

System Simulation with PULP Virtual Platform and SystemC

Éder F. Zulian
TU Kaiserslautern
Germany
zulian@eit.uni-kl.de

Germain Haugou
ETH Zürich
Switzerland
haugoug@iis.ee.ethz.ch

Christian Weis
TU Kaiserslautern
Germany
weis@eit.uni-kl.de

Matthias Jung
Fraunhofer IESE
Kaiserslautern, Germany
matthias.jung@iese.fraunhofer.de

Norbert Wehn
TU Kaiserslautern
Germany
wehn@eit.uni-kl.de

ABSTRACT

Driven by performance, power and energy requirements compute platforms evolved from single-core homogeneous into highly parallel heterogeneous architectures. These architectures use different CPUs, GPUs, accelerators, interconnects, memories, etc., and target diverse applications creating a vast design space. Moreover, ever-growing security concerns leverage the adoption of open source silicon designs and tools, specially those with collaborative research and engineering efforts from industry and academia to develop and maintain for the long term. In this context, RISC-V based solutions are an outstanding choice and virtual platforms are essential for a fast design cycle. However, simulation frameworks often do not provide off-the-shelf interoperability hindering the reusability of existing models.

In this paper, we present for the first time a coupling of the PULP virtual platform, which provides manycore RISC-V accelerators in a feature-rich simulation environment, with SystemC/TLM-2.0, a de facto standard in industry that is also largely used in the academia. Furthermore, we evaluate the coupling of simulation engines and demonstrate its usefulness in a case study.

CCS CONCEPTS

• **Computing methodologies** → **Discrete-event simulation; Simulation environments; • Computer systems organization** → **Heterogeneous (hybrid) systems.**

KEYWORDS

RISC-V, PULP, Virtual Prototyping, Heterogeneous Systems, System-level Modeling, SystemC/TLM

ACM Reference Format:

Éder F. Zulian, Germain Haugou, Christian Weis, Matthias Jung, and Norbert Wehn. 2020. System Simulation with PULP Virtual Platform and SystemC. In *Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO '20)*, January 21, 2020, Bologna, Italy. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/3375246.3375256>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

RAPIDO '20, January 21, 2020, Bologna, Italy

© 2020 Association for Computing Machinery.

ACM ISBN 978-1-4503-7777-5/20/01...\$15.00

<https://doi.org/10.1145/3375246.3375256>

'20), January 21, 2020, Bologna, Italy. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/3375246.3375256>

1 INTRODUCTION

Recently, Google and other founding partners from industry and academia announced OpenTitan [10] the first open source silicon root of trust (RoT) project. According to its announcement, OpenTitan will deliver a high-quality RoT design and integration guidelines for use in data center servers, storage, peripherals, and more. OpenTitan is based on RISC-V, hence it provides us with a prominent example for the huge impact of the RISC-V ecosystem on industry. Moreover, GAP8 [7] is another example from industry that is a derivative from PULP [26] that originated in the academia. The System-on-Chip (SoC) designed by GreenWaves targets edge computing and Internet of Things (IoT) systems consisting of autonomous energy-constrained devices. The SoC relies on the PULP RISC-V architecture, it has nine RISC-V cores, I/O peripherals and a hardware convolution computation engine specialized for use with convolutional neural networks (CNNs) related applications.

Today's huge diversity of applications, system heterogeneity, performance and energy-efficiency requirements, along with a vast number design choices, expose system-level architects to an ever-growing complexity when developing compute platforms. In this context, *Virtual Platforms* (VPs) are not only a valuable resource, but also mandatory to: (1) Facilitate rapid research on all software and hardware layers. (2) Manage the inherent complexity of contemporary system's *Design Space Exploration* (DSE). (3) Accelerate the design cycle, anticipate analysis, optimization, verification. Moreover, SystemC and TLM are largely used in academic research and it is a de facto industry standard, which is successfully used by leading companies to advance their products by means of VPs.

The purpose of this work, therefore, is to present the coupling of GVSoC, the simulation engine of the PULP VP, with SystemC/TLM. For the first time, researchers and industry can profit from models provided by the PULP VP, such as manycore RISC-V accelerators and several peripherals, in a SystemC simulation environment. The coupling allows PULP VP components to be combined with SystemC compatible models to form heterogeneous simulation frameworks. Moreover, the source code developed for the coupling along with artifacts, such as SystemC modules and scripts for automated setup of prerequisites, are open source and committed to official repositories of the PULP project. Furthermore, we believe that the

interoperability of simulation engines not only increases their application scope, but also facilitates further extensions while promoting their use and long-term maintenance.

