Investigating the benefits of Composable Emulation: Styx Emulator as a Case Study

Jordan Moore[1], Emilie Taylor[2], Ramesh Balaji[3],
Matthew Revelle[4], Kevin Orr[2]

Styx Emulator[1], Kudu Dynamics,[2] NSF REU student at Montana State University,[3] Montana State University[4]

Styx Emulator

The **Styx Emulator** (**Styx**) is a new modular emulation framework purpose built for debugging target programs in non-standard architectures.

Included Features

- Built-in fuzzer, gdbserver, and memory error detection
- Interrupt data tracing
- High fidelity (and cross emulator) tracebus
- Ghidra interoperability plugins
- Swap out any component for your own (including execution backends)
- Concurrent emulation, fuzzing, debugging, and tracing of different architectures

Styx can be used to build tailored emulation through composition, without burdening users with emulation boilerplate. Styx can utilize existing Ghidra [2] **processor models** which describe the architecture and how it should be translated to the **P-Code** intermediate representation [3].

Emulation Framework Features

Comparison of emulation frameworks and their capabilities

Feature	**Styx**	**QEMU/Unicorn**	**Icicle-Emu**
License	BSD-2	GPL-2	MIT or Apache2
Language	Rust	C	Rust
Intended Usecase	Embedded + DSP Bug-finding	General OS Emulation	Linux usermode Bug-finding
Multi-processor	Native	No	Maybe
Fuzzing	Customizable	Old fork	Semi-customizable
Instruction Execution	Configurable	QEMU Tcg	SLEIGH + Cranelift JIT
User Configurable	All core interfaces	No	No
New Target Support	Checklist process	Good Luck	Possible

Case Study 1: Modular Component Development

New Architecture Support

Using Styx, an emulation developer was able to add **SuperH** support to the **P-Code backend** and Styx proper in a matter of hours. This resulted in the team finding a previously undiscovered bug in the target system.

External Device Modeling

A common pattern is to use Styx to create a representative **digital twin** of the system, focusing on the dataflow between the application processor and the analog data coprocessor. These devices commonly have components that perform bespoke tasking or data sensing. Using any programming language with a `gRPC` library, a developer can write a model that simulates the behavior of external devices affecting target code or replace in-tree models that require user tailoring.

External Tool Interoperability

Using the previous `gRPC` technique the Styx team was able to model a high fidelity **IMU** and provide visualizations and live data feedback to test and evaluate analog data flowing through the emulated IMU microcontroller.

Motivations for Composable Emulation

Several insights were discovered from past experiences building and using digital twins and emulators:

Emulation is not the end goal

- Testing, debugging, etc. is the end goal
- Emulation is an enabler of that goal

Making digital twin emulators takes time

- If it takes too long to produce a digital twin, a simpler one-off tool that implements a subset of the desired functionality will replace it
- The less effective one-off replacement will accumulate tech debt

You don't always need full hardware support

- No need to support hardware interrupts if you only need user-mode emulation
- No need to support syscalls if you only need to test a single simple function

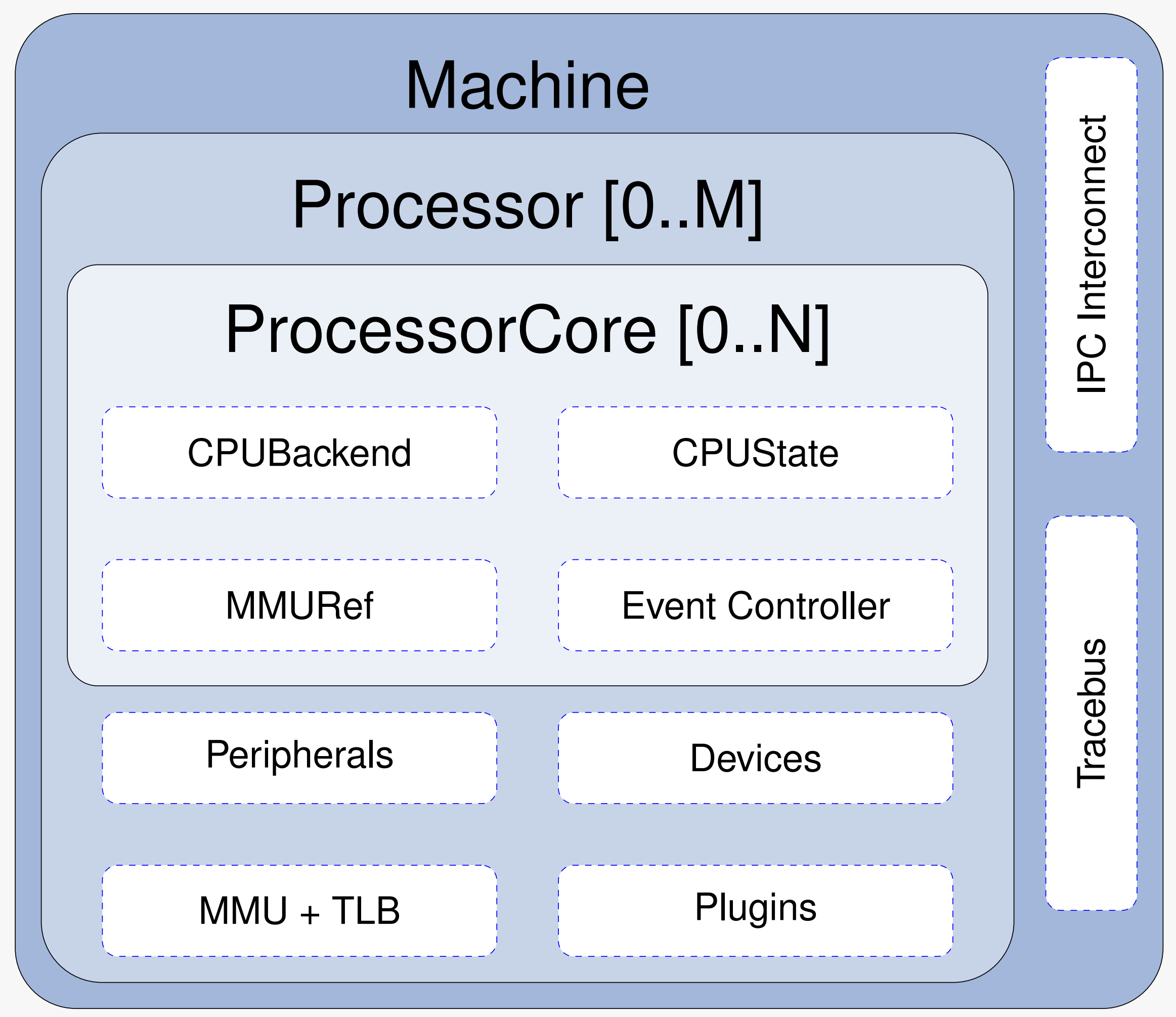
Re-tooling or building one-off tools is not scalable

