GVSoC: A Highly Configurable, Fast and Accurate Full-Platform Simulator for RISC-V based IoT processors

Paper review

Vinicius Peixoto

MO801 - Tópicos em Arquitetura de Computadores

Sep 17th 2025

Agenda

1. Introduction
2. Target architecture
3. GVSoC architecture
4. Use cases and results
5. Comparison with other tools and conclusion
6. Discussion of the paper

Introduction

Architecture-level simulation

- Different levels of abstraction
- Tradeoff between accuracy vs. simulation speed
- **Functional simulators**
  - Only simulate behavior (no uarch nuances)
  - Typically very fast, often inaccurate
  - Example: Spike

Architecture-level simulation

- **Timing simulators**
  - Model the uarch of the target (cache hierarchy, pipelines, branch pred., ...)
  - Much slower than functional simulation
  - **Cycle-accurate**: accurate cycle-by-cycle simulation
    - Example: Verilator (System Verilog → C++ executable)
  - **Instruction-level**: instruction-by-instruction simulation
    - Faster than cycle-accurate
    - Example: Gem5's O3CPU model

Architecture-level simulation

- **Timing simulators**
  - **Event-driven**: events instead of cycles
    - event = change of state in the system occurring at a certain point in time
    - Schedule events in a queue based on their latency
    - Jump directly to time of occurrence of next event
    - Skipping idle cycles → consistent savings in simulation time
    - Examples: SystemC + Transaction Level Modeling (TLM), RISC-V-TLM

State of the art

- Established timing simulation solutions lack flexibility (e.g. testing SoCs)
  - Cycle-accurate sims: slow, adding peripherals is cumbersome
  - Timing sims: faster, but still difficult to extend
- **Author's proposal**: a highly flexible, event-driven simulator targeted at full system emulation

Target architecture

PULP

- **P**arallel **U**ltra-**L**ow **P**ower platform
  - Open-source heterogeneous computing platform
  - RISC-V MCU (PULP SoC) + parallel programmable accelerator (PULP CL) + peripherals
  - Separate clock domains for easier workload tuning

PULP SoC

- FC (Fabric Controller): RISC-V processor
  - Manages the peripheral subsystem
  - Offloads compute-intensive tasks to the accelerator
  - Equipped with 256KiB - 2MiB of SRAM
    - Stores the code and application data
    - Paper calls it L2 (?)

PULP SoC

- FC (Fabric Controller): RISC-V processor
  - Comprehensive set of peripherals (JTAG, SPI, I2C, I2S, GPIOs, ...)
    - I/O DMA (called uDMA) for L2 memory ↔ peripheral data transfer
  - HyperBUS DDR interface
    - 8-bit high-speed bus for memory expansion (external DRAM, flash, ...)

PULP CL (compute cluster)

- Many RISC-V cores sharing multiple banks of **TCDM** (**T**ightly **C**oupled **D**ata **M**emory)
- TCDM banks can be accessed in a single clock cycle due to an intricate low-latency logarithmic interconnect
- 2-level instruction cache: private L1 cache (512B - 1KiB) and L1.5 cache (4-8KiB) that refills from L2 memory
- L1 ↔ L2 data transfers through multi-channel DMA

PULP CL (compute cluster)

- Hardware sync unit controls common sync operations such as barriers, thread dispatching, etc.
- Support for energy-efficient DSP extensions (Xpulp)
- Standard interface (Hardware Processing Engines - HWPE) for connecting custom HW accelerators to the PULP CL domain + L1 memory