The rest of this paper is structured as follows: Section 2 discusses related work. An overview of the PULP VP focused in its simulation engine *GVSoC* along with SystemC/TLM-2.0 is provided in Section 3. The coupling is explained in Section 4. Further, in Section 5 we evaluate essential aspects of our work while demonstrating its usefulness with case studies that can be used as entry point for the creation of heterogeneous simulation frameworks. Section 6 concludes the paper.

2 RELATED WORK

A vast body of research has proposed a multitude of approaches and resourceful solutions to create VPs. As expected, the trade-off accuracy vs. simulation speed is a constant concern in virtually all works. Moreover, considerable part of these works focus not only on breakthrough innovations started from scratch, but also on the integration, extension and reuse of existing solutions. We summarize some recent achievements in the following.

The authors of [4] present an integrated virtual prototyping solution called *VPSim* that promises exceptional performance, easy of use and automation of modelling and exploration. The proposed VP uses QEMU [1] linked to SystemC TLM-2.0 modules. Performance is granted by using the TLM-2.0 specialized transport interface *DMI* at the cost of accuracy, but the solution allows different accuracy levels for specific regions of interest (ROIs). Similar approach is presented by [15], where a processor model of a recent ARMv8 ISA is generated with a QEMU-based CPU emulator framework called *Unicorn* [20]. The authors use SystemC to integrate the processor model into a VP in combination with a few third party SystemC/TLM peripherals from [27]. Temporal decoupling is used to increase simulation performance.

Ptolemy [5] and *FERAL* [17] allow multiple models of computation (MoCs) in a hierarchical heterogeneous design environment. Moreover, *FERAL* allows the integration of compiled/linkable tools for which the source code is available with closed-source or non-compiled frameworks. Different MoCs are also integrated in [21], where a new approach for virtual prototyping of analog and mixed-signal embedded (AMS) systems based on SysML is proposed. The authors extract low-level cycle accurate prototypes from the high-level abstraction models and present the integration of analog components into a virtual prototyping framework. Another characteristic of *FERAL*, also used by the *Structural Simulation Toolkit* (SST) [22], is the use an intermediate software layer to bind simulation components into an integrated environment. According to its documentation, the SST was designed to explore highly concurrent systems where the ISA, microarchitecture, and memory interact with the programming model and communications system. Recently, a parallel simulation environment based on MPI was added to the toolkit, what may counterbalance any overhead generated by the middleware layer. Parallelization is a relevant option for accelerating simulations that is also explored in [28] that uses temporal decoupling and a parallel SystemC kernel called *SCoPe* [29] to speedup simulations of a SystemC-based MPSoC platform formed by a SystemC-wrapped Instruction Set Simulator (ISS)

together with peripherals. Moreover, further speedup is achieved by skipping the simulation of processors in idle state.

An extendable translating instruction set simulator is presented in [19]. The proposed framework does binary translation and implements a plugin mechanism that allows to quickly include new functionality whenever an interrupt is received, e.g., into the translation stage, the simulation loop, during accesses to the memory. Moreover, the framework has special focus on virtual prototypes written in SystemC/TLM and its plugins include tracing tools, SystemC interfaces, peripherals and fault injection capabilities.

The authors of [25] create a heterogeneous VP which consists of a SystemC-based GPU attached to the existing open source SystemC-based framework *SoCRocket* [6]. Besides GPU, the proposed VP supports the open source LEON CPU and GRLIB components. According to the authors RTL implementations are provided what makes possible a feedback loop between simulation results into realistic hardware models. Similarly, the PULP project provides open source RTL implementations. In [16] a memory simulator called *NVMain* is extended with a reference implementation of a new SRAM cache and implementation of various latency optimizations for die-stacked cache. In this work, SystemC/TLM models from *DRAMSys* [14] build a memory subsystem used in a case study for evaluation purposes.

The authors of [12] proposed and implemented a RISC-V based virtual prototype in SystemC/TLM that provides detailed models of a RISC-V core, an interrupt controller and essential peripherals that can be used for system-level exploration. RISC5, an implementation of a RISC-V ISA in gem5, was presented in [23]. According to its authors, RISC5 is validated against performance data from the Chisel C++ emulator and FPGA soft core and is shown to have less than 10% error on several performance statistics. Moreover, [23] was further evaluated and extended by [24]. Also in gem5, the authors of [18] provided support to SystemC co-simulation. This was a key step that allowed the reuse of openly available models developed for more than a decade in joint effort of researchers and industry and simulation workloads together with SystemC-based models.

3 BACKGROUND

This section provides a brief overview of the PULP project along with the PULP VP and SystemC/TLM-2.0 with focus on characteristics that are relevant to this work.