- Leads to developer burnout
- Accumulate tech debt as focus changes
- Each new tool adds a new single-point-of-failure to the person who made it
- Each use case is different, and may focus on different specifics

Design and Architecture of a Composable Emulator

These interfaces make up our implementation of **composable emulation**. All of these interfaces can be tailored by the user at will.

- Utilize any pre-made code via the interfaces
- Extensions communicate and interact with machines
- Compose or customize pieces as needed



Acknowledgments

Styx Emulator is Copyright of Kudu Dynamics LLC, a Leidos Company. This research was partially supported by the National Science Foundation under award number 2243010 and DARPA under award HR001125C0014. The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

Case Study 2: Rapid Instruction Set Architecture (ISA) Support

Tool-assisted ISA support

From past experience adding new ISA support in Styx, we have found that Styx saves significant time relative to other frameworks by providing the following tools:

- Target support checklist to guide the overall process
- Abstracted memory MMU/TLB model that brokers access to code and data
- Built-in P-Code execution backend built to be extended by any Ghidra architecture definition
- Deterministic abstracted code execution loop
- Ability to accurately execute P-Code instructions
- Common functionality for manipulating and using registers
- Hooks to support P-Code's `CALLOTHER` instruction that requires special functionality
- Hooks for special register reads and writes

Hexagon Specifics

The Qualcomm Hexagon ISA is a **VLIW** (very long instruction word) architecture [1]. Instructions are contained within "packets" and instructions in the same packet are executed in **parallel**. Hexagon is the first VLIW architecture supported by Styx.

While the modular design of Styx allowed Hexagon support to succeed in a matter of weeks, fitting in packet-based semantics to a serial execution pipeline came at great cost to performance and maintenance overhead that is not sustainable. The final revision of Hexagon support culminated in a new **Request For Comments** (**RFC**) planning for a packet-based instruction executor abstraction under the current `CpuBackend` currently being implemented.

Conclusion and Future Work

Styx is a first-of-its-kind emulation framework, abstracting over multiple instruction emulation backends, and providing builtin tracebus, plugins, and programmable I/O. allowing users to get a level of introspection and control over the target environment not available in other emulation frameworks.

In the future, Styx will focus on improving the multi-emulator capabilities, and creating a robust emulation development and debugging experience centered around the Styx Emulator. Additionally Styx will investigate modular abstractions centered around instruction decoding to enable integration of poorly supported architectures in a manner requiring less work from users.

References

[1] Lucian Codrescu, Willie Anderson, Suresh Venkumanhanti, Mao Zeng, Erich Plondke, Chris Koob, Ajay Ingle, Charles Tabony, and Rick Maule.
Hexagon dsp: An architecture optimized for mobile multimedia and communications.
IEEE Micro, 34(2):34–43, 2014.

[2] National Security Agency.
Ghidra software reverse engineering framework.
`https://github.com/NationalSecurityAgency/ghidra`.

[3] Nico Naus, Freek Verbeek, Dale Walker, and Binoy Ravindran.
A formal semantics for p-code.
In Akash Lal and Stefano Tonetta, editors, *Verified Software. Theories, Tools and Experiments.*, pages 111–128, Cham, 2023. Springer International Publishing.