Modular SAIL: dream or reality?

Petr Kourzanov, Anmol Anmol

Imec DSRD Compute Systems Architecture (CSA)
Kapeldreef 75, 3001 Leuven, Belgium. {first}.{last}@imec.be

Artefacts:

Know more about us:

INTRODUCTION

Unlocking the Full Potential of RISC-V ISA Modularity: Addressing Compositionality

Challenge: To fully benefit from RISC-V ISA modularity, the community must address compositionality. This involves extending beyond module specifications. It encompasses broader aspects of the RISC-V development flow, including *emulation*, *simulation*, *testing* and *verification*.

Our Proposal: Modular SAIL

- An approach to infuse compositionality into the SAIL-RISCV golden model.
- **Feasibility**: We demonstrate that it is possible to adapt the flow (and in future, the SAIL compiler itself) to support modules at the emulator level.
- **Results**: Comparative study of the pluggable emulator's overheads
- for both static and dynamic bindings
- Same functional behavior as the original monolithic emulator (RISC-V ISS)

METHODOLOGY

Our methodology centers on transforming a monolithic SAIL-RISCV emulator into a modular design that enhances compositionality and facilitates both static and dynamic module bindings

Approach: *Decompose the original monolithic emulator into a base module and individual extension modules. This separation allows each extension to expose a well-defined API while relying on a shared architectural state provided by the base emulator.*

Modular variants on top of (0) Baseline LTO: (1) Statically linked static binding, (2) Statically linked (semi-) Dynamic binding, and (3) Runtime-linked Dynamic binding (see IMPLEMENTATION)

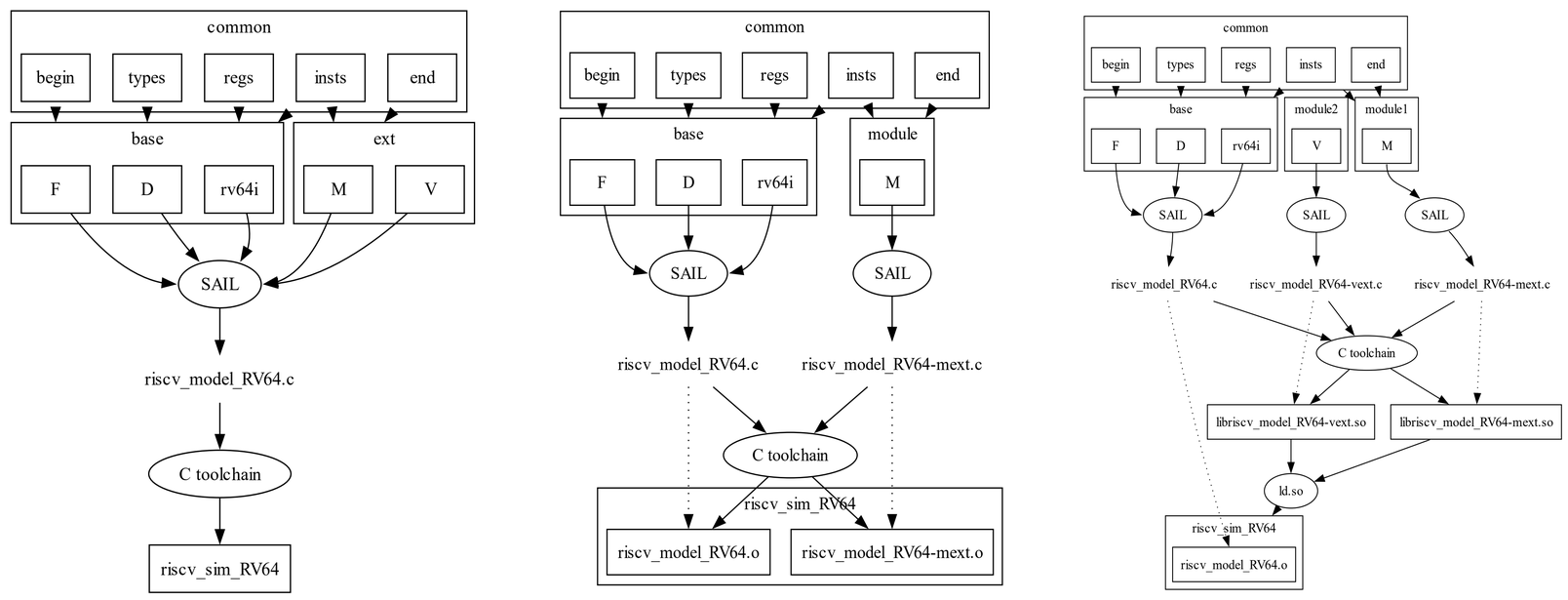
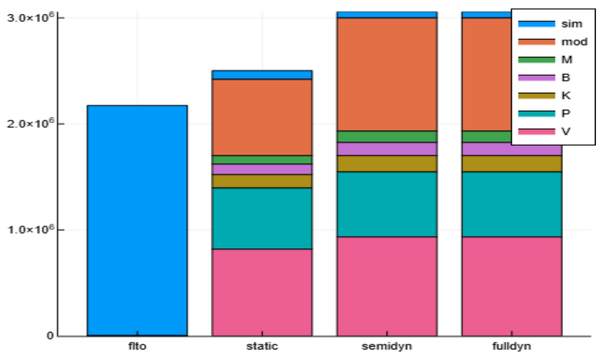