3.1 PULP Project

PULP [26] is an open source parallel architecture. One of its configurations has a main core and a main memory called *Accelerator DRAM* connected to a cluster of RISC-V cores sharing a local memory and using a DMA engine to transfer data between the accelerator DRAM and the cluster internal memory. Tasks can be offloaded to the PULP accelerator cluster. The RISC-V core(s) within the accelerator that are assigned to the task uses a DMA engine to transfer data between memories.

3.2 PULP Virtual Platform

The PULP Virtual Platform is part of the PULP project, hence it evolves and is maintained along with the PULP ecosystem. The

simulation engine of the PULP VP is called GVSoc, which is developed in Python and C++ using a component-based approach. Each component has client and server interfaces. Similarly to gem5 [2], the instantiation and binding of all components is done in Python while the models are written in C++. Python is used to control the simulation but once it is running, the execution uses only C++. The whole platform is a set of parametric models. Each target is described by a JSON description of the architecture which is driving the instantiation, connection and configuration of each component.

The execution model is using an event-driven approach. Any activity in a model is simulated as a function call which executes until completion. To correctly model frequency scaling, each model is part of a clock domain and the model is scheduling its activity using cycles. An activity can be registered in the future by specifying a number of cycles so that the model is not affected by a frequency change. A *Top Time Engine* is responsible for scheduling the various clock domains according to their activity. Time stubs are automatically inserted when two components of two different clock domains are bound together to apply timing conversions. Performance is modeled by scheduling activities at a specific time. For example, the function modeling an instruction will enqueue another activity two cycles later to model the fact that the instruction took two cycles to execute. Accuracy depends on properly timing the core pipeline (data dependency, branch penalty and so on). The interconnects are also timed so that they can properly model bandwidth and latency, using an asynchronous protocol. A master can send multiple outstanding requests and the slave can grant them and reply to them asynchronously so that advanced behaviors can be modeled. This approach resembles the TLM-2.0 approximately-timed (AT) coding style with non-blocking transport interface.

3.3 SystemC & TLM

SystemC is a C++ class library and an industry standard defined in the IEEE Standard for Standard SystemC® Language Reference Manual [13] often used in combination with TLM. A robust and mature proof-of-concept simulator is made openly available by Accellera Systems Initiative. TLM has two main focuses: transaction-based *communication* and *interoperability* of modules. A TLM transaction comprises a collection of pin wiggles, hence it allows a substantial reduction in the number of events that have to be handled by the event-driven simulation kernel reducing the simulation time. Interoperability can be achieved for non-native SystemC/TLM models with the use of *wrappers*, *transactors*, *adapters* and *bridge components*.

4 COUPLING PULP VP AND SYSTEMC

In this section we explain the most relevant aspects of the coupling of the PULP VP and SystemC/TLM-2.0.

An overview of the coupling is shown in Figure 1. The required changes and new open source components created to this end are detailed in the following.

4.1 Preliminary Aspects

The coupling is achieved by executing the *Top Time Engine* of GVSoc (the simulation engine of the PULP VP) as a SystemC thread as shown in Listing 1, hence it is scheduled for execution by the

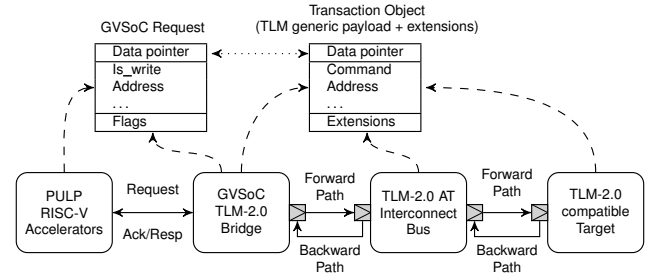


Figure 1: Transaction objects are constructed on-demand in the Bridge. The data pointer within a request is referenced in the transaction avoiding the copy/movement of data.

Listing 1: Top Time Engine within a SystemC thread

```

1 SC_MODULE(gvsc_module) {
2   SC_HAS_PROCESS(gvsc_module);
3   gvsc_module(sc_module_name n, vp::time_engine *eng) :
4     sc_module(n), engine(eng) {
5     SC_THREAD(run);
6   }
7   void run() {
8     eng->run_loop();
9   }
10  vp::time_engine *engine;
11 };

```

SystemC simulation kernel. Moreover, at the beginning of a simulation, during the transition from the Python configuration phase to the C++ execution phase an initialization function is called by the thread running the *Top Time Engine* to execute the elaboration phase in which all GVSoc and SystemC modules are instantiated. Subsequently, `sc_start()` is called.

