Advanced Verification Suite for RISC-V Cores*

Murat Tökez[1]†, Merve Eyüboğlu[1], Ibrahim Mouamar Ali Ahmed[1],
Melike Atay Karabalkan[1], Berna Ors[1,2]

1Electra IC, Turkey
2Istanbul Technical University, Turkey

Abstract

We have developed a verification enviroment called ElectraIC Advanced Verification Suite (EAVS) for verification of any RISC-V core. EAVS includes an Instruction Set Simulator (ISS), YAML configuration files, and a RISC-V Core UVM Testbench. The RISC-V Core UVM Testbench contains a RISC-V Core, referred to as the Design Under Test (DUT), along with an Instruction Generator, Compiler, and a RISC-V Core Base Test.

Introduction

ElectraIC Advanced Verification Suite (EAVS) includes an Instruction Set Simulator (ISS), YAML configuration files, and a RISC-V Core Universal Verification Methodology (UVM) Testbench, as shown in Fig. 1. The RISC-V Core UVM Testbench contains a RISC-V Core, referred to as the Design Under Test (DUT), along with an Instruction Generator, Compiler, and a RISC-V Core Base Test.

ISS operates as the core's reference model, acting as a golden model to determine whether the core's executed instructions are correct. Spike is a RISC-V ISA simulator officially released by the RISC-V International Foundation, capable of simulating one or more RISC-V harts [1].

The instruction set generator (ISG) produces assembly file which include the configured instructions according to the targeted test scenarios, where the instructions are randomly generated in accordance with the scenario's constraints. Examples of various instruction generators include Force-riscv [2], an ISG for the RISC-V ISA from OpenHW Group which supports all instructions of RV32GC, and Google riscv-dv [3].

The RISC-V Formal Interface (RVFI) [4] is a interface designed to facilitate the formal verification of RISC-V processors. It provides a comprehensive set of signals that capture the internal state and behavior of a processor during the execution of each instruction.

Advanced Verification Suite for RISC-V Cores

The primary objective of this study is to address all stages of the verification flow in a manner that complies with standards and remains configurable. With this goal in mind, the system has been designed to accommodate different RISC-V cores; as an example, the verification of the cv32e40p core is presented. Figure 1 illustrates the overall architecture and the interactions among its components.

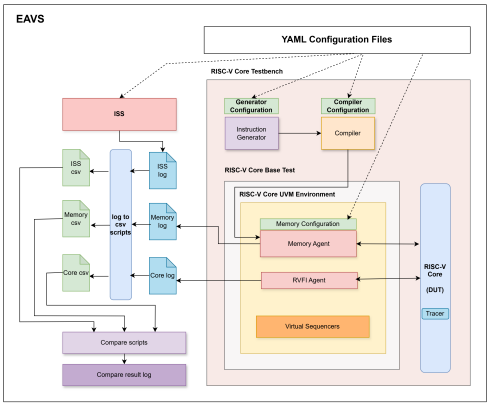

Figure 1: *EAVS RISC-V Core Verification Environment*

Components of the Verification Environment

ISS We have subsequently converted the complex log format obtained from Spike into a ".csv" file. Currently, Spike runs externally rather than being integrated into the UVM environment.

YAML Configuration YAML configuration files ensure that test parameters, such as boot_address, virtual peripheral adresses, ram address width etc. , and scenarios like floating-point tests, loop tests, and CSR tests, as well as memory mapping used during verification, are managed in a consistent manner. We have configured the memory map information to avoid any conflicts with Spike's embedded memory map. Having parameters that can be flexibly modified allows users to quickly integrate different RISC-V cores or custom extensions into the verification flow.

Instruction Generator and Compiler In this study, COREV-DV[5]—layered on top of Google riscv-dv[3]—has been used. We propose EAVS-DV as an enhancement to COREV-DV.

*This work was supported by TUBITAK 3231156 project.
†Corresponding author: murat.tokez@electraic.com

Table 1: *RVFI Agent Log Output Snippet*

TIME	RVFI	CYCLE	ORDER	PC	INSTR	M	RS1	RS1_DATA	RS2	RS2_DATA	RD	RD_DATA
146961.000 ns	RVFI	48981	11472	8000986c	80400bb7	M	x0	00000000	x4	00000000	x23	80400000
146970.000 ns	RVFI	48984	11473	80009870	00001497	M	x0	00000000	x0	00000000	x9	8000a870
146973.000 ns	RVFI	48985	11474	80009874	8104a483	M	x9	8000a870	x16	ffffffff	x9	075bcd15
146979.000 ns	RVFI	48987	11475	80009878	009ba023	M	x23	80400000	x9	075bcd15	x0	8000a870
146982.000 ns	RVFI	48988	11476	8000987c	30405073	M	x0	00000000	x4	00000000	x0	00000000

Table 2: *Memory Agent Log Output Snippet*

TIME	OBI	RW	ADDR	BE	DATA
146946.000 ns	OBI	W	80500000	f	0000000a
146970.000 ns	OBI	R	8000a080	f	075bcd15
146976.000 ns	OBI	W	80400000	f	075bcd15

The assembly file generated by the instruction generator is compiled using a compiler, producing a machine language file. This file is then written into the memory defined within the memory agent and subsequently delivered to the core.

