Processor Validation Guide

This document provides miners with a comprehensive explanation of how their RV32I processor design will be evaluated using the provided EDA server. The evaluation process ensures that submitted designs meet functionality, performance, and area requirements. By following this guide, miners can also locally verify and benchmark their designs before final submission.

Design.zip and Evaluator.zip

The evaluation setup requires two inputs: **design.zip** and **evaluator.zip**. The **design.zip** file must contain the miner's processor RTL implementation along with a proper `rtl.f` file. The **evaluator.zip** is provided as part of this challenge and contains all necessary testbenches, memory models, scripts, and configuration files required for validation.

Evaluator Package Structure

The **evaluator.zip** includes three subdirectories:

- **gateway/**: Contains `weights.json` for scoring configuration.
- **verilator/**: Contains testbench files, memory models (`imem.sv`, `dmem.sv`), tracer module, Makefiles, Python scripts, and the complete test suite.
- **openlane/**: Contains `run.py`, `flow.tcl`, `config.json`, and `constraints.sdc` for synthesis and timing analysis.

Directory Structure and Flow

Inside the **verilator/** directory, you will find:

- **tests/**: Contains the five categories of test programs used for validation:
    1. `riscv_arithmetic_basic_test/`
    2. `riscv_loop_test/`
    3. `riscv_rand_jump_test/`
    4. `riscv_jump_stress_test/`
    5. `riscv_mmu_stress_test/`

- Each folder includes 100 assembly tests (`.S`) with corresponding compiled memory images (`imem_X.mem` and `dmem_X.mem`) and Spike golden traces for reference.

- **scripts/**: Includes helper Python tools such as `core_log_to_trace_csv.py`, `instr_trace_compare.py`, and `regression_summary.py`. These are used to convert simulation logs to CSV format, compare them with Spike traces, and compute scores.

- **Makefiles**:

    1. `Makefile`: Handles compilation of the Verilator simulation.
    2. `Regression.mk`: Runs the complete regression flow across all test sets.

- **run.py**: The central controller. It:

    1. Reads `rtl.f` from the design directory and `tb_files.f` from the testbench.
    2. Merges these into a single `verilator.f` file containing all design and testbench sources.
    3. Builds the Verilator simulation with `+define+tracer` to enable tracing.
    4. Runs each test case (loading `imem_X.mem` and `dmem_X.mem`).
    5. Saves the generated log files into the corresponding test directory (e.g., `tests/riscv_arithmetic_basic_test/`).

6. Converts the logs to CSV format and compares them with Spike reference traces.
7. Produces a regression summary (`regression_summary.csv`) with functionality score, instruction counts, and cycles.

This flow ensures every miner's design is tested consistently across all categories. The results include pass/fail status for each instruction, total instructions executed, cycles taken, and the final functionality score.

Functional Verification

Functional correctness is validated using Verilator with the tracer enabled. The tracer captures retired instructions and generates logs that are compared against Spike reference traces. The evaluation includes **500 assembly test programs** across five categories, each with 100 tests:

- Arithmetic Basic Test
- Loop Test
- Random Jump Test
- Jump Stress Test
- MMU Stress Test

During evaluation, each test is executed by loading instruction and data memory images (`imem.mem` and `dmem.mem`). The tracer log is compared with the Spike golden trace to compute the functionality score. In addition to pass/fail results, the Verilator log provides cycle counts, enabling the calculation of **Instructions Per Cycle (IPC)**.

Performance Evaluation

Processor performance is measured in terms of **Instructions Per Second (IPS)**. This is derived from two key metrics:

1. **IPC (Instructions per Cycle)** – obtained from Verilator simulation logs.
2. **Maximum Frequency** – obtained from OpenLane synthesis results.

By multiplying IPC with the maximum achievable frequency, the effective instruction throughput (IPS) of the processor is computed. This metric reflects the overall performance of the submitted design.

Area Evaluation

OpenLane synthesis provides the estimated silicon area of the processor in μm². This value is compared against a reference target to compute the area efficiency score. Designers are encouraged to optimize their architecture for both frequency and area to maximize their ranking.

Local Verification Flow

Miners can test their designs locally using the provided evaluator.zip. They just have to submit the design.zip and the evaluator.zip attached with the challenge. And they will get the final result in the form of functionality score, performance score, area score and the overall score that is computed from functionality, performance and area.

At the backend following steps are done in the verilator and the openlane api's:

1. Run Verilator with tracer enabled (`+define+tracer`) to generate a log file.
2. Use `core_log_to_trace_csv.py` to convert the log into CSV format.
3. Compare the generated CSV against Spike reference traces using `instr_trace_compare.py`.
4. Run `regression_summary.py` to compute functionality scores and IPC.
5. Run OpenLane synthesis using `run.py` to obtain area and timing estimates.

Note: Exactly the same steps will be done by the validators as well later once the miners actually submit their solution design.zip. Only the test cases will be different, this means the functionality score may change (but shouldn't differ by more than 1% in the functionality score), but the area, and the performance score will stay exactly the same.

Important Notes

- Validator test cases will differ from the provided 500 tests but will follow the same structure and methodology.
- The tracer must be correctly instantiated in the datapath and only enabled during simulation. Tracer logic must be wrapped under `ifdef tracer` to avoid impacting synthesis results.
- Designs that fail functionality (score < threshold), do not include the tracer, or fail synthesis will receive a total score of zero.
- Scoring is based on functionality, performance (IPS), and area. Power is not considered in this challenge.