A Comprehensive Verification Platform for RISC-V based Processors

Emre Karabulut, Design and Verification Engineer, Yonga Technology Microelectronics R&D, Istanbul, Turkey (emre.karabulut@yongatek.com)

Berk Kisinbay, Design and Verification Engineer, Yonga Technology Microelectronics R&D, Istanbul, Turkey (berk.kisinbay@yongatek.com)

Abdullah Yildiz, Design and Verification Engineer, Yonga Technology Microelectronics R&D, Istanbul, Turkey (abdullah.yildiz@yongatek.com)

Rifat Demircioglu, Design and Verification Engineer, Yonga Technology Microelectronics R&D, Istanbul, Turkey (rifat.demircioglu@yongatek.com)
Agenda

• Introduction
• Motivation
• Proposed Methodology
• Preliminary Results
• Conclusion
• Future Work
Introduction

• RISC-V ecosystem\textsuperscript{[1]} enables SoC designers to benefit from
  – more than 100 core/SoC design and implementations
  – instruction set simulators
  – toolchains

• RISC-V ISA specification allows to extend the instruction set with specialized extensions\textsuperscript{[2,3]}

• It is estimated that there will be about 62 billion RISC-V based CPU cores in the market by 2025\textsuperscript{[4,5]}. 

Motivation

• RISC-V’s licensing model allows to reduce non-recurring engineering (NRE) costs

• There is no condition or restriction on how a RISC-V based system will be designed and implemented

• Verification of a RISC-V based SoC should not only include the verification of the core itself, but rather the verification of the whole system!
Proposed Methodology

• We present a comprehensive and UVM based RISC-V verification platform
• Our platform provides verification at both SoC, core, and block levels
• Module level verification of branch predictor and cache blocks
• Presents functional as well as stress tests in addition to standard test suites
• Provides a logging mechanism that includes committed instructions, cache transactions, branch predictor operations, etc.
• Supports debugging via OpenOCD + GDB + JTAG
Proposed Methodology (cont’d)

YONGATEK RISC-V Verification Flow
Proposed Methodology (cont’d)

• Cache simulator
  – supports different cache configurations
  – placement: direct-mapped, set-associative, fully associative
  – replacement: random, LRU, FIFO
  – request handling: blocking/non-blocking
  – error handling: ECC

• Written in C++

Cache verification flow
Proposed Methodology (cont’d)

• Branch predictor simulator
  – supports bimodal, g-share, hybrid predictor models

• Written in C++
Preliminary Results

• Regression consists of 300+ test cases
• Created about 10,000 covergroup elements, containing coverpoints and crosses
• Obtained on a customized SoC based on BOOM [24]

Code coverage and functional coverage

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Register Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Field</td>
</tr>
<tr>
<td>LOAD</td>
<td>100.0</td>
</tr>
<tr>
<td>STORE</td>
<td>96.88</td>
</tr>
<tr>
<td>BRANCH</td>
<td>100.0</td>
</tr>
<tr>
<td>JALR</td>
<td>56.25</td>
</tr>
<tr>
<td>JAL</td>
<td>96.88</td>
</tr>
<tr>
<td>LOAD-FP</td>
<td>57.29</td>
</tr>
<tr>
<td>STORE-FP</td>
<td>46.88</td>
</tr>
<tr>
<td>OP</td>
<td>100.0</td>
</tr>
<tr>
<td>OP-32</td>
<td>100.0</td>
</tr>
<tr>
<td>OP-IMM</td>
<td>100.0</td>
</tr>
<tr>
<td>OP-IMM-32</td>
<td>100.0</td>
</tr>
<tr>
<td>OP-FP</td>
<td>94.06</td>
</tr>
<tr>
<td>AMO</td>
<td>12.07</td>
</tr>
<tr>
<td>(N)MADD/(N)MSUB</td>
<td>85.68</td>
</tr>
</tbody>
</table>
Conclusion

• Our verification platform
  – can support all the base instruction set and standard extensions of RISC-V ISA
  – provides verification at SoC, core, and block levels
  – gathers different features of different verification environments under a single platform
Future Work

• Trace interface and encoder support
• JTAG VIP
• Flow management GUI
Questions
References

References (cont’d)


