RISC-V Instruction Generator Comparison

The table below is a comparison between the Nvidia RISC-V random instruction generator and the Google “riscv-dv” generator. Items under the “Nvidia Generator” column that are marked “Unknown” are questions for Nvidia. Unknown items under the “Google Generator” column will be investigated by OpenHW.

No.	Criteria	Nvidia Generator	Google Generator	Comment	S
1	Implementation	SystemVerilog/ UVM	SystemVerilog/ UVM	**Note**: neither generator is a uvm_env or agent	0
2	OpenHW Team Knowledge	High	Low	The Nvidia team that developed the generator is providing ACs	+1
3	Top-level Class	uvm_sequence	**uvm_object** (riscv_asm_program_gen is top-level)	Ideally either uvm_env or uvm_agent.	+1
4	Transaction Class	uvm_sequence_item	uvm_object	uvm_sequence_item is preferred.	+1
5	Functional Coverage Model	Yes	Yes	OpenHW project will need to extend this for PULP instructions	0
6	Output Format	UVM Sequence hexfile (optional)	ASM file	ASM output requires use of toolchain to compile before simulation t=0.	+1
7	Randomization of EEI	Yes	Yes	See below	0
8	Randomization of GPRs	Yes	Yes	See below	0
9	Randomization of CSRs	Yes	Yes	See below	0
10	Forward Branch Randomization	Supported via random sequence	Supported via random sequence		0
11	Backward Branch Randomization	Supported via random sequence	Supported via direct sequence		+1
12	LSU Random	Random	Directed sequence		+1
13	CSR Random	Dynamically changing	Not change	Need more information.	+1
14	Exceptions	All kinds of	Some exceptions	Need a list of	+1

		exception		exceptions.	
15	Generation Mode	Pre-run	Pre-run	See below. On-the-fly is preferred	0
16	Simulation Performance	~10 instructions/ sec	~200 instruction/ sec	Performance is less of an issue with on-the-fly generation.	-1
17	Global Configuration	Separate control	Central control	More information needed.	-1
18	Configurable Target	No	Yes	This is a “nice to have” for OpenHW as we only need to support a small number of cores at any one time.	-1
19	Simulation Interoperability	Developed under VCS. Not known if it was used with another simulator.	Yes	OpenHW generator must compile/run using Metrics, Cadence, Mentor and Synopsys simulators.	-1
20	Opcode Control	Yes	Yes	Would like the ability to select specific instruction sets (e.g. Integer)	0
21	Ability to generate malformed Instructions	Yes	Yes	Multiple types: - illegal opcodes - reserved opcodes - reserved fields	0
22	Ability to generate illegal Instruction sequences	Unknown	Yes (via direct sequence)	Are there any illegal instruction sequences in the RISC-V ISA?	-1
23	Number of Harts	No	Configurable	All current CORE-V cores support a single hart.	0
24	Physical Memory Protection	Unknown	Yes	Not needed for CV32E40P	0
25	Privileged Modes	U/S/M	U/S/M	Only Machine mode supported by CV32E40P	0
26	XLEN control	Unknown	32, 64	Must-have	-1
				Total	+2

Scoring

The “S” column in the table above indicates a score for each criteria (row) in the table:

- “0” indicates that both generators score equally for the criteria.
- “-1”: indicates that the Google generator scores better for the criteria.
- “+1”: indicates that the Nvidia generator scores better for the criteria.

The sense of the sum of the scores (positive or negative) indicates the overall score. With a score of +2 the Nvidia generator is favoured. This is significant as 11 criteria out of 26 are scored 0 (meaning the two generators are perceived to be equivalent for the specific criteria). A score of +2 out of the remaining 15 criteria indicates a solid preference for the Nvidia generator.

It is recognized that the scoring is somewhat objective and that some criteria could be weighted more than others. For example, criteria 1..6, 15 and 26 are of high importance to the OpenHW projects. If these were weighted higher than other criteria, the score for the Nvidia generator would be higher.

Additional/Background Information

Some cells in the Table above contain the phrase “see below” to indicate that additional information is required. This is provided below.

Generation Mode

In constrained-random verification environments there are two modes of generating stimulus. **Pre-run** generation generates all of the stimulus at one time, typically at t=0 and before power-on-reset. **On-the-fly** generates stimulus as it is consumed by the DUT. An on-the-fly generator does not generate a new transaction unless a specific criteria is met. Examples of this include completion of an instruction read on the instruction memory interface or the instruction-retire signal.

On-the-fly generation is generally preferred because:

- performance is less of an issue with on-the-fly generation because the generator is paced by the speed of the RTL.
- The generator can use the current state of the DUT and/or verification environment as constraints.

Randomization of EEI

For the purposes of this discussion, the term Execution Environment Interface or EEI is defined[1] to be:

- the initial state of the program;
- supported privilege modes;
- the accessibility and attributes of memory and I/O regions;
- the handling of interrupts or exceptions raised during execution.

1 The RISC-V ISA definition also covers multiple harts and “the behaviour of all legal instructions”. Both of these are considered outside the scope of the random instruction generators.

Ideally, the generator should model these attributes of the EEI, randomize them prior to generating any instructions sequences, and use them as constraints to the randomized instructions.

Randomization of CSRs

It is expected that DUT CSRs will be in a known state after after power-on-reset of the core. However, preconfiguration randomization of CSRs (and backdoor loading of DUT CSRs to match) would be a useful way of exercising the core from a “non clean”, but still legal, initial state.

Randomization of GPRs

The RISC-V ISA states that the GPRs are in an unknown state after power-on-reset of the core (the exception of course is x0). Preconfiguration randomization of GPRs (and backdoor loading of DUT GPRs) would be a useful way of exercise this.