Advanced RISC-V verification methodology projects

15 March 2023
Agenda

- The evolution of OpenHW CORE-V-VERIF
- Defining open standards for verification with RISC-V Verification Interface (RVVI)
- Asynchronous events
- Functional coverage
- Future direction
OpenHW group

• A not-for-profit, global organization registered in Canada and Europe.
  • Almost 100 Members and Partners from all regions of the world.
• Members (corporate & academic) collaborate in the development of open-source cores, related IP and SW.
• Primary focus is the CORE-V Family of open-source RISC-V processors.
  • Use of industrial methodology, verification and tooling to create quality IP
What is CORE-V-VERIF?

• A UVM environment for RISC-V processor verification
• Open source and available on GitHub:
  • [https://github.com/openhwgroup/core-v-verif](https://github.com/openhwgroup/core-v-verif)
• Developed by the OpenHW Group’s Verification Task Group (VTG)
  • Contributions from Silicon Labs, Imperas, EM Microelectronic
  • On-going work by Silicon Labs, Imperas, Thales, Dolphin, NXP, Intrinsix
• Created in 2019; evolving and improving ever since
• In use at many academic and commercial organizations today
First generation CORE-V-VERIF
Second generation CORE-V-VERIF
RISC-V Verification Interface (RVVI)

- Common components should have standard interfaces
- Standardize communication between RTL, testbench, and RISC-V VIP
  - **RVVI-TRACE**: signal level interface (tracer) between RTL and testbench
  - **RVVI-API**: function level interface between testbench and RISC-V VIP
  - **RVVI-VVP**: virtual verification peripherals

[GitHub Link](https://github.com/riscv-verification/RVVI)
RVVI-TRACE

https://github.com/riscv-verification/RVVI/tree/main/RVVI-TRACE

- Generically defines information to be extracted by tracer
- SystemVerilog interface
- Includes functions to handle asynchronous events
  - implements a queue to store multiple net changes during a single interval
RVVI-API

- Standard functions that RISC-V processor VIPs need to implement
- Supports a co-simulation, continuous comparison methodology
- C and SystemVerilog versions available
- Example: functions to marks registers/fields as “volatile”

https://github.com/riscv-verification/RVVI/blob/main/include/host/rvvi/rvvi-api.h

- rvviRefEventStep()
- rvviRefGprsCompare()
- rvviRefPcCompare()
- rvviRefCsrsCompare()
- rvviRefGprGet()
- rvviRefPcGet()
- rvviRefInsBinGet()
- rvviRefCsrGet()
Asynchronous events

\[
\text{IRQ} = \text{MSTATUS.MIE} \&\& \left( (\text{MIE} \& \text{MIP}) \neq 0x0 \right);
\]
Functional Coverage

- Prove all functionality of the design has been tested
  - Eg all registers used, error conditions tested
  - Far more comprehensive than just compliance testing
  - Design intent has to be well specified to allow complete verification against it

- Complexity relates to design choices, configuration options
  - Eg which ISA extensions are being used

- Hard to write by hand
  - ~1000 instructions in the RISC-V ISA (RV64GC, not including privilege modes etc)
  - Maybe 10-40 lines of SystemVerilog for each instruction...
  - That is 10,000-40,000 lines of SystemVerilog code to be written... (and be correct and working...)
ISA complexity

- **RV64ISA:**
  - Integer: 56
  - Maths: 13
  - Compressed: 30
  - FP-Single: 30
  - FP-Double: 32
  - Vector: 356
  - Bitmanip: 47
  - Krypto-scalar: 85
  - P-DSP: 318
  - 967 instructions

- **Example from RV32I:**
  - Instructions: 37
  - Covergroups: 37
  - Coverpoints total: 438

<table>
<thead>
<tr>
<th>Extension</th>
<th>Instruction</th>
<th>Covergroup</th>
<th>Coverpoint</th>
<th>Coverpoint Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RV32I</td>
<td>addi</td>
<td>addi_cg</td>
<td>cp_asm_count</td>
<td>Number of times instruction is executed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>cp_rd</td>
<td>RD (GPR) register assignment</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>cp_rd_sign</td>
<td>RD (GPR) sign of value</td>
</tr>
</tbody>
</table>

[https://github.com/riscv-verification/riscvISACOV/blob/20221117.0/documentation/RV32I_coverage.md](https://github.com/riscv-verification/riscvISACOV/blob/20221117.0/documentation/RV32I_coverage.md)
Limitations on reuse

- Interrupts
- Debug
- Pipeline
- Custom instructions
- Custom CSRs
- All of these are design specific so hard to make coverage generic
riscvISACOV

https://github.com/riscv-verification/riscvISACOV

RVVI allows use of Functional Coverage with any RISC-V core
Future Work

- Advanced RISC-V Verification Methodologies (ARVM) projects:
  - [https://github.com/openhwgroup/programs/tree/master/TGs/verification-task-group/projects](https://github.com/openhwgroup/programs/tree/master/TGs/verification-task-group/projects)
- Enhanced functional coverage for Privilege specification, MMU
- Enhanced functional coverage for hazards, microarchitectural features
Conclusions

• CPU DV for RISC-V is a new problem for many SoC designers

• An open, standardised approach to verification improves re-use and enables faster integration of third party verification IP
  • There’s no downside to adopting RVVI for RISC-V DV

• Functional coverage is essential to prove the design matches intent
Thank you!

Jon Taylor - jont@imperas.com
Visit our booth: Hall: 4A, Booth: 4A-620