Fig 1: Proposed changes to the baseline SAIL flow (0): Static **M** (1), Semi-Dynamic **M** & **V** (2)

EXPERIMENTAL RESULTS

We evaluated the performance of the pluggable emulator using static, and dynamic bindings

- Our performance evaluation on dedicated Xeon systems demonstrates that both static and dynamic binding techniques preserve the functional behavior of the original monolithic SAIL-RISCV emulator, and that the performance is in many cases improved
- The modular approach incurs no excessive binary size overheads. While the baseline LTO is still the most code size-efficient solution the segregation of base model and extension models into modules with static- or dynamic binding does not compromise code size.



- Furthermore, by enabling isolated testing of both the base core and individual extension modules, our approach improves testing and debugging processes, ensuring that each component can be verified independently before integration.

Fig 2: Impact of changes on binaries

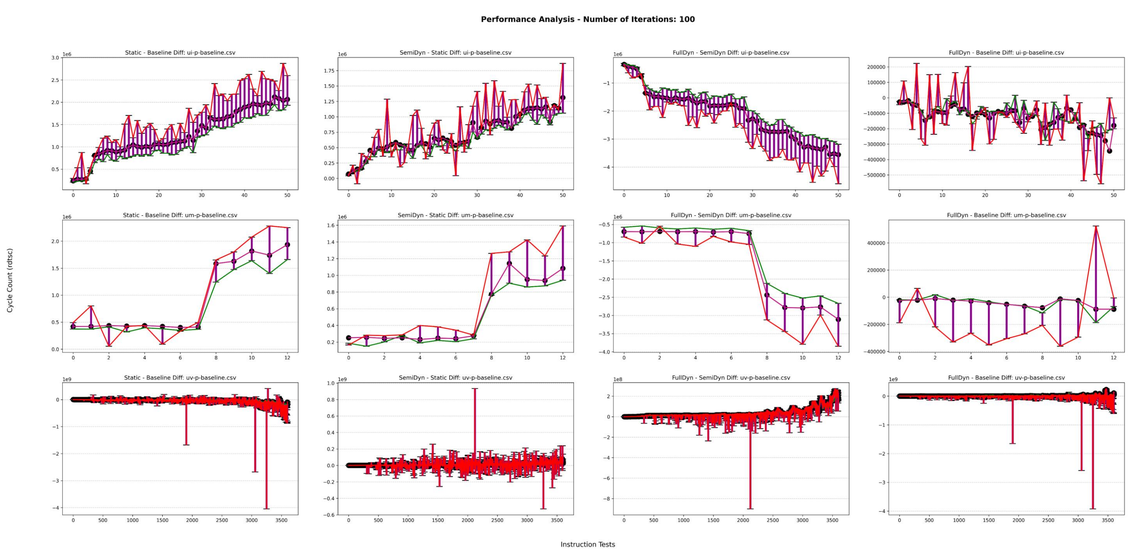


Fig 3: Performance of adapted SAIL flow

IMPLEMENTATION

- Each module in our system provides a clear API for querying, initializing, decoding, executing, and pretty-printing instructions. These module emulators support only the instructions from their defined ISA extensions and rely on the base emulator module for e.g., RF, system context and LSU
- Integration is done through simple patch that adds required calls to respective functions in **zstep()** and **model_init()** of the base emulator.
- Statically linked static and dynamic bindings are inserted directly into the code for the Static case (1) and Semi-Dynamic case (2)
- For Fully-Dynamic (case 3) extension loading is parameterized from cmdline
- Only used modules are loaded & invoked for execution of the RISC-V ISS
- Insertion of extension API calls is performed via an simulator indirection
- Results are positive:
  1. Less cache pressure
  2. Less branching