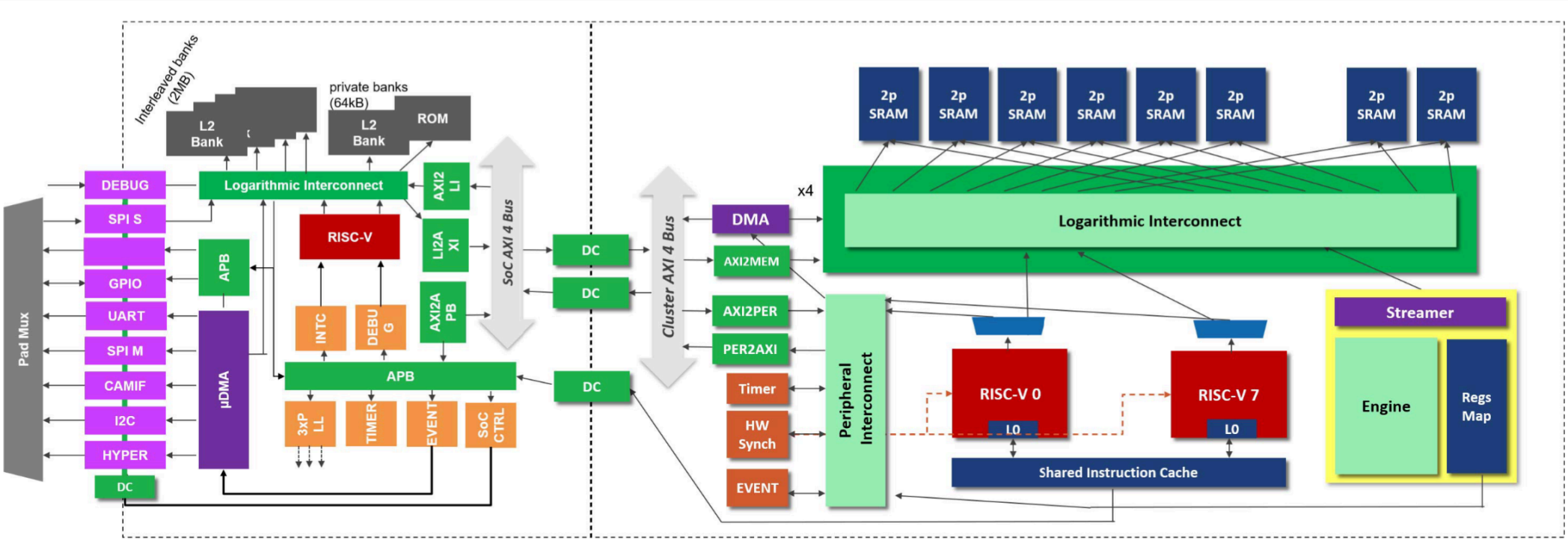


Figure 1: The PULP SoC architecture.

GVSoC architecture

Overview

- Event-driven simulator targeting the PULP architecture
- Models the uarch + common building blocks (cores, memory, peripherals, interconnects, ...)
- Designed to support easy testing of arch + uarch + I/O functionality side-by-side
- Offers debug tools and HW counters for early-stage perf evaluation

Structure

- Comprised of three main components:
  - C++ models: simulate the behavior components (cores, memories, DMA, peripherals, ...)
  - JSON configs: specify architecture params (core counts, interconnect bw/latency, ...)
  - Python generators: orchestrate the instantiation of components
  - Compile C++ models first → quick prototyping of different system configurations without the need to rebuild the models

