Property-Driven Development of a RISC-V CPU

Tobias Ludwig
Technische Universität Kaiserslautern (TUK)
Status of Formal Verification

The biggest hurdle

black box verification
(simulation)

white box verification
(property checking)

Property Checking:
More like *design* than *verification*!
Vision

Formal RTL verification should:

- do more than *bug hunting*
- support new *abstraction principles* between electronic system level models (ESL) and low-level implementations (RTL)
- help to emancipate ESL models from *prototypes to golden design models*
Semantic Gap

High trust in RT level as golden reference!
– Why do we trust the RT level?
– Equivalence proven through equivalence checking (EC)

Trust in system-level models as golden reference?
– Almost no trust, except of HLS
– No notion of equivalence \(\rightarrow\) EC not possible
– High-level synthesis is taking away too many RT-design decisions
Path Predicate Abstraction (PPA)

Operationally colored graph $G$

Path predicate abstracted graph $P$
Soundness Theorem

Let P be a path predicate abstraction of a graph G. Then, for every (finite or infinite) path in G visiting a sequence of colored states \((w_0, w_1, w_2, \ldots)\) there exists an abstract path representing the same sequence of colors \((c(w_0), c(w_1), c(w_2), \ldots)\) in P, and vice versa.
Path Predicate Abstraction (PPA)

- Colored node: Important state
- Uncolored node: Unimportant state
- Operation: Transition between important states:
  - e.g., blue goes to green
Soundness

– Verification results obtained at ESL translate to the RTL
– Global verification tasks can be moved from the RTL to the ESL
– Significantly less chip-level simulation is required

Theorem shown for LTL properties
PPA in practice?

Setup:
- ESL model of a bus
- FSM of each module is a PPA
- Implement sound RTL for each PPA
PPA in practice?

Setup:
- ESL model of a bus
- FSM of each module is a PPA
- Implement sound RTL for each PPA

Verify on system-level:
- E.g.: Simulate sending a message across the bus at the ESL
- Working correctly? Due to soundness RTL works correct, too
- No more verification required for system-level behavior
PPA in practice?

Setup:
- ESL model of a bus
- FSM of each module is a PPA
- **Implement sound RTL for each PPA**

Verify on system-level:
- E.g.: Simulate sending a message across the bus at the ESL
- Working correctly? Due to soundness RTL works correct, too
- No more verification required for system-level behavior
Property-Driven Development (PDD)

Basic Idea

• System model as formal specification:
  – Designable/implementable subset of SystemC (SystemC-PPA)
  – Generate complete set of properties
• Refine generated properties (templates)
• Prove properties on final implementation
DeSCAM: from ESL to RTL
DeSCAM

- **DeSCAM** (“**Design from SystemC Abstract Models**”):
  - Analyzes a given SystemC model for compliance with the designable subset
  - Supports by refining the model into a SystemC-PPA
  - Automatically generates the properties

- A manual explains how to use DeSCAM in Property-Driven Development (from a practical point of view)

**Available on GitHub**: github.com/ludwig247/DeSCAM
Designable Subset: SystemC-PPA

- Models an FSM in a *time-abstract* fashion
- Single thread executing infinitely
- Only *blocks* if a communication interface is called
- No cyclic path without a blocking communication
- No *dynamic memory* allocation
Extracting the PPA from the SystemC CFG

SC_MODULE(encoder) {
    // constructor and declarations
    void fsm() {
        while (true) {
            bus → read(status, data)
            if (status == encode) {
                // encode(data)
                bus → write(data);
            }
            // some code
        }
    }
};
Extracting the PPA from the SystemC CFG

```cpp
SC_MODULE(encoder) {
   // constructor and declarations
   void fsm() {
      while(true){
         bus→read(status,data)
         if (status==encode) {
            // encode(data)
            bus →write(data);
         }
         //some code
      }
   }
};
```
Extracting the PPA from the SystemC CFG

```
SC_MODULE(encoder) {
   // constructor and declarations
   void fsm() {
      while(true){
         bus->read(status,data)
         if (status==encode) {
            // encode(data)
            bus->write(data);
         }
         //some code
      }
   }
};
```
Operation properties