➔

Faster overall execution☺

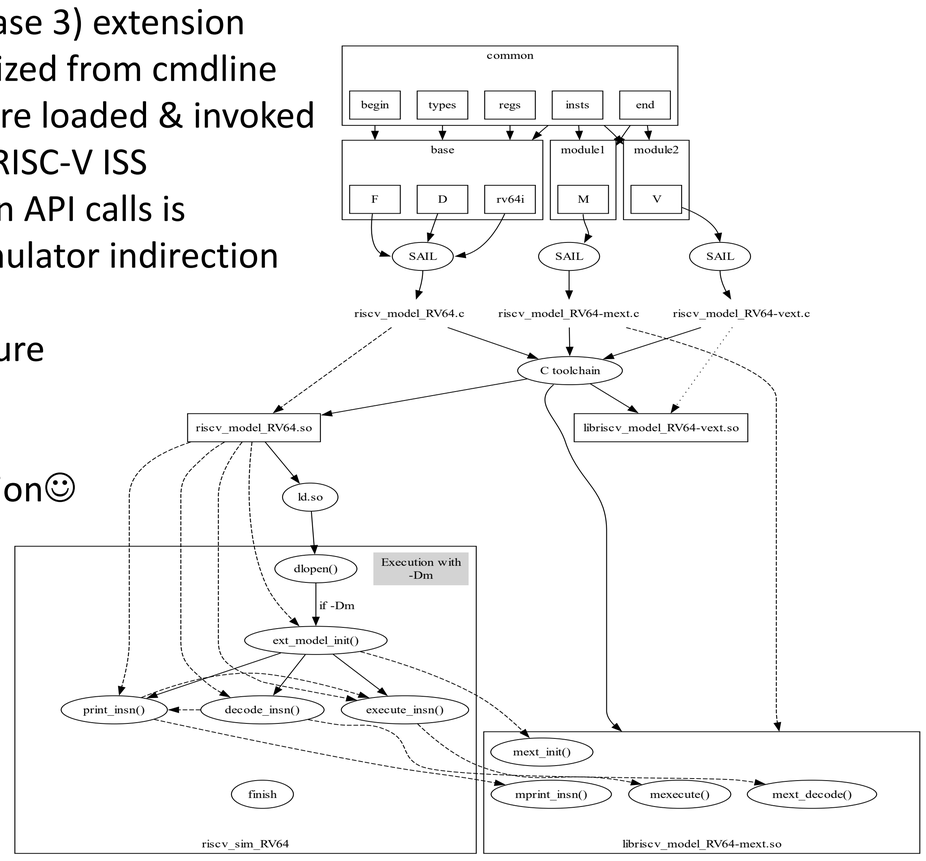


Fig 4: Proposed SAIL flow changes for the Fully-Dynamic **M** (3)

CONCLUSION AND FUTURE WORK

This experiment validates that:

1. injecting compositionality into the SAIL-RISCV flow is not only feasible but also advantageous for the evolving RISC-V ecosystem
2. By decoupling extensions from the base emulator, vendors can independently develop and exchange dynamic libraries without exposing proprietary details
3. thereby fostering a more open, collaborative and secure environment.

This modularization strategy paves the way for future enhancements, such as

- on-demand loading and runtime certification of extension modules
- supporting further extensions as modules, including floating-point **F**, **D** as well as **VectorCrypto** or the upcoming IME/AME/AI enhancements

We believe that this work represents a significant step toward creating a scalable, adaptable, and compositional framework that meets the growing complexity of modern RISC-V designs.

REFERENCES

[1] Andrew Waterman, "The RISC-V instruction set". https://doi.org/10.1109/HOTCHIPS.2013.7478332
[2] Open Vector Iface. https://shorturl.at/ulFa9
[3] CORE-V-XIF. https://shorturl.at/egnAV
[4] CX Proposal. https://shorturl.at/ufIGF
[5] SAIL. https://github.com/rems-project/sail
[6] Model. https://github.com/riscv/sail-riscv
[7] Spike RISC-V ISS. https://shorturl.at/OOBlf
[8] CBI from Codasip. https://shorturl.at/X61Kj
[9] Evaluating a RISC-V processor running Benchmarks using the QEMU VP: https://shorturl.at/aF11p