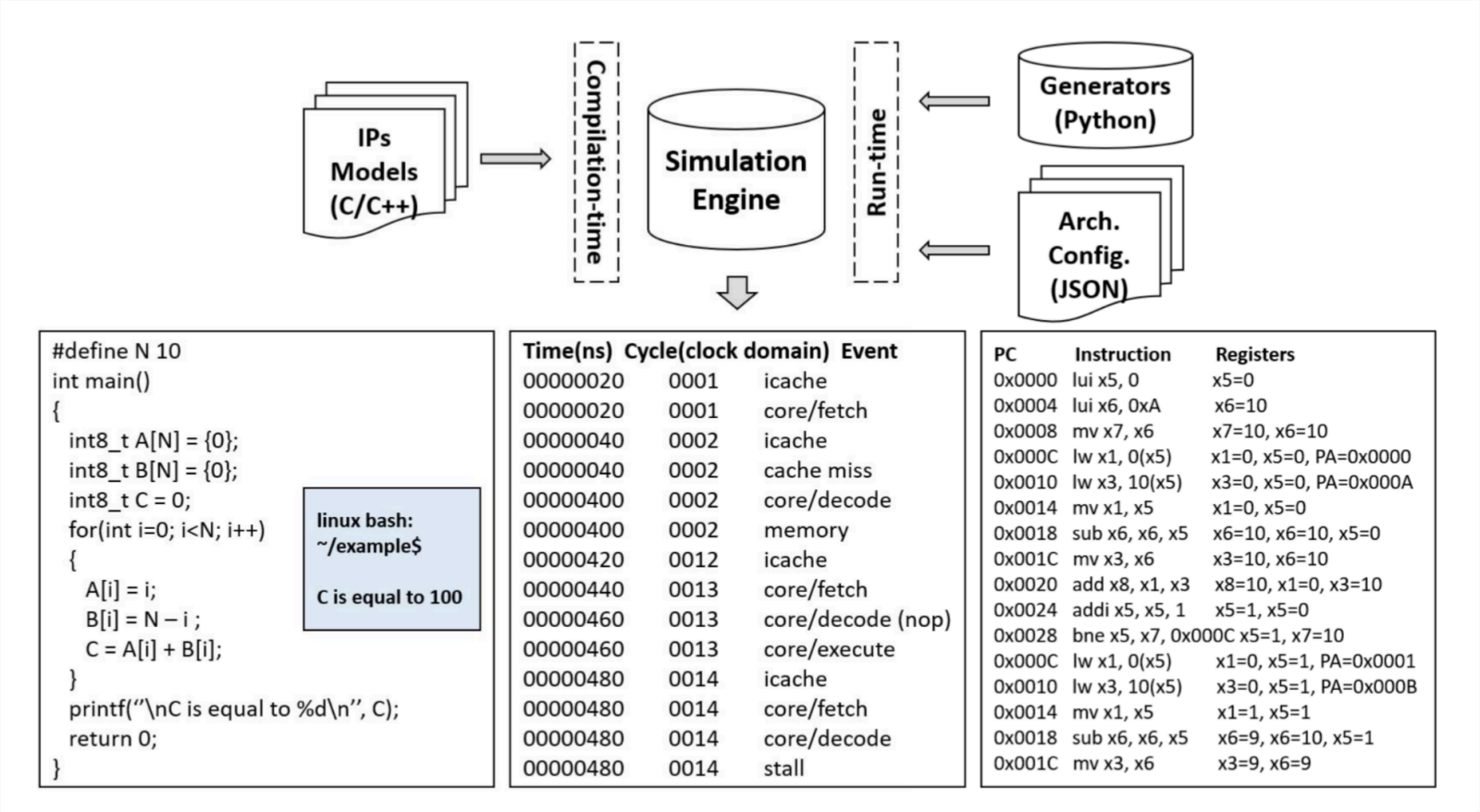


Figure 2: Overview of the main components of GVSoC.

Structure

- GVSoC components interact through **message-passing**
- Components receive **requests** and can choose to handle them by themselves or forward them to another component
- Tries to simulate latency accurately (e.g. when requests go through multiple components)

