A New Equivalence Checker for Demonstrating Correctness of Synthesis and Generation of Safety–Critical Software

Eui-Sub Kim, Junbeom Yoo

Dependable Software Laboratory
KONKUK University

2016.05.27
Introduction

• **Purpose : New Equivalence Checker Development**
  – for demonstrating correctness of synthesis and generation of safety-critical Software

• We are developing the equivalence checking engine from the scratch.
  – The checker can directly verify the FBD program
    • without any translation process and any assistance of other checking engine.

• **Q. Why equivalence checking is necessary?**
Introduction

• A. Many Transformations

  – The initial design usually undergo a number of transformations such as program translation, synthesis and P&R.

  – How can you prove that the translation/optimization tools correctly translate the original program into another one?

  – Especially when the tools are used in development of safety critical software, you have to verify the tool! to prevent unintended accident.
Representative Techniques

1. Simulation
   - Most widely used technique
   - It requires test vectors and tiring process (re-simulation and comparison)
   - **It is impossible to check all of input vector.**
     - It should check all possible input vectors ($2^{\text{input bits}}$).
     - Simulating all possible input-output pairs is Co-NPHard.

2. Equivalence Checking
   - Formal Verification technique
     - Formal verification is a type of static analysis that applies *mathematical techniques* to rigorously prove that a design functions correctly.
   - **It formally prove that two programs exhibit exactly the same behavior.**
   - This verification technique can be performed quickly and without the need for test vectors.
Platform Change from PLC to FPGA

• **Target:**
  - RPS (Reactor Protection System)
  - Safety critical component of I&C in the Nuclear power plants

• **Recently,**
  - **FPGA** has received much attention from nuclear industry
    - Increasing maintenance cost
    - CCF(Common Cause Fault) problem
    - In-depth strategy for security
We developed an integrated development framework

**NuDE →**

Each process needs specific tools provided by the FPGA vendors.

- **Nuclear regulation authorities** require more considerate demonstration of the correctness of the mechanical tools,
- even if the FPGA industry have acknowledged them empirically as correct and safe processes and tools.
Logic Synthesis in FPGA Development

- **In theory**, a logic synthesis tool guarantees that the first netlist is logically equivalent to the RTL source code.

- **In practice, software can have bugs!!**
  - It would be a major risk to assume that all steps from RTL through the final tape-out netlist have been performed without error.

- **Therefore, a verification** for synthesis tool and process is needed to check the logical equivalence.
Development of the EC tool from the scratch

- Industrial equivalence checking tools
  - Imitated application and royalty
  - No applicable LEC for Synopsys Synplify Pro (in Actel Libero IDE)
    - In this case, we need to develop a customized or new LEC

<table>
<thead>
<tr>
<th>Logic Synthesis</th>
<th>IDE</th>
<th>Mentor Graphics FormalPro</th>
<th>Cadence Encounter Conformal EC</th>
<th>Synopsys Formality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mentor Graphics</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Precision RTL</td>
<td>Xilinx ISE</td>
<td>O</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>Actel Libero Soc</td>
<td>O</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synopsys Synplify Pro</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Xilinx ISE</td>
<td>O</td>
<td>O</td>
<td></td>
<td>No LEC available</td>
</tr>
<tr>
<td>Actel Libero Soc</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Altera Quartus II</td>
<td>O</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Xilinx XST</td>
<td>Xilinx ISE</td>
<td>O</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>Synopsys DC Ultra</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
New Equivalence Checker

• **We are developing the equivalence checking engine from the scratch.**
  
  – We can now perform **combinational equivalence checking** against two version of FBD programs (an original FBD vs. a modified version FBD).

  – It can **directly verify the FBD program** without any translation process and any assistance of other checking engine.

• **Equivalence Checking Engine**

  • We will check the combination ‘Actel Libero IDE’ with ‘Synopsys Synplify Pro’ synthesizer, which is the combination of the project we are working with.

  • We can save the royalty and have our EC core engine.
In detail

- Functional Equivalence Checking
  - two programs are equivalent if their representations are identical.
- Using ROBDD (Reduced Ordered Binary Decision Diagram)
- Branch based Backward tracking method
- Process
  - Input \(\rightarrow\) two FBD programs
  - Translation \(\rightarrow\) intermediate format for BDD
  - Generation \(\rightarrow\) BDD
  - Comparison \(\rightarrow\) both BDD
  - Result \(\rightarrow\) “equivalent” or “counter example”
Conclusion & Future work

• Conclusion
  – New combinational equivalence checker
  – The first target is FBD program,
  – We developed the equivalence checking engine from the scratch.

• Future work

  • Function:
    – Sequential Equivalence Cheeking of FBD program

  • Various Input programs:
    – Verilog
    – VHDL
    – Gate-level netlist (EDIF)

  • Graphic:
    – Visualization of counter example
    – User friendly GUI

  • Evaluation & Improvement !!
    – Speed
    – Memory usage

  • Model checking
Future Work

• We are now planning to extend the tool can be used in anywhere in \textit{NuDE}.

• If all verifications succeed, we can say that the \textbf{final} software will operate exactly as we intended.