Besides the *Top Time Engine*, other two components are used to establish communication and integration of GVSoc and SystemC components. They are: (1) The *GVSoc TLM-2.0 Bridge*, a transactor responsible for the translation of GVSoc requests into TLM transactions and vice versa. (2) The *Approximately-Timed (AT) Interconnect Bus*, a memory mapped interconnect bus where any number of TLM compatible components can be attached.

4.2 GVSoc Top Time Engine

To have a close interaction between GVSoc and SystemC schedulers, *GVSoc Top Time Engine* is embedded in a SystemC thread as shown in Listing 1. Every time this special thread is scheduled for execution by the SystemC kernel, it executes all the activities scheduled for the current simulation time, allowing modules from both sides to create new events. Once no more activity has to be executed, it waits until the simulation time is advanced by the SystemC simulation kernel to the time of the next activity. It is possible that an external event coming from the SystemC domain, for example, an external access, creates an activity before the next scheduled one, in which case it wakes up the GVSoc thread for proper treatment of the external event. As shown in Listing 2, the *Top Time Engine* of GVSoc waits for either an amount of time when the next activity is known and internally scheduled or an activity that may happen before that originated in the SystemC domain. It is also possible that GVSoc scheduler has nothing to schedule and is just waiting for an event (using the same mechanism) which is used to wake it up to execute the new activity.

Listing 2: Main changes in the GVSoC scheduler

```

1 while(1) {
2   if (!first_client) {
3     if (stop_req || locked)
4       break;
5     // Wait for events from the SystemC domain
6     wait(sync_event);
7   } else {
8     int64_t now = (int64_t)sc_time_stamp().to_double();
9     int64_t ne_time = first_client->next_event_time;
10    vp_assert(ne_time >= now, NULL, NULL);
11    // Wait next event or an asynch event from SystemC
12    wait(ne_time - now, SC_PS, sync_event);
13    now = (int64_t)sc_time_stamp().to_double();
14    if (now == first_client->next_event_time)
15      break;
16  }
17 }

```

4.3 GVSoC TLM-2.0 Bridge

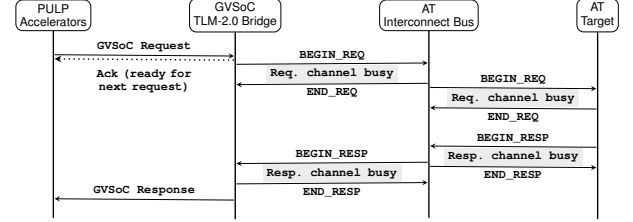
The *GVSoC TLM-2.0 Bridge* consists of a transactor component that converts *GVSoC* requests into TLM transactions and vice versa. Hence, it allows native *GVSoC* models to communicate with any TLM-compliant component. The bridge is TLM-2.0 base protocol compliant, hence it respects the exclusion rules defined by the protocol. The approximately-timed coding style is adopted, therefore transactions are initiated in the forward path using the non-blocking transport interface. Moreover, it honors timing annotations and supports responses via both backward and return paths, including early completion and implicit end request (*END_REQ*).

For the sake of illustration, in a typical scenario a memory access request from the DMA engine is a few kilobytes in size, then it relies in a DRAM controller to perform multiple memory requests. In each request 64 bytes of data are exchanged between DRAM controller and the DRAM through a 64-bit wide data bus in a burst consisting of 8 beats, each beat 8 bytes. The *GVSoC TLM-2.0 Bridge* breaks a memory access request into multiple parts adequate to its immediate target. For that, it allocates new TLM transaction objects on-demand using a *memory manager*, which contains a pool of transaction objects for reuse. In the context of TLM, a memory manager is a common/mandatory C++ class/object used to avoid recurrent construction and deletion of transaction objects. Moreover, the bridge uses TLM generic payload extensions to check the integrity of transactions and the request they derive during the communication process between PULP components modeled by *GVSoC* and SystemC modules. Ignorable extensions are used as they are transparent to other components allowing transactions to be normally routed through any number of interconnect components in their way to the target.

Table 1 shows the relevant fields of the *GVSoC* request along with their equivalents in the TLM transaction. The *is_write* flag of the *GVSoC* request is used to define the TLM command, read or write, the *size* defines the number of parts a *GVSoC* request is split into. Each of the parts is a TLM transaction. The *data_length* of TLM transaction is a configuration that must fit requirements from the target side. In order to avoid movement of large amounts of data, the same data region pointed by the *data_pointer* in a *GVSoC* request is directly referenced by TLM transactions. The pointer to the base of the data region is incremented with an offset relative to the part of the *GVSoC* request a transaction corresponds to. Similarly, the *address* field is reused and incremented by an offset. Flags are