RISC-V Core UVM Environment

The environment has been built following the UVM testbench architecture [6], providing detailed monitoring of the core's program counter (PC), instruction and data memory, and register file behavior. Within this environment, components such as the Memory Agent, RVFI Agent, Scoreboard, Virtual Sequencers, and Coverage work together to ensure comprehensive verification.

The Memory Agent includes both instruction and data memory. It has two primary duties: driving instructions and data to the DUT using the sequencer and driver, and observing the program counter (PC) and data addresses from the DUT with the monitor.

The RVFI Agent is a passive agent that continuously monitors the connected RVFI, which can be configured to include any signal in the pipeline stages.

The cv32e40p tracer exhibits several issues that impact the accuracy of instruction decoding and logging. The srai instruction is improperly decoded. Additionally, all compressed instructions are logged with incorrect binary formats, as their uncompressed counterparts instead of the compressed format. Moreover, lui and auipc instructions append three zeros to the LSB of their immediate values, leading to operand comparison errors. Lastly, all pseudo-instructions are decoded in their normal form, which diverges from Spike's convention. For these reasons, the cv32e40p tracer was not utilized in this work.

Virtual Sequencers coordinate multiple sequencers across different agents within the same test environment.

Tests and Results

In this study, a wide range of randomly generated tests was applied to the `cv32e40p` core, and the resulting data were collected. Output snippets from the RVFI and Memory Agent monitors are provided in Tables 1 and 2.

The same randomly generated tests were also executed on the Spike ISS, with the results shown in Table 3.

Table 3: *Spike Log Output Snippet*

```
core   0: 0x8000986c (0x80400bb7) lui     s7, 0x80400
core   0: 3 0x8000986c (0x80400bb7) x23 0x80400000
core   0: 0x80009870 (0x00001497) auipc   s1, 0x1
core   0: 3 0x80009870 (0x00001497) x9  0x8000a870
core   0: 0x80009874 (0x8104a483) lw      s1, -2032(s1)
core   0: 3 0x80009874 (0x8104a483) x9  0x075bcd15 mem 0x8000a080
core   0: 0x80009878 (0x009ba023) sw      s1, 0(s7)
core   0: 3 0x80009878 (0x009ba023) mem 0x80400000 0x075bcd15
core   0: 0x8000987c (0x30405073) csrwi   mie, 0
core   0: 3 0x8000987c (0x30405073) c772_mie 0x00000000
```

We have converted the log files to a CSV format, a snippet of which is presented in Table 4. In the Spike ISS `.log` file, memory contents are displayed only for load instructions; therefore, both data and addresses are compared for load instructions, whereas only addresses are compared for store instructions. The *mode* indicates the privilege level at which the program is running: this corresponds to the `M` column in the RVFI Agent log and to lines labeled with `3` in the Spike ISS log.

Table 4: *Spike and Core Random Arithmetic Test Results*

pc	instr	gpr	mem	csr	binary	mode	instr	operand
0x8000986c	lui	s7:0x80400000			0x80400bb7	3	"lui s7 0x80400"	"s7 0x80400"
0x80009870	auipc	s1:0x8000a870			0x00001497	3	"auipc s1 0x1"	"s1 0x1"
0x80009874	lw	s1:0x075bcd15	mem[0x8000a080]		0x8104a483	3	"lw s1 -2032(s1)"	"s1 1 -2032"
0x80009878	sw		mem[0x80400000]:0x075bcd15		0x009ba023		"sw s1 0(s7)"	"s1 s7 0"
0x8000987c	csrwi			mie:0x00000000	0x30405073		"csrwi mie 0"	"zero mie 0"

After comparing the generated CSV files, we observed that each random test produced by `eavs-dv` successfully executed on the `cv32e40p` RISC-V core.

Conclusion

We have designed a verification environment for RISC-V Cores. The advantage of our design is mainly our proposal as EAVS-DV which is an enhancement over COREV-DV. All fixed address spaces in COREV-DV have been parameterized in EAVS-DV to enable compatibility with any DUT and Spike that has memory address limitations.

References

[1] RISC-V International. *Spike: The RISC-V ISA Simulator*. `https://github.com/riscv/riscv-isa-sim`. Accessed: 2024-06-01. 2020.

[2] OpenHW Group. *Force-riscv: A RISC-V Instruction Generator*. `https://github.com/openhwgroup/force-riscv`. Accessed: 2024-06-01. 2021.

[3] Google. *riscv-dv: RISC-V Directed Verification*. `https://github.com/google/riscv-dv`. Accessed: 2024-06-01. 2020.

[4] SymbioticEDA. *RISC-V Formal Interface (RVFI)*. `https://github.com/SymbioticEDA/riscv-formal`. 2021.

[5] OpenHardware. *corev-dv: Library of extensions to the Google riscv-dv instructuion stream generator*. `https://docs.openhwgroup.org/projects/core-v-verif/en/latest/corev_dv.html`. 2021.

[6] Accellera Systems Initiative. *UVM User's Guide*. `https://accellera.org`. 2015.