Shaping Formal Traces without Constraints: A Case Study in Closing Code Coverage on a Crypto Engine using Formal Verification

David N. Goldberg
Principal Engineer
Ubicom, Inc.

Adriana Maggiore
Principal Engineer
Ubicom, Inc.

David J. Simpson
Director of Verification
Ubicom, Inc.
Overview

• Verification of Ubicom IP8000
• Code Coverage and Unreachability Analysis
• Crypto Engine and Missing Test Cases
• Closing the Loop:
  – Combining Formal Analysis and Simulation Results
• Summary
Verification of Ubicom IP8000

• Ubicom IP8000
  - 800MHz 12-threaded embedded CPU
  - Single non-stalling pipeline, memory-to-memory ISA
  - Several new features: MMU, BTB, FPU, multiple outstanding cache misses, DDR3, PCIe Gen-2, USB3.0

• Verification Strategy
  - Small team, large design: Pick the best tool for the job!
    • Directed and Constrained random tests (using DRAG)
    • ABV and Formal Verification
    • Off-line comparison to golden reference model
    • Coverage Analysis
## Coverage Analysis

### Initial Block Coverage

<table>
<thead>
<tr>
<th>Block Coverage</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>93% (13710/14677)</td>
<td>IP800_core (cumulative)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Immediate sub-instances</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>93% (5303/5679)</td>
<td>Unit1</td>
</tr>
<tr>
<td>83% (59/71)</td>
<td>Unit2</td>
</tr>
<tr>
<td>78% (142/182)</td>
<td>Unit3</td>
</tr>
<tr>
<td>96% (880/915)</td>
<td>Unit4</td>
</tr>
<tr>
<td>78% (757/969)</td>
<td>Unit5</td>
</tr>
<tr>
<td>96% (1328/1378)</td>
<td>Unit6</td>
</tr>
<tr>
<td>93% (255/273)</td>
<td>Unit7</td>
</tr>
<tr>
<td>96% (3705/3852)</td>
<td>Unit8 (including crypto engine)</td>
</tr>
<tr>
<td>92% (649/701)</td>
<td>Unit9</td>
</tr>
<tr>
<td>96% (561/586)</td>
<td>Unit10</td>
</tr>
<tr>
<td>100% (58/58)</td>
<td>Unit11</td>
</tr>
<tr>
<td>100% (5/5)</td>
<td>Unit12</td>
</tr>
<tr>
<td>100% (6/6)</td>
<td>Unit13</td>
</tr>
<tr>
<td>100% (2/2)</td>
<td>Unit14</td>
</tr>
</tbody>
</table>

967 uncovered targets

How many of those are reachable?
Unreachability Analysis

• Formal proof that code is unreachable
  - Deadcode Check of Formal Verification
• Analyze uncovered blocks only, save resources!
  - Extract uncovered blocks from coverage log
  - Convert to formal verification deadcode checks
  - Create coverage filter for proven dead blocks
  - This flow now part of Cadence IEV
    • Not available at the time of the project, we used Cadence IFV
• Review results
  - Is unreachable a design bug?
• No Formal Constraints
  - Look for structural unreachable
## Coverage after Unreachability Analysis

<table>
<thead>
<tr>
<th>Block Coverage</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>97% (13710/14098/579)</td>
<td>IP800_core (cumulative)</td>
</tr>
<tr>
<td><strong>Immediate sub-instances</strong></td>
<td></td>
</tr>
<tr>
<td>98% (5303/5393/286)</td>
<td>Unit1</td>
</tr>
<tr>
<td>91% (59/65/6)</td>
<td>Unit2</td>
</tr>
<tr>
<td>88% (142/162/20)</td>
<td>Unit3</td>
</tr>
<tr>
<td>99% (880/886/29)</td>
<td>Unit4</td>
</tr>
<tr>
<td>88% (757/856/113)</td>
<td>Unit5</td>
</tr>
<tr>
<td>97% (1328/1365/13)</td>
<td>Unit6</td>
</tr>
<tr>
<td>93% (255/273)</td>
<td>Unit7</td>
</tr>
<tr>
<td><strong>98% (3705/3768/84)</strong></td>
<td>Unit8 (including crypto engine)</td>
</tr>
<tr>
<td>95% (649/680/21)</td>
<td>Unit9</td>
</tr>
<tr>
<td>97% (561/579/7)</td>
<td>Unit10</td>
</tr>
<tr>
<td>100% (58/58)</td>
<td>Unit11</td>
</tr>
<tr>
<td>100% (5/5)</td>
<td>Unit12</td>
</tr>
<tr>
<td>100% (6/6)</td>
<td>Unit13</td>
</tr>
<tr>
<td>100% (2/2)</td>
<td>Unit14</td>
</tr>
</tbody>
</table>