Table 1: GVSoC requests and TLM transactions

GVSoC Request	TLM Transaction	Observations
<i>is_write flag</i>	-	Request type
<i>size</i>	<i>data_length</i>	1:N
-	<i>command</i>	Defined by <i>is_write</i>
<i>address</i>	<i>address</i>	1:N (base + offset)
<i>data_pointer</i>	<i>data_pointer</i>	1:N (pointer + offset)
<i>flags</i>	-	-
-	<i>byte_enable</i>	-
-	<i>streaming_width</i>	-
-	<i>dmi_allowed</i>	-
-	<i>extensions_pointer</i>	Integritycheck and routing

**Figure 2: The Bridge and Interconnect Bus allow multiple transactions in-flight and flow control in both directions.**

GVSoC-specific, therefore they are not propagated. Other fields of the TLM transaction such as *byte_enable* and *streaming_width* are initialized to their default values and *dmi_allowed* is not used in this work since the memory model design is approximately-timed, therefore a higher accuracy level that is more suitable for architectural exploration.

4.4 TLM-2.0 AT Interconnect Bus

The *TLM-2.0 Approximately-Timed Interconnect Bus* models a SoC interconnect memory mapped bus. The bus is the main junction point for the *GVSoC Bridge* and other TLM compatible modules/frameworks. It allows multiple connections in both upstream and downstream directions with *multi_passthrough sockets*, hence the number of initiators and targets is defined dynamically based on socket bindings. Moreover, it is designed to support address decoding functionality, but in its current state of development there is “dummy” decoding that serves as placeholder to be extended on-demand.

Figure 2 provides an illustration on how requests and responses are propagated. The four-phases of the TLM non-blocking protocol allows implementation of flow control in both directions.

5 EVALUATION

In the following we present experiments used to evaluate the functionality of the coupling. Different SystemC/TLM compatible components and simulators are used together to form a heterogeneous simulation framework/test scenarios in which proper scheduling and treatment of events is mandatory. Moreover, we use the open source *TLM-2.0 Base Protocol Compliance Checker* to analyse one million transactions that pass through our new components.

5.1 Heterogeneous Modeling Framework

As part of our evaluation we build the heterogeneous modeling framework shown in Figure 3. The framework consists in a SystemC/TLM simulation environment where RISC-V accelerators from

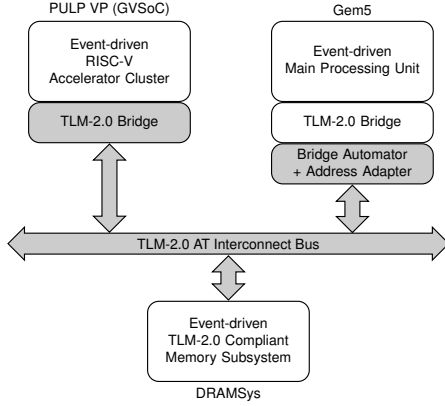


Figure 3: Example of a heterogeneous modeling framework. New components are highlighted in gray.

PULP (enabled by this work) are combined with other simulators, namely DRAMSys [14] and gem5 [2]. An overview of the simulators used along with open source components created to support the coupling is presented in the following.

DRAMSys [14] is a flexible memory subsystem design space exploration framework that consists of models reflecting the DRAM functionality, power consumption with a bind to DRAMPower [3], temperature behaviour and modeling of DRAM cell data retention. The framework provides models to build a complete memory subsystem, such as arbitration module, elaborate memory controller and multiple DRAM models. DRAMSys is particularly interesting for it is an AT TLM-compliant framework, hence it can be used for memory architectural DSE.

The gem5 simulator [2] is a modular platform for architecture research, encompassing system-level architecture as well as processor microarchitecture [9]. It provides models of diverse system components including CPUs, buses, caches, DRAM controllers [11] along with peripheral devices. RISC-V, x86, ARM, ALPHA and MIPS are among the ISAs supported [8]. Moreover we profit from the accumulated achievements of [18] to couple gem5 to our environment and utilities provided by [30] facilitate the setup.

We create the *Gem5 TLM Bridge Automator* and an *Address Adapter*. The former parses gem5 configuration files to proper instantiate and initialize the key components provided by [18] such as the *Gem5SimControl* object. Moreover, it instantiates the *Address Adapter* component that can be used to convert memory addresses originated in gem5 to the address range of the *Interconnect Bus* and vice versa. Furthermore, we also provide a *Memory Manager* along with an example TLM-2.0 target component.