- Operations start and end in important states
- Important states subsume millions of concrete states
- Operations have an arbitrary, (but fixed) length of n cycles
- Objects of the system-level are referenced by macros/functions

```
property encode;
assume:
  at t: start_state;
  at t: status == encode;
prove:
  at t+n: out(encoded(data@t));
  at t+n: end_state;
endproperty;
```
property encode(length);
//freeze variables
    int data;
    status_t status;
// freeze values @t
    t ##0 hold(data_0, port_in_data()) and
// triggers
    t ##0 start_state() and
    t ##0 sync() and
    t ##0 port_in_status() == encode
implies
    t_end(length) ##0 end_state() and
    t_end(length) ##0 port_out == encoded(data_0)
endproperty

function int port_in_data()
    return {
        $past(unit/data_in,7),
        $past(unit/data_in,6),
        $past(unit/data_in,5),
        $past(unit/data_in,4),
        $past(unit/data_in,3),
        $past(unit/data_in,2),
        $past(unit/data_in,1),
        unit/data_in
    }
endfunction
What is the *promise*?

• Correct-by-construction design
• Increased design productivity
• No RTL simulation
• Support for aggressive optimization techniques
Case study: RISC-V CPU

- PDD design of a RV32I processor
- Instruction set simulator (ISS) as a designable SystemC-PPA
- Two RTL implementations
- Implementations are sound refinements of the same ESL
Experimental Results RISC-V

• Work effort for RTL design:
  – Sequential: 2 weeks
  – Pipelined: 4 weeks

• Work effort for verification:
  – Sequential: 3 person days
  – Pipelined: 4 weeks

<table>
<thead>
<tr>
<th>Design Size</th>
<th>RTL States</th>
<th>LoC</th>
<th>PPA /Oper./Var.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td>1881</td>
<td>1430</td>
<td>6 states/ 21 op/ 7 var</td>
</tr>
<tr>
<td>Pipelined</td>
<td>2502</td>
<td>2020</td>
<td>6 states/ 21 op/ 7 var</td>
</tr>
</tbody>
</table>

Simulation Results

- Prime Calc
  - ISS: 5
  - Sequential(RTL): 55
  - Pipelined(RTL): 83

- Fibonacci Calc
  - ISS: 1
  - Sequential(RTL): 109
  - Pipelined(RTL): 135

- Bubble Sort
  - ISS: 8
  - Sequential(RTL): 133
  - Pipelined(RTL): 208
Industrial Case Study

- Provided industrial Framer
- Extracting SystemC-PPA: 6 PM\(_{\text{person month}}\)
- Top-down redesign: 1PM
- 22 properties proven in 22min
- FF reduction 10%
- Power saving: up to 50%

<table>
<thead>
<tr>
<th>Module</th>
<th>RTL States</th>
<th>LoC RTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framer(or.)</td>
<td>4.2k – 47k</td>
<td>27k</td>
</tr>
<tr>
<td>Monitor(or.)</td>
<td>30</td>
<td>850</td>
</tr>
<tr>
<td>Framer(re.)</td>
<td>3.9k – 42k</td>
<td>12k</td>
</tr>
<tr>
<td>Monitor(re.)</td>
<td>92</td>
<td>92</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module</th>
<th>PPA States</th>
<th>PPA Oper.</th>
<th>PPA Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framer</td>
<td>4</td>
<td>13</td>
<td>4</td>
</tr>
<tr>
<td>Monitor</td>
<td>2</td>
<td>9</td>
<td>2</td>
</tr>
</tbody>
</table>
Conclusion

• **Property Driven Development (PDD):**
  – Results in a formally sound correct-by-construction design
  – No formal verification knowledge required

• In practice, PDD is based on:
  – The provided open-source tool SCAM
  – State-of-the-art property checker

→ Shifting global design and verifications to the ESL!