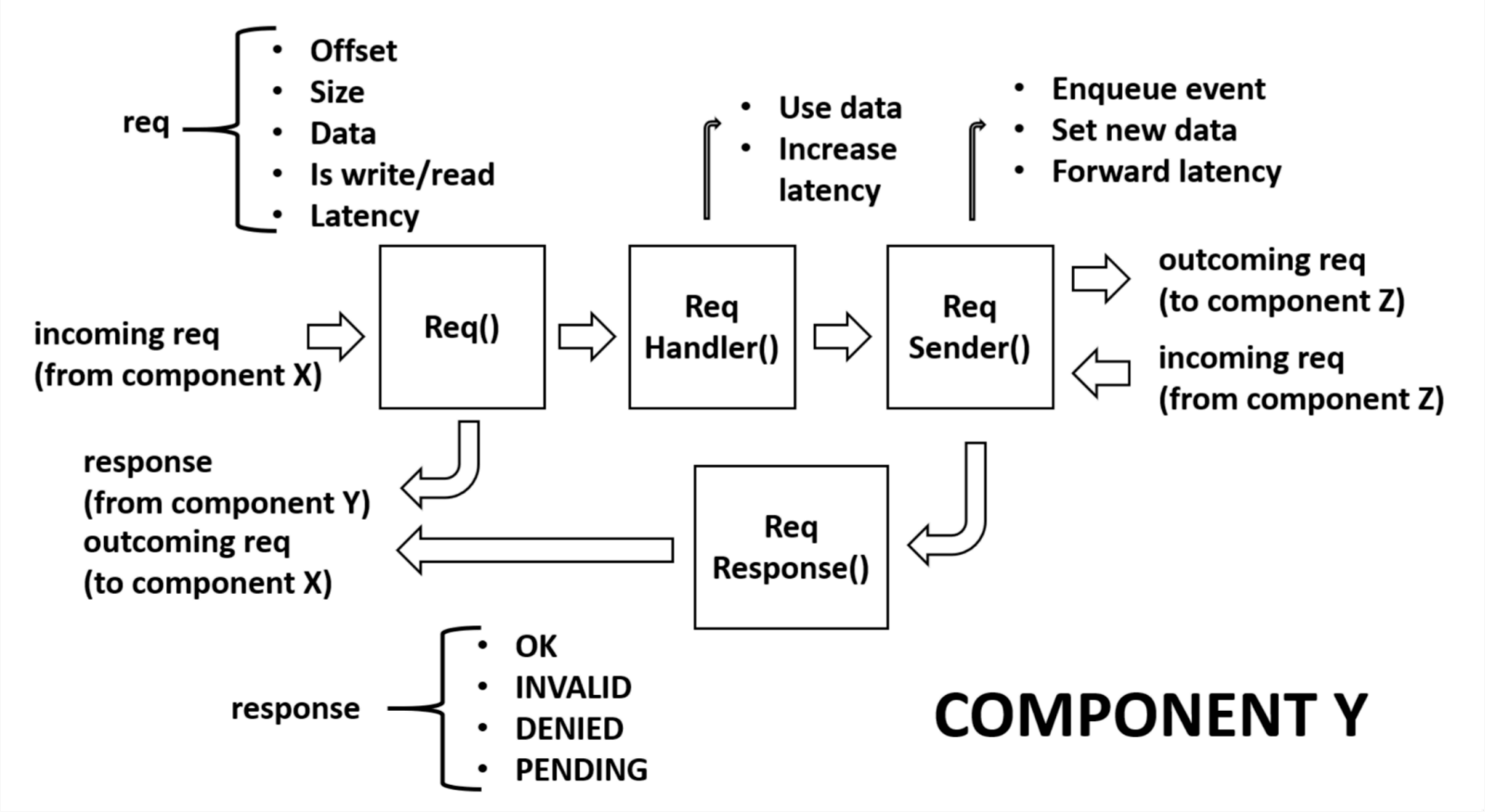


Figure 3: Example of a GVSoC request lifecycle.

Time modeling

- A **global time engine** manages the overall time (at picosecond scale)
- **Clock engines** model individual clock sources
  - Forward monotone counters associated with a circular event queue
  - Each engine defines a **window** T_w; every event that will happen within T_w cycles in included in the circular queue
  - Simultaneous events are executed sequentially within the same cycle

Time modeling

- **Clock engines** model individual clock sources
  - Events outside of the T_w window are sent to a separate ordered queue (called the **delayed event queue**)
  - Whenever a lap around the circular event queue is completed, events are read from the delayed event queue