To evaluate the coupling we execute different workloads in the simulation framework shown in Figure 3. A RISC-V core within the PULP accelerator cluster executes a memory access pattern first writing values to the external DRAM (end point of the memory subsystem) and later reading them back from the DRAM. At the same time, gem5 executes Linux in a quad-core ARM based system that uses the aforementioned DRAM as main memory. In this case study the system simulated in gem5 and the RISC-V in the PULP VP use separate memory address ranges, hence different areas of the

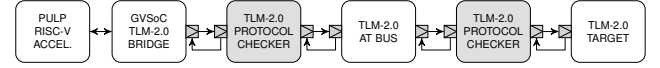


Figure 4: TLM-2.0 Base Protocol Compliance Checker usage.

Listing 3: Instantiation, binding and config (10⁶ checks).

```
1 void scdomain::elab() {
2   bridge = new gvsoc_tlm_br("br", this, ACCEPT_DELAY_PS, BPA);
3   atbus = new ems::atbus("bus");
4   pcbb = new tlm_utils::tlm2_base_protocol_checker<>("pcbb");
5   pcbt = new tlm_utils::tlm2_base_protocol_checker<>("pcbt");
6   pcbb->set_num_checks(1000000);
7   pcbt->set_num_checks(1000000);
8   bridge->isocket.bind(pcbb->target_socket);
9   pcbb->initiator_socket.bind(atbus->tsocket);
10  atbus->isocket.bind(pcbt->target_socket);
11  #ifdef __VP_USE_SYSTEMC_GEM5
12    std::string cfg = get_config_str("tlm/gem5-config");
13    g5automator = new ems::gem5_automator("g5automator", cfg);
14    for (auto adpt : g5auto->adapters)
15      adpt->isocket.bind(atbus->tsocket);
16  #endif /* __VP_USE_SYSTEMC_GEM5 */
17  #ifdef __VP_USE_SYSTEMC_DRAMSYS
18    std::string sim = get_config_str("tlm/dramsys-config");
19    dramsys = new DRAMSys("DRAMSys", sim);
20    pcbt->initiator_socket.bind(dramsys->tsocket);
21  #else
22    tgt = new ems::at_target("t", ACCEPT_DELAY_PS, LATENCY_PS, BPA);
23    pcbt->initiator_socket.bind(tgt->tsocket);
24  #endif /* __VP_USE_SYSTEMC_DRAMSYS */
25 }
```

DRAM. Interactions with the memory subsystem consist of TLM transactions.

Requests from the RISC-V core in the PULP VP domain are translated into TLM transactions by the *GVSoc TLM-2.0 Bridge* and forwarded to the *TLM-2.0 AT Interconnect Bus*. TLM transactions coming from gem5, pass through our *Gem5 TLM Bridge Automator*, which instantiates the control module provided by gem5, and through our *Address Adapter* before being forwarded to the *TLM-2.0 AT Interconnect Bus*. Next, transactions from both domains go through the *TLM-2.0 AT Interconnect Bus* to the memory controller and to the DRAM, the last two implemented in DRAMSys.

Moreover, the third party simulators (gem5 and DRAMSys) are compiled as libraries and linked with the PULP VP platform to form a simulation environment on top of the SytemC simulation kernel. The environment is used, not only to test the scheduling capabilities of the modified simulation engine, but also as realistic stimuli generator for testing communication.

As result we observe millions of transactions generated in both domains properly routed through our new components to reach their targets. Further, the Linux system in gem5 is responsive, the values written to and read back from the memory by the RISC-V core are compared and match.

5.2 TLM-2.0 Protocol Compliance Check

We adapt the case study described in 5.1 to use the open source *TLM-2.0 Base Protocol Compliance Checker*, created by John Aynsley from Doulos, to analyse at least 10⁶ transactions. Figure 4 illustrates how multiple instances of the protocol checker module are put in-line between the hops. Instantiation, binding and base configuration of relevant components are shown in Listing 3.

As result of our experiment, no protocol violations were observed.

Table 2: Execution Time (30 reps., Intel i7-4790 host CPU)

Memory Model	Maximum	Minimum	Average
PULP-VP Native	12.563 s	11.907 s	12.168 s
SystemC/TLM-2.0 AT ^a	17.792 s	16.621 s	17.071 s

^aImplies use of SystemC-based scheduling.

5.3 SystemC Simulation Overhead

To estimate the runtime overhead introduced by the coupling, which results in the execution of the PULP VP on top of the SystemC simulation kernel, we create equivalent simulation scenarios with and without SystemC that we summarize in the following¹.

A workload, which consists of a simple compute kernel for matrix addition, manipulates data from a memory component accessed via DMA engine. The workload iterates 2000 times over a 100×100 matrix executing approximately 200 million instructions. Moreover, 8 RISC-V cores are used, hence there is also inter-core synchronization as the DMA is managed by one core, and each core needs to fetch a *chunk of data* to process (the DMA ensures coarse-grained accesses to the memory). The RISC-V cores within the cluster work on local scratchpad memory (128 KB) organized in 8 banks with a logarithmic interconnect with an average contention ratio of 10%. The matrices operated are stored in a memory component within the SoC, but outside the cluster, so there is a double buffering between the two memories for both input and output using the DMA.

