:

- Defined a formal semantics for subset of Verilog
  - $\rightsquigarrow$ Observed differences in Verilog simulators
- Developed efficient analysis of non-determinism
- Identified functional behavior contained in timing descriptions

Acknowledgments

People that contributed to the Valichip project:

MohammadReza Mousavi

Jan-Willem Roorda

Chris Strolenberg

Wieger Wesselink

Hans Zantema

Outline

Cell Libraries

Cell Library: Collection of *standard cells* with different levels of abstraction, usually

- Transistor Netlist implementation

- Functional descriptions of cells in a subset of Verilog, called VeriCell and consisting of
  - Ternary Constants $\mathbb{T} = \{0, 1, X\}$
  - Variables, e.g., `ck`, `d`, ...
  - **Built-in primitives**, e.g., **not**, **and**, ...
  - **User Defined Primitives** (**UDPs**)
  - A module instantiating a number of primitives, thereby defining the cell

Example (D Flip-Flop with Active Low Enable)

```
    module dff_enb(q, d, ck, enb);
      output q; input d, ck, enb;
      not(en, enb);
      dff_en(q, d, ck, en);
    endmodule
```

Order-Dependence of UDP Evaluation

Example

```
primitive dff_en(Q, D, CK, EN);
  output Q; reg Q; input D, CK, EN;
  table
  //   D    CK    EN     : Q : Q'
       0   (01)   1      : ? : 0;
       1   (01)   1      : ? : 1;
       ?   (10)   ?      : ? : -;
       *     ?    ?      : ? : -;
       ?     ?    0      : ? : -;
       ?     ?    *      : ? : -;
  endtable
endprimitive
```

Orders: `CK, D` / `D, CK`

Values: $\overbrace{(0,1)}^{\texttt{D}}, \overbrace{(0,1)}^{\texttt{CK}}, \overbrace{(1,1)}^{\texttt{EN}} \overbrace{X}^{\texttt{Q}}$

Results: 0 / 1

- ⇝ Evaluation is parametrized by an order
  - Simulators use one specific order of evaluation
  - Not justified by real hardware behavior
- ⇝ Check order-independence
  - Whether output is independent of the order of considering inputs

UDP Evaluation

Given a UDP with n inputs.

- Input vector $\vec{i} = \big(\ (i_1^p, i_1), \ldots, (i_n^p, i_n)\ \big)$ contains previous and current value of all inputs
- $\Phi_j(\vec{i}, o^p)$: Output when considering j-th input changed
- List $\ell = j_1 : \ldots : j_k$ with entries between 1 and n not containing duplicates
  - $\ell = \mathrm{nil}$ denotes the empty list
  - ℓ is a permutation if $k = n$

Definition (UDP Evaluation Function)

$[\![\vec{i}, o^p, \ell]\!]$: Output of UDP after considering inputs in order ℓ

- $[\![\vec{i}, o^p, \mathrm{nil}\ ]\!] = o^p$
- $[\![\vec{i}, o^p, j : \ell]\!] = [\![\big(\ (i_1^p, i_1), \ldots, (i_j, i_j), \ldots, (i_n^p, i_n)\ \big), \Phi_j(\vec{i}, o^p), \ell]\!]$

- Most simulators use permutation $\ell = n : n{-}1 : \cdots : 1$

Semantics of VeriCell programs

Operational semantics with three phases: Execute, Update, Time-Advance

Execute:	Determine new outputs of *active processes* (instances for which an input has changed)
Update:	Clear current transitions, store new output values
Time-Advance:	When no more active processes and no updates, advance simulation time and apply new inputs

Model-Checking Equivalence [ACSD'09]

1. Encode VeriCell into transition system (using the presented semantics)
    - Encodes only the simulator order for UDPs to prevent blow-up
2. Create transition system from Transistor Netlist (using a standard algorithm)
3. Write both transition systems into one SMV file
4. Apply SMV model checker to verify equivalence

Order-Independence

- Output of a UDP might depend on order of evaluation
  - $\Rightarrow$ Non-deterministic behavior, when order is uncontrollable
  - $\Rightarrow$ Undesired in practice

Definition (Order-Independence)

A UDP with n inputs is called order-independent, if for all input vectors $\vec{i}$, all previous outputs o^p, and all permutations π, π':

$$[\![\vec{i}, o^p, \pi]\!] = [\![\vec{i}, o^p, \pi']\!]$$

- Checked in $\mathcal{O}(n!)$ function comparisons
  - Keeping one permutation constant, e.g., the identity permutation

Can we do better?

Commuting Diamond Property

Definition (Commuting Diamond Property)

Inputs $1 \leq k, m \leq n$ with $k \neq m$ have the commuting diamond property $(k \diamond m)$, if for all input vectors $\vec{i}$ and previous outputs o^p:

$$[\![\vec{i}, o^p, k : m]\!] = [\![\vec{i}, o^p, m : k]\!]$$

$$\begin{tikzcd}
 & \vec{i}, o^p \arrow[dl, "k"'] \arrow[dr, "m"] & \\
\vec{i}_k, o' \arrow[dr, dashed, "m"'] & & \vec{i}_m, o'' \arrow[dl, dashed, "k"] \\
 & \vec{i}_{k,m}, o &
\end{tikzcd}$$

Efficient Analysis of Order-Independence

Theorem [FMICS'09]

A UDP with n inputs is order-independent, if and only if for every pair $1 \leq k < m \leq n$ we have $k \diamond m$.

- Checked in $\mathcal{O}(n^2)$ function comparisons
- Relies on specific properties of UDP evaluation

Considering Timing Checks

- Full order-independence is very unlikely
  - Often some data is clocked in, then the order is important

- Use further information given in the cell library

  Timing Checks specify time windows in which two inputs must not both change

Example

`$setuphold(posedge ck, d,` t_s, t_h`);`

⇒ Remove counterexamples contradicting the timing checks

⇒ When no more counterexamples, then UDP is order-independent in environments respecting the timing checks

Module Paths [DATE'10]

Timing behavior of cells given by Module Paths
(a.k.a. Timing Arcs, Delay Arcs, . . .)

- Describe that input changes can cause certain output changes
  $\rightsquigarrow$ Functional behavior

1. Checking feasibility of module paths to increase confidence in delay calculation
   - Not taking the exact values into account
2. Complementing technique: Deriving module paths from the functional description
   - All possible module paths have been treated
   - Forgotten module paths treated as 0 delay by simulators

Approach

Express as reachability problems and use symbolic model checking

Experimental Results

Validated all presented techniques on industrial cell libraries

- Including publicly available Nangate Open Cell Library

Results:

- Time required for complete analysis in the range of a few seconds per cell
- Order-dependent behavior found for 2 cells of the Nangate cell library
  - Seems to be a forgotten timing check
  - When adding the missing timing check then also order-independent

Conclusion and Outlook

Conclusion:

- Automatic equivalence checking of cell libraries [ACSD'09]
- Efficient method to analyze non-determinism of Verilog UDPs [FMICS'09]
  - Recently also adapted to transistor netlists [ACSD'10]
- Feasability checking and derivation of module paths from functional descriptions [DATE'10]
- Applied our techniques to industrial cell libraries

Future Work:

- Encode delays into transition systems
- Enlarge VeriCell subset of Verilog
  - Include built-in primitives that distinguish fourth value Z
  - Problem: Introduces further non-determinism
- Incorporate slicing to deal with larger designs