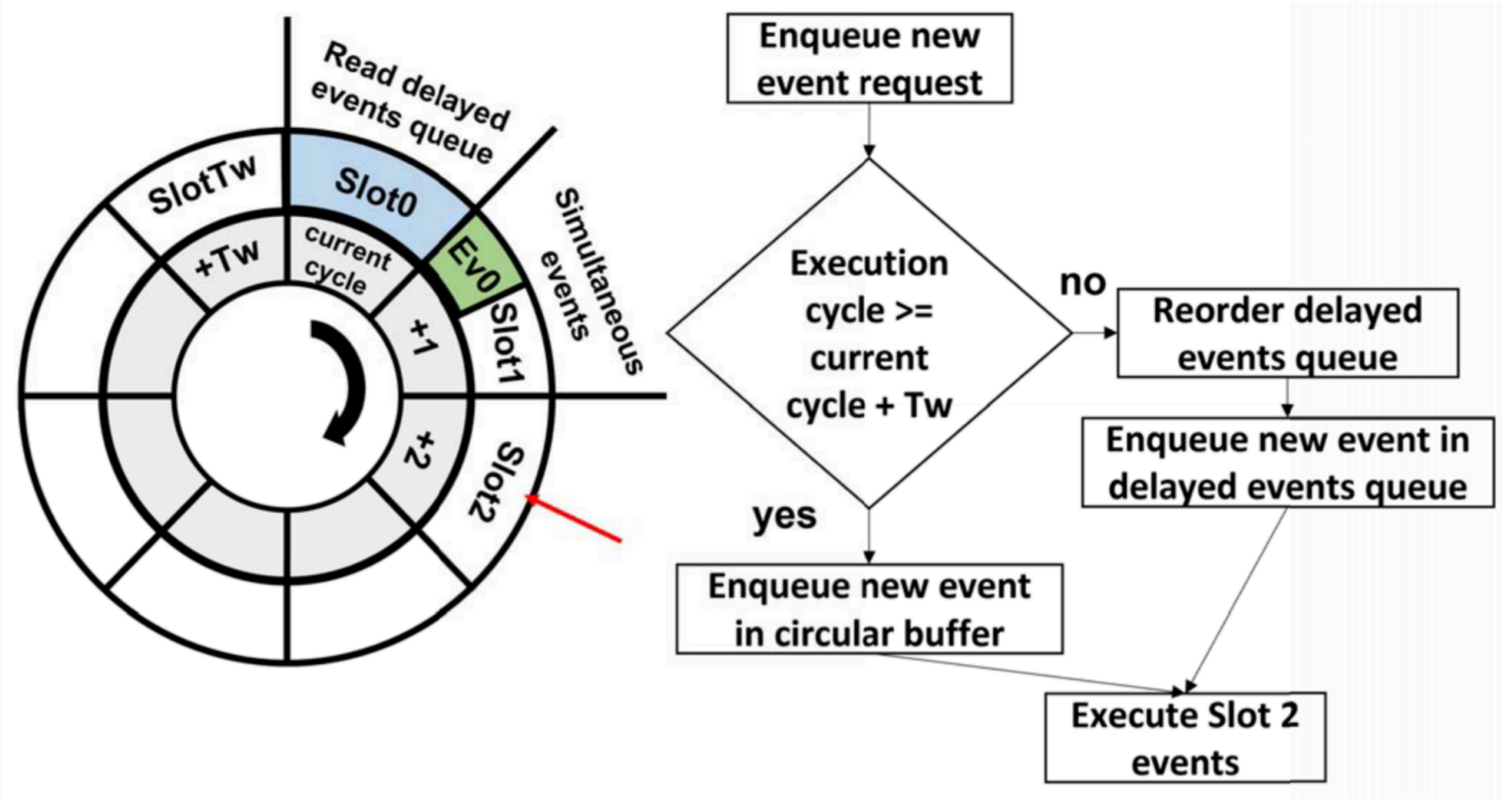


Figure 4: Overview of the GVSoC event queue.

Time modeling

- All componentes are related to a **clock domain**, i.e. a tree of clock sources (with associated clock engines)
- Mechanisms for synchronizing requests across clock domains (**stubs**)