Two versions of the memory component are created: (1) Native PULP-VP memory module. (2) SystemC/TLM-2.0 target memory, that is a simple implementation but in AT coding-style, non-blocking transport, four-phases protocol.

From our experimental results presented in Table 2 we observe about 40% increase in the simulation time when using our coupling.

6 CONCLUSION

The PULP architecture is resourceful with high relevance in industry and academia. The PULP Virtual Platform provides sophisticated models of the RISC-V accelerators and peripherals that accumulate considerable research/engineering efforts, hence the reuse of such models is of substantial importance.

In this work we presented, for the first time, a coupling that enables co-simulation of RISC-V accelerators cluster from the PULP VP in a SystemC environment. To reduce the engineering efforts in further couplings, all components created are open source and available for download. Furthermore, we evaluated the coupling and provided a case study that can be used as entry point for the creation of further heterogeneous simulation frameworks.

As future work we aim at reducing the simulation time overhead introduced by our coupling.

ACKNOWLEDGMENTS

This work was partially funded by the EU OPRECOMP project under grant agreement No. 732631 (<http://www.oprecomp.eu>) and also supported by the *Fraunhofer High Performance Center* for

Simulation- and Software-based Innovation. We acknowledged the support of Synopsys Inc..

REFERENCES

- [1] Fabrice Bellard. 2005. QEMU, a Fast and Portable Dynamic Translator. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference (ATEC '05)*. USENIX Association, Berkeley, CA, USA, 41–41. <http://dl.acm.org/citation.cfm?id=1247360.1247401>
- [2] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The gem5 simulator. *SIGARCH Comput. Archit. News* 39, 2 (Aug. 2011), 1–7. <https://doi.org/10.1145/2024716.2024718>
- [3] Karthik Chandrasekar, Christian Weis, Yonghui Li, Sven Goossens, Matthias Jung, Omar Naji, Benny Akesson, Norbert Wehn, and Kees Goossens. 2011. DRAMPower: Open-source DRAM Power & Energy Estimation Tool. <http://www.drampower.info>.
- [4] Amir Charif, Gabriel Busnot, Rania Mameesh, Tanguy Sassolas, and Nicolas Ventroux. 2019. Fast Virtual Prototyping for Embedded Computing Systems Design and Exploration. In *Proceedings of the Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO '19)*. ACM, New York, NY, USA, Article 3, 8 pages. <https://doi.org/10.1145/3300189.3300192>
- [5] Johan Eker, Jorn W Janneck, Edward A. Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Sonia Sachs, Yuhong Xiong, and Stephen Neuendorffer. 2003. Taming heterogeneity - the Ptolemy approach. *Proc. IEEE* 91, 1 (2003), 127–144. <http://chess.eecs.berkeley.edu/pubs/488.html>
- [6] Luca Fossati, Thomas Schuster, Rolf Meyer, and Mladen Berekovic. 2013. Socrocket: a virtual platform for soc design. *DAta System In Aerospace (DA-SIA)* (2013).
- [7] GeenaWaves. 2018. GAP8. <https://en.wikichip.org/wiki/greenwaves/gap8>. Last accessed 10 Nov 2019.
- [8] gem5.org. 2011. gem5: Supported Architectures. www.gem5.org/Supported_Architectures. Last accessed 07 Nov 2019.
- [9] gem5.org. 2011. The gem5 Simulator. http://gem5.org/Main_Page. Last accessed 07 Nov 2019.
- [10] Google. 2019. OpenTitan. <https://opensource.googleblog.com/2019/11/opentitan-open-sourcing-transparent.html>. Last accessed 10 Nov 2019.
- [11] A. Hansson, N. Agarwal, A. Kolli, T. Wenisch, and A.N. Udupi. 2014. Simulating DRAM controllers for future system architecture exploration. In *ISPASS 2014 - IEEE International Symposium on Performance Analysis of Systems and Software*. 201–210. <https://doi.org/10.1109/ISPASS.2014.6844484>
- [12] V. Herdt, D. Große, H. M. Le, and R. Drechsler. 2018. Extensible and Configurable RISC-V Based Virtual Prototype. In *2018 Forum on Specification Design Languages (FDL)*. 5–16. <https://doi.org/10.1109/FDL.2018.8524047>
- [13] IEEE. 2012. IEEE Standard for Standard SystemC Language Reference Manual. *IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005)* (Jan 2012). <https://doi.org/10.1109/IEEESTD.2012.6134619>
- [14] Matthias Jung, Christian Weis, and Norbert Wehn. 2015. DRAMSys: A Flexible DRAM Subsystem Design Space Exploration Framework. *IPSS Transactions on System LSI Design Methodology* 8 (2015), 63–74. <https://doi.org/10.2197/ipsjtsldm.8.63>
- [15] Lukas Jünger, Jan Henrik Weinstock, Rainer Leupers, and Gerd Ascheid. 2019. Fast SystemC Processor Models with Unicorn. In *Proceedings of the Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO '19)*. ACM, New York, NY, USA, Article 2, 6 pages. <https://doi.org/10.1145/3300189.3300191>
- [16] Asif Ali Khan, Fazal Hameed, and Jeronimo Castrillon. 2018. NVMain Extension for Multi-Level Cache Systems. In *Proceedings of the Rapido'18 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO '18)*. ACM, New York, NY, USA, Article 7, 6 pages. <https://doi.org/10.1145/3180665.3180672>
- [17] T. Kuhn, T. Forster, T. Braun, and R. Gotzhein. 2013. FERAL — Framework for simulator coupling on requirements and architecture level. In *2013 Eleventh ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE 2013)*. 11–22.
- [18] C. Menard, J. Castrillon, M. Jung, and N. Wehn. 2017. System simulation with gem5 and SystemC: The keystone for full interoperability. In *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. 62–69. <https://doi.org/10.1109/SAMOS.2017.8344612>
- [19] Daniel Mueller-Gritschneider, Keerthikumara Devarajegowda, Martin Dittich, Wolfgang Ecker, Marc Greim, and Ulf Schlichtmann. 2017. The Extendable Translating Instruction Set Simulator (ETISS) Interlinked with an MDA Framework for Fast RISC Prototyping. In *Proceedings of the 28th International Symposium on Rapid System Prototyping: Shortening the Path from Specification to Prototype (RSP '17)*. ACM, New York, NY, USA, 79–84. <https://doi.org/10.1145/3130265.3138858>
- [20] A. Q. Nguyen and H. V. Dang. 2015. Unicorn: Next Generation CPU Emulator Framework. <https://www.unicorn-engine.org/>. Last accessed 07 Nov 2019.

