
1 Abstract

Technology trends prompting architects to consider greater heterogeneity and hardware specialization have exposed an increasing need for vertically integrated research methodologies. These methodologies may benefit from accelerated performance, area, and energy metrics of future architectures. However, constructing such a methodology with existing tools is a significant challenge due to the unique languages, design patterns, and tools used in functional (FL), cycle-level (CL), and register-transfer level (RTL) modeling. We introduce a new framework called PyMTL that aims to close this computer architecture research methodology gap by providing a unified design environment.

2 Motivation

Energy and power constraints in modern computing systems have driven architects to consider optimizations which reach across the entire computing stack. This has prompted a need for vertically integrated research approaches that use multiple modeling methodologies to effectively explore novel architectures at various levels of abstraction. For example, the incremental design of a specialized accelerator from algorithm to implementation may leverage the following modeling methodologies:

- Functional (FL) modeling to perform algorithmic exploration.
- Cycle-level (CL) modeling for rapid architectural exploration.
- Register-transfer level (RTL) modeling for extraction of critical area, energy, and timing.

We call such a vertically integrated approach to design space exploration a modeling towards layout methodology. Unfortunately, current tools for FL, CL, and RTL modeling typically use different programming languages, design patterns, and software tools that are difficult for designers to quickly transition between abstraction levels. The computer architecture research methodology gap makes it a challenge for architects to rapidly move across these levels and create a productive, vertically integrated design flow.

3 PyMTL

PyMTL is a Python-based proof-of-concept framework designed to be a unified environment for constructing FL, CL, and RTL models, enabling a productive vertically integrated computer architecture exploration. PyMTL sets the foundation for code generation and simulation. The PyMTL community has contributed versions available on GitHub.

The PyMTL framework consists of the following core components:

- PyMTL for constructing custom high-level behavioral models.
- A Python API for analyzing models described in the PyMTL DSEL.

A PyMTL tool for translating PyMTL RTL models into Verilog.

A Python testing framework for model verification.

4 The PyMTL DSEL

The PyMTL domain-specific embedded language (DSEL) provides several constructs to enable concurrent-structural modeling and simulation. The DSEL supports several domain-specific features, including:

- Iterative, Declarative, and Functional for specifying parameterizable interfaces.
- Views and a connect for programmatic structural computation of models.
- Several decorators for specifying concurrent block execution semantics (i.e., tick(FL), tick(CL), tick(RTL), and combinational).

The base class model provides a simple, clean boundary between hardware modeling logic and simulation implementation logic testing user focus on hardware design rather than simulator software engineering.

5 PyMTL Testing Models

Testing of PyMTL tools and libraries is performed in Python using the open-source pytest library. Below is an example test which is parameterized to verify behavior for different bitwidths and port numbers.

6 Modeling Towards Layout in PyMTL

PyMTL was developed for the incremental refinement of a component from a high-level model to bit-precise RTL implementation. The use of post- and pre-translation intermediate models designed to construct a test harness once and reuse across abstraction levels to verify FL, CL, and RTL PyMTL models, as well as to validate Verilog models using the TranslationTool.

7 SimJIT

SimJIT significantly improves CL/RTL simulation speed over CPython, but even greater benefit when combined with PyMTL. SimJIT has an advantage when compared to CPython, richer features, and better performance. SimJIT provides a simple framework to integrate existing tools into the PyMTL ecosystem.

8 Acknowledgments

This work was supported in part by NSF CAREER Award 1149144, DARPA Young Faculty Award, and donations from Intel Corporation and Synopsys. The authors acknowledge contributions from Grace Neubert, Scott Blankenhorst for their valuable PyMTL models and thoughtful feedback. Edgar Muñoz for his help writing PyMTL models, and Sean Clark and Matheus Ogleari for their help developing the C++ and Verilog model libraries.