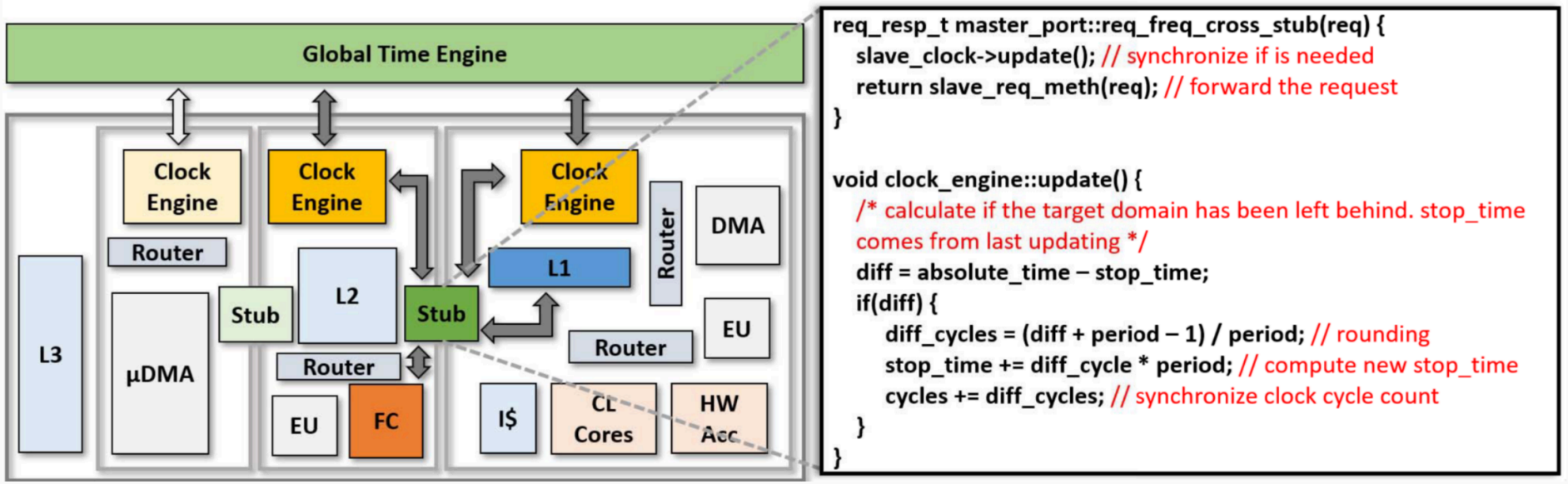


Figure 5: Overview of the GVSoC time engine.

Hardware perf counters

- Models performance metrics for real hardware
- Full tracing capabilities for events in GVSoC

Hardware Performance Counters
Total number of cycles
Total number of cycles spent in active mode
Number of executed instructions
Number of load data hazards
Number of jump register data hazards
Cycles waiting for instruction fetches
Number of executed data memory loads
Number of executed data memory stores
Number of executed unconditional jumps
Number of executed total branches
Number of executed taken branches
Number of executed compressed instructions
Number of external loads
Number of external stores
Cycles for external loads
Cycles for external stores
Number of TCDM contentions

