Formal Verification of Floating-Point Hardware with Assertion-Based VIP

Nicolae Tusinschi – OneSpin Solutions
Agenda

Floating-point (FP) arithmetic
Functional verification of FP hardware
Formal verification with assertion-based VIP
Results
Conclusions
Floating-Point Arithmetic

Compared to fixed-point arithmetic

- Covers wider range of values
- No loss of precision, higher accuracy
- More complex hardware
- Notoriously hard to verify

IEEE 754 Half / Single / Double / … Precision

<table>
<thead>
<tr>
<th>Sign</th>
<th>Exponent</th>
<th>Mantissa</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 bit</td>
<td>5 / 8 / 11 / … bits</td>
<td>10 / 23 / 52 / … bits</td>
</tr>
</tbody>
</table>

Total of 16/32/64/… bits
Functional Verification

Simulation misses bugs
- Exhaustive verification is not feasible
  - One and a half engineer-years to cover all scenarios
- Many implementation-dependent corner cases

Sequential Equivalence Checking (SEC) of RTL against reference model
- Requires detailed understanding of both implementations
- Reference model (C++/SystemC) needs adaptation
- High effort, little reusability

Formal Assertion-Based VIP (ABVIP)
- Xilinx tried but ran into usability and convergence issues
- Xilinx worked with OneSpin to develop a new solution
Floating-Point ABVIP

FP ABVIP
- SystemVerilog package
- ieee... functions and data types
- No reference model required
- Minimal design knowledge by end user

OneSpin Tool
- FP ABVIP available
- Proof engines/strategies for arithmetic
- Debug with FP data types
Floating-Point ABVIP

Compliant to IEEE-754

Supports
- Half, single and double precision formats
- All the rounding modes and the exception flags
- Tininess before or after rounding
- ADD, SUB, MULT, ABS value, negation, and all comparison operations
- Conversion functions also included

Customizable
- Custom precision
- Intended deviations from standard
FP ABVIP Property Template

```verilog
property fp_add_p;
  ieee_with_flags_t expected;
  @(posedge clk)
  disable iff (~reset_n)
  (<trigger to add>, expected = ieee_add(.a(op_a), .b(op_b), .rm(rm)))
=>
  ##<latency>
  result_valid && ieee_check_result(.expected(expected),
  .actual(actual),
  .supported_flags(supported_flags));
endproperty

fp_add_a : assert property (fp_add_p);
```
Xilinx FPU

Supports addition, subtraction and multiplication

Tool found a previously undiscovered bug in the module interface constraints

General and specific scenarios assertions created
- Example: operations with signalling or quiet NaN

Design bugs previously found in simulation and emulation
- FP ABVIP found them within seconds
## Results

### Open Cores

<table>
<thead>
<tr>
<th>Operation</th>
<th># bugs</th>
<th>Setup effort</th>
<th>Runtime</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>FADD</td>
<td>0</td>
<td>30 minutes</td>
<td>52 seconds</td>
<td>Full proof</td>
</tr>
<tr>
<td>FSUB</td>
<td>1</td>
<td>5 minutes</td>
<td>1 minute to find a bug</td>
<td>Fail</td>
</tr>
<tr>
<td>FMUL</td>
<td>2</td>
<td>5 minutes</td>
<td>1 second to find a bug</td>
<td>Fail</td>
</tr>
</tbody>
</table>

### Xilinx FPU

<table>
<thead>
<tr>
<th>Operation</th>
<th># bugs</th>
<th>Setup effort</th>
<th>Runtime</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>FADD</td>
<td>0</td>
<td>4 days</td>
<td>3 minutes</td>
<td>Full proof</td>
</tr>
<tr>
<td>FSUB</td>
<td>0</td>
<td>3 days</td>
<td>1 minute</td>
<td>Full proof</td>
</tr>
<tr>
<td>FMUL</td>
<td>1</td>
<td>15 days</td>
<td>4 minutes</td>
<td>Full proof</td>
</tr>
</tbody>
</table>

Tool and FPU App familiarization and constraints setup
Formal Coverage

Metric-driven verification
- OneSpin Quantify

Answers
- How much has been verified?
- What is the next assertion to write?
- Is my design over-constrained?

User written assertions and covers

Overall coverage ~90%

Holes point to logic not contributing floating-point operations
Formal Coverage Results

### Structural Coverage Overview

<table>
<thead>
<tr>
<th>Status</th>
<th>Statements</th>
<th>Branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>99.45%</td>
<td>51.72%</td>
</tr>
<tr>
<td>R</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>U</td>
<td>0.00%</td>
<td>6.90%</td>
</tr>
<tr>
<td>OR</td>
<td>0.54%</td>
<td>0.00%</td>
</tr>
<tr>
<td>0</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>OC</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>OD</td>
<td>0.00%</td>
<td>41.38%</td>
</tr>
<tr>
<td>Sum</td>
<td>quantify targets</td>
<td></td>
</tr>
</tbody>
</table>

Verified with assertions

### Excluded Code Overview

<table>
<thead>
<tr>
<th>Code Status</th>
<th>Statements</th>
<th>Branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xu excluded by user</td>
<td>75.84%</td>
<td></td>
</tr>
<tr>
<td>Xr excluded redundant code</td>
<td>3.92%</td>
<td>1.56%</td>
</tr>
<tr>
<td>XV excluded verification code</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>0/1/U quantify targets</td>
<td>20.24%</td>
<td>9.06%</td>
</tr>
<tr>
<td>Sum total code</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Verified with assertions
Conclusions

FP ABVIP is compliant with the IEEE-754 standard
Solution is easy to set up and use
Excellent experience for Xilinx FPU project
  • Low effort
  • Uncovered corner-case bugs within seconds
  • Exhaustive verification with full proofs, within minutes
  • High coverage
Current limitations
  • No support for iterative operations (division, square root)
  • Additional effort may be required to achieve full proofs
Thanks to:
Ravi Ram, Adam Elkins, Adnan Pratama – Xilinx Inc.
Sasa Stamenkovic, Sven Beyer, Sergio Marchese – OneSpin Solutions