A Comparison of Virtual Platform Simulation Solutions for Timing Prediction of Small RISC-V Based SoCs

Felix Böseler, Jörg Walter, Behnam Razi Perjikolaei
OFFIS - Institute for Information Technology
Oldenburg, Germany
{felix.boeseler, joerg.walter, behnam.razi.perjikolaei}@offis.de

***Abstract*—In an electronic system level design flow, early timing predictions can be realized by employing virtual platform simulations in the transaction level abstraction. However, there is little research comparing virtual platform solutions for small RISC-V based System-on-a-Chip architectures. This paper tries to fill this gap with a qualitative and quantitative comparison focusing on timing prediction accuracy. Starting with a total list of 25 RISC-V simulators, we found that only three candidates fulfill a comprehensive set of requirements for research applications. For these candidates we performed a quantitative comparison in which we model a single-master bus-based System-on-a-Chip at different abstraction levels and measure each simulator's timing prediction accuracy and simulation speed.**

***Index Terms*—transaction level modeling, virtual platform, RISC-V**

I. Introduction

A small application-specific System-on-a-Chip (SoC) usually executes fixed periodical processes and is used in many devices, such as microwaves, washing machines, or even self-driving cars [1]. Performant and reliable predictions of timings in early SoC design phases can help to produce reliable hardware while also enabling more productive hardware design workflows. With such timing predictions external time-sensitive components, such as watchdogs, can be designed, refined, and simulated in parallel with the SoC according to an Electronic System Level (ESL) design flow (see [2]).

The so-called Transaction Level (TL) abstraction offers possibilities for timing predictions in early design phases by utilizing a more abstract modeling approach with Virtual Platforms (VPs) than in the classic Register Transfer Level (RTL) abstraction. Moreover, the RISC-V Instruction Set Architecture (ISA) is particularly interesting because of its modular/non-proprietary design and practical focus. This allows a wide usage of this ISA in embedded systems in the future and good ESL tool/simulator integration [3].

While the name TL suggests a fixed abstraction level, it is a huge abstraction space [2] with no