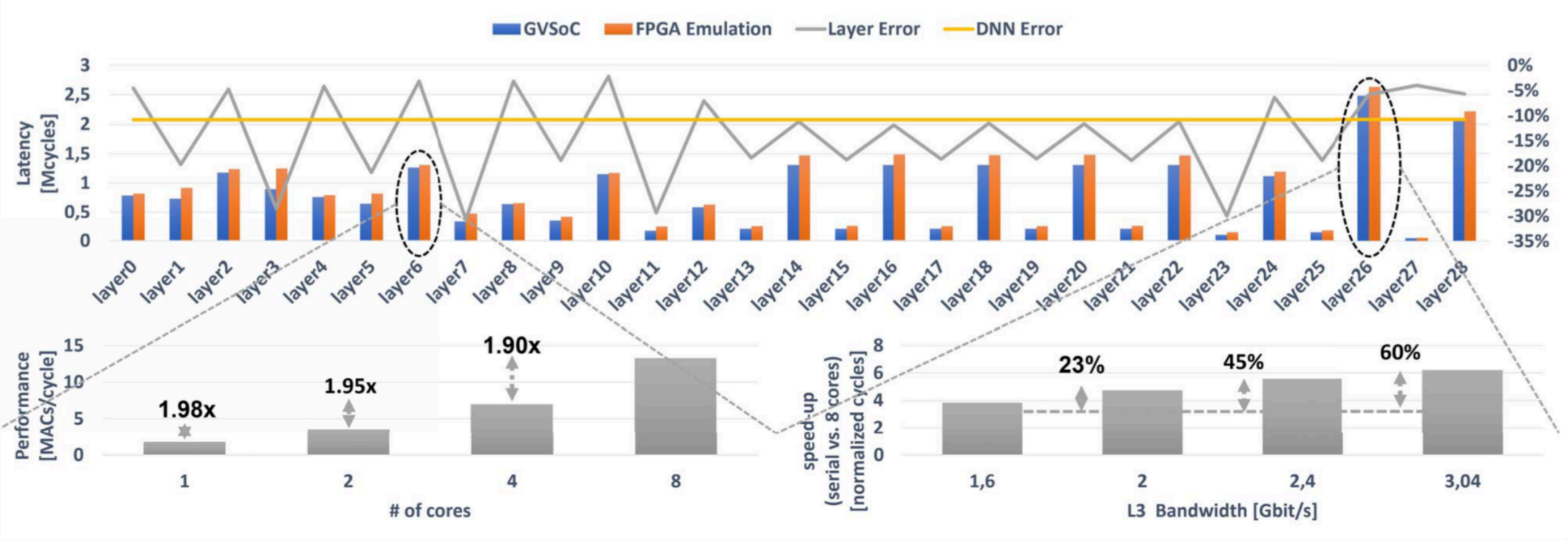
Use cases and results

The author goes over 3 use case studies:

- Execution of a full MobileNetV1 model
- Running commonly-used DSP kernels using the PULP CL
- Integrating a custom convolution hw accelerator in PULP CL

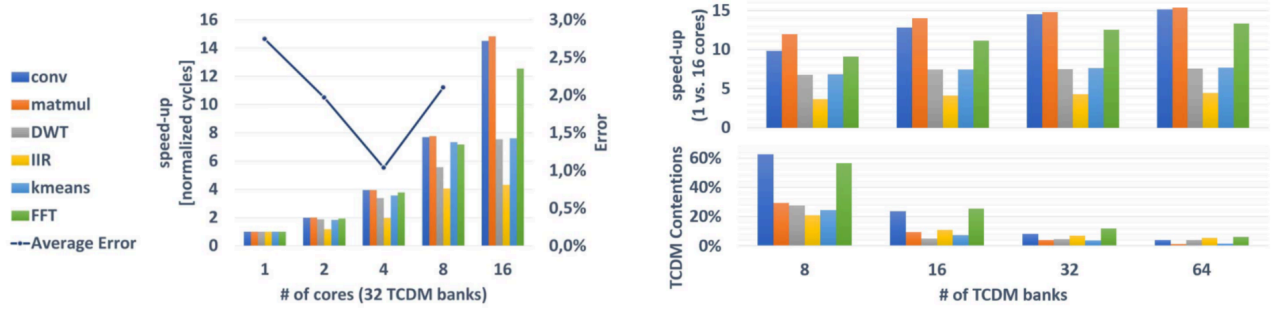
MobileNetV1

- Emulated both on GVSoC and on a Xilinx ZCU102 FPGA
- Uses PULP CL cores in a SIMD fashion
- Compares simulation error between FPGA and GVSoC


GVSoC
FPGA Emulation
Layer Error
DNN Error
Latency [Mcycles]
3
2,5
2
1,5
1
0,5
0
0%
-5%
-10%
-15%
-20%
-25%
-30%
-35%
layer0 layer1 layer2 layer3 layer4 layer5 layer6 layer7 layer8 layer9 layer10 layer11 layer12 layer13 layer14 layer15 layer16 layer17 layer18 layer19 layer20 layer21 layer22 layer23 layer24 layer25 layer26 layer27 layer28
Performance [MACs/cycle]
15
10
5
0
1.98x
1.95x
1.90x
1
2
4
8
# of cores
speed-up (serial vs. 8 cores) [normalized cycles]
8
6
4
2
0
23%
45%
60%
1,6
2
2,4
3,04
L3 Bandwidth [Gbit/s]

