Jump start your RISCV project with OpenHW

Mike Thompson (OpenHW Group) Jingliang Wang (Futurewei Technologies)
Steve Richmond (Silicon Labs) Lee Moore (Imperas Software Ltd.)
David McConnell & Greg Tumbush (EM Microelectronic-US)
Agenda

• OpenHW Verification Environment
• Reference Model
• Step and Compare
• Conclusion
• Future Work
• Questions
OpenHW Verification Environment

- Provides a robust, comprehensive simulation environment for the CV32E40P RV32IMCZifencei processor
- Freely available on github at openhwgroup/core-v-verif
- Industrial-grade verification
  - UVM environment
  - Runs on any commercial SystemVerilog-compatible simulator
  - Complete code coverage
  - Well-defined comprehensive functional coverage
  - Open and complete verification plans
CORE-V-VERIF Testbench

Verification Plan is the spec for functional coverage model

Most test-runs use “corev-dv”, extended from riscv-dv

Same test-program on core and ISS

Toolchain invoked by Makefile

GCC compile

elf2hex

Random Instruction Generator
corev-dv

SVA

Re-use Assertions from Design team

Compare all CSRs, GPRs and PC

Functional Coverage

Debug Agent

Interrupt Agent

Scoreboard

Async Debug and Interrupts

CV32E40P RTL

SVA

mm_ram

dp_ram

vp

CV32E40P RTL

Nicholas Brown

Toolchain invoked by Makefile

GCC compile

elf2hex

Random Instruction Generator
corev-dv

SVA

Re-use Assertions from Design team

Compare all CSRs, GPRs and PC

Functional Coverage

Debug Agent

Interrupt Agent

Scoreboard

Async Debug and Interrupts

CV32E40P RTL

SVA

mm_ram

dp_ram

vp

CV32E40P RTL

Nicholas Brown
Environment

• SystemVerilog Components
  • tracer: Reports instructions for checking and register writebacks
  • step_and_compare: Manages the ISS and checks functionality
  • mmram: OBI I/D port stimulus and virtual peripherals
  • interrupt_assert: Properties for interrupt coverage/checking
  • debug_assert: Properties for debug coverage/checking

• UVM Agents
  • obi: Monitor/functional coverage for OBI
  • debug: Random stimulus of external debug requests
  • interrupt: Random stimulus for external interrupts
  • rv32isa_covg
    • Coverage of all RV32IMCZifencei instructions
    • Includes interrupt and debug requests with instruction execution
Tests

• Directed and random tests supported
  • Directed/custom tests
    • Assembly or C
    • BSP package provides test utilities
  • Random tests
    • Built on Google riscv-dv to generate random tests
    • Fully randomize external iterations during random test
      • Interrupt, debug requests
      • OBI I/D RAM stalls

• YAML test specifications
  • Control simulation
    • Run-time plusargs
  • Control random test generation
    • Knobs to instruction set generator
• Customization layer based on Google riscv-dv
• riscv-dv is included via a *git clone* by make when generating a test
• OpenHW corev-dv extensions
  • Custom configuration for cv32e40p
  • M-mode register fields for cv32e40p interrupts
  • Nested interrupt support
  • Debug ROM stack for more robust debug tests
  • M-mode CSR stimulus with interrupts
  • Numerous directed streams to achieve better ISA coverage, especially around jumps and branches
Reference Model

- Functional Coverage
- Debug Agent
- Interrupt Agent
- Scoreboard
- SV - DPI Wrapper

© OpenHW Group
Reference Model Standard Config

- Reference model is central to the DV plan and overall verification quality
- Imperas reference model covers the envelope of the full RISC-V specification
- OVP model is a binary shared object of a RISC-V CPU model
- Encapsulated into a SystemVerilog module, using SystemVerilog DPI
- Instanced in SystemVerilog design or testbench as a module
- Control interface
- State Interface
Custom Extensions
- Control & State Registers (CSR)
- Instructions

Example: Debug Specification
- Highly configurable with options, and customizable
- Many possible subset selections.
- User configurable
Reference Model
Instruction Execution

- Instruction execution continued to retirement
  - State update
- Instruction execution discontinued by exception
  - Synchronous
    - Misaligned load/store
    - Illegal instruction (privileged, unsupported)
  - Asynchronous
    - Interrupts
    - Debug-Request
- Instruction execution to halt
  - WFI
Reference Model Encapsulation
Reference Model Debug/Analysis Capabilities

- **RTL Debug**
- **S/W Debug**
- **S/W Trace**
- **S/W Analysis**

**BIU (Fetch, Load/Store)**

**Code**  
**Data**

**imperas Reference Model**

**Custom Ext**

**SV - DPI Wrapper**

**Async Events**

**State**

**Interrupts**  
**Debug**

**GPR**  
**CSR**

**MODE**

**DEBUG**

**accellera SYSTEMS INITIATIVE**
Step and Compare

- Imperas Reference Model (RM) used in step-and-compare mode
- RTL and RM in sync at an instruction level
- Invaluable for debug
- No modifications to RTL
- Testbench keeps RTL and RM in sync
- Tracer flags testbench that RTL completed an instruction
• Implemented as a 4-state State Machine
Step and Compare Example

<table>
<thead>
<tr>
<th>12c: csrw mepc, a2</th>
<th>converted by tracer</th>
<th>130: jal ra, 961c</th>
</tr>
</thead>
</table>

| 12c: csrrw x0, x12, 0x341 | 130: jal x1, 38124 |

<table>
<thead>
<tr>
<th>Clk_to_RTL state</th>
<th>STEP_RTL</th>
<th>STEP_OVP</th>
<th>STEP_RTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step_RTL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTL_retire</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Step_RM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RM_retire</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compare</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>insn_pc[31:0]</td>
<td>0x000012C</td>
<td>0x0000130</td>
<td></td>
</tr>
<tr>
<td>insn_disas</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>csrrw</td>
<td>x0, x12, 0x341</td>
<td>jal x1, 38124</td>
<td></td>
</tr>
</tbody>
</table>
Compare

• Compare is done in the COMPARE state
  • PC
  • GPRs
  • CSRs
GPR Compare

Issue: Instructions using the EX WB stage update GPR after the RTL retire signal

Input:

- lw x8, 24(x2) completes
- x8 updates

Output:

Compare fails
GPR Compare

**Fix:** Use tracer queue *insn_regs_write*
- Contains address/data of any GPR updated in EX WB stage

```plaintext
insn_regs_write[0].address=8
Insn_regs_write[0].data=0x364C
```

 lw x8, 24(x2) completes  Compare succeeds
CSR Compare

- At RTL Retire CSRs have updated and are probed directly
- At RM Retire predicted CSRs written to array CSR
- Array CSR traversed at compare event

```verilog
foreach(iss_wrap.cpu.CSR[index]) begin
    csr_val = 0;
    case (index)
        "misa" : csr_val = `CV32E40P_CORE.cs_registers_i.MISA_VALUE;
        "mie"  : csr_val = `CV32E40P_CORE.cs_registers_i.mie_q;
        ...
    endcase
    check_32bit(.compared(index),
           .expected(iss_wrap.cpu.CSR[index]),
           .actual(csr_val));
```
Conclusion

• CV32E40P RISCV CPU is fully verified and open source
• Functional and Code Coverage is 100%
• All tests pass
Future Work

• CVA6
• CV32E4
• CV32E2
• Common Tracer interface for RTL and RM, similar to RVFI
• Google riscv-dv generator as a UVM component

GET INVOLVED!
Questions?