- 388 uncovered targets (down from 967!)
- 346 declared unreachable by the designer
- 42 to be covered
- 5 of those in the crypto engine
Crypto Engine and Missing Tests

- IP8000 accelerates network security protocols, such as IPSEC, VPN, SSL
- Cryptographic algorithms
  - AES, MD5, DES, 3DES, SHA
- Uncovered blocks in 3DES

DES Encryption

Feistel “F” function

Uncovered blocks
Strategies for Missing Tests

- More Random Simulation
  - consume simulation cycles
  - slower due to coverage collection
  - uncertain outcome
- Modify the C-reference model
  - time consuming and one-time throwaway work
- Reverse engineering from internal values to inputs
  - difficult and highly time consuming
- Formal Analysis
Formal Analysis - first attempt

- Witness from the deadcode check
Formal Analysis - first attempt

- Witness from the deadcode check
  - no environment description

Adriana Maggiore, Ubicom, Inc.
Formal Analysis - first attempt

• Witness from the deadcode check
  – no environment description
  – provides input values at `des.des_f.des_s1` level
  – we need input values at `des` level
  – line reached due to value in an uninitialized register
  – result cannot be used to create a test

• Better witness needs better environment modeling, however
  – no assertions for this block
  – new to the formal engineer
  – cost to understand functionality, environment and create SVA constraints
Getting the Environment from Simulation

- Look at existing simulation results

DES computation:
- Starts with `des_start`
- Lasts 16 cycles
- Ends with `des_done`
DES timing as SVA sequence

cover_b1_08: \textbf{cover property}
(@posedge clk_core)
(des_start && !des_stop
  ##1 !des_start && !des_stop [*16]
  ##1 !des_start && des_stop)
intersect
  (##1 (des.des_f.des_s1.b1[5:0] == ‘h08)[=1]));
DES timing as SVA sequence

\texttt{cover\_b1\_08: cover property} \\
\texttt{(\texttt{@posedge clk\_core})} \\
\texttt{(des\_start \&\& !des\_stop)} \\
\texttt{\hspace{1cm}##1 !des\_start \&\& !des\_stop [*16]} \\
\texttt{\hspace{1cm}##1 !des\_start \&\& des\_stop)} \\
\texttt{intersect} \\
\texttt{\hspace{1cm}##1 (des.des\_f.des\_s1.b1[5:0] == ‘h08)[=1])} \\
\texttt{intersect} \\
\texttt{\hspace{1cm}(des\_key[63:0] == ‘hdeadbeef12345678)[*])};
Cover Witness

- Provides values of `des_in` and `des_key` that can be used to create new test cases.
Tuning formal traces

• DES case study was simple
  - Function with no illegal inputs
  - Timing of the computation only constraint

• In general, trial-and-error approach to deal with:
  - Under-constraining
    • Compare current formal trace against desired (simulation) trace
    • Learn and capture more environment behavior
  - Over-constraining
    • “Failed” cover statement, no trace available
    • Relax part of the behavior description, until a trace is obtained
    • Find out why it contradicts the desired behavior
Closing the loop

• Modify an existing test with the input values provided by the formal trace
• Calculate the expected results with DES calculator
• Run the test and collect coverage
• Merge the new coverage data
Summary

- Formal Verification and Simulation together provided the most cost-effective solution
  - Unreachability analysis to filter coverage results
  - Formal and simulation results to close the coverage hole
- Formal Analysis is a versatile tool that can ease several verification problems
  - ABV, post-silicon debugging, connectivity verification
- System level simulation traces show how your design and its environment work
- SVA is a powerful language
  - Use its expressiveness to quickly capture the behavior