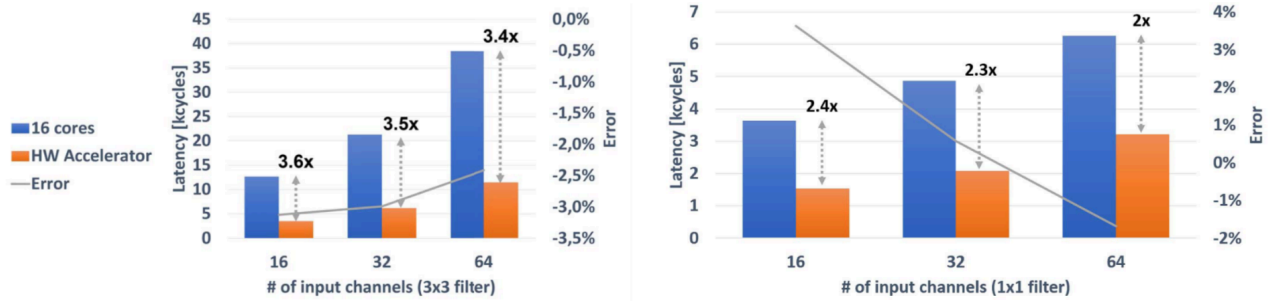
DSP kernels + custom convolution accelerator

- Emulated only on GVSoC
- Explores performance scalability in response to number of cores and TCDM banks



(a) (b)

Fig. 6: Common DSP kernel execution to show (a) the scalability with number of cores (b) how much TCDM banks impact the performance in relation with the TCDM contentions.



(a) (b)

Fig. 7: Comparison on convolution executions between 16 PULP CL cores and hardware accelerator varying CH_{in} in: (a) $K_{size} = 3$ (b) $K_{size} = 1$.

Comparison with other tools and conclusion

RISC-V Simulator	Target Field	Category	Open-Sourced	Core Extensions	Multicore	I/O Peripherals	Accelerators	Speed [MIPS]	Timing Accuracy
Spike [17]	CPU	functional	yes	yes	yes	no	no	170	n.d
RISCV-OVPSim [26]	Full-platform	functional	yes	yes	yes	yes	no	1000	n.d.
QuestaSim [27]	all	cycle-accurate	no	yes	yes	yes	yes	0.01	100%
Verilator [27]	all	cycle-accurate	yes	yes	yes	yes	yes	0.1	100%
gem5 (MinorCPU) [28]	System	instruction-level	yes	no	yes	no	no	0.2	75%
RISCV-TLM [17]	Full-platform	event-driven	yes	no	no	yes	no	8	unknown
RISCV-VP [29]	Full-platform	event-driven	yes	no	no	yes	no	27	95%
GVSoC	**Full-platform**	**event-driven**	**yes**	**yes**	**yes**	**yes**	**yes**	**25**	**90%**

Conclusion

- GVSoC offers comparable performance to other event-driven timing simulators
- Lots of flexibility for prototyping and doing early validation on SoCs
- Shows potential for simulating IoT/low-power devices using the PULP architecture

Discussion of the paper

Discussion

- The good:
  - Very well-written, lots of content, informative discussion of the internals of GVSoC's architecture
  - The GVSoC framework itself is open source and available on GitHub
  - Provides a useful tool for other researchers to prototype their platforms on
  - Presents a good performance comparison with other available simulation tools

Discussion

- The bad:
  - Very rushed discussion of the use cases and testing methodology
    - e.g. how exactly are they leveraging the PULP CL for accelerating MobileNetV1?
  - Would like to see more details about the logarithmic interconnect bus
  - Only applicable to the simulation of resource-constrained systems (and especifically the ones using the PULP architecture)

Discussion

- The ugly
  - GVSoC's documentation is lacking
  - No source code for reproducing the tests in the paper
  - Most examples in GVSoC's documentation use the GAP9 core, which is **closed-source** and thus most examples in the docs are rendered useless
    - Some important features (such as GDB debugging) only work on this proprietary core