¹The interested reader may refer to [26] for further information on the PULP architecture.

- [21] Rodrigo Cortés Porto, Daniela Genius, and Ludovic Apvrille. 2019. Modeling and Virtual Prototyping for Embedded Systems on Mixed-Signal Multicores. In *Proceedings of the Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO '19)*. ACM, New York, NY, USA, Article 4, 7 pages. <https://doi.org/10.1145/3300189.3300193>
- [22] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls, and B. Jacob. 2011. The Structural Simulation Toolkit. *SIGMETRICS Perform. Eval. Rev.* 38, 4 (March 2011), 37–42. <https://doi.org/10.1145/1964218.1964225>
- [23] Alec Roelke and Mircea R Stan. 2017. Risc5: Implementing the RISC-V ISA in gem5. In *First Workshop on Computer Architecture Research with RISC-V (CARRV)*.
- [24] Robert Scheffel. 2018. *Simulation of RISC-V based Systems in gem5*. Master's thesis. Technische Universität Dresden.
- [25] Patrick Siegl, Rainer Buchty, and Mladen Berekovic. 2016. Towards Bridging the Gap Between Academic and Industrial Heterogeneous System Architecture Design Space Exploration. In *Proceedings of the 2016 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO '16)*. ACM, New York, NY, USA, Article 4, 6 pages. <https://doi.org/10.1145/2852339.2852343>
- [26] PULP Team. 2013. PULP Platform: Open hardware, the way it should be! <https://www.pulp-platform.org/>. Last accessed 10 Nov 2019.
- [27] Jan Henrik Weinstock. 2018. Virtual Components Modeling Library. <https://github.com/janweinstock/vcml>. Last accessed 07 Nov 2019.
- [28] Jan Henrik Weinstock, Rainer Leupers, and Gerd Ascheid. 2017. Accelerating MPSoC Simulation Using Parallel SystemC and Processor Sleep Models. In *Proceedings of the 9th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO '17)*. ACM, New York, NY, USA, Article 2, 6 pages. <https://doi.org/10.1145/3023973.3023975>
- [29] J. H. Weinstock, C. Schumacher, R. Leupers, G. Ascheid, and L. Tosoratto. 2014. Time-decoupled parallel SystemC simulation. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*. 1–4. <https://doi.org/10.7873/DATE.2014.204>
- [30] Éder F. Zulian. 2017. gem5.TnT: Tips and tricks to make your life easier when dealing with gem5. <https://github.com/tukl-msd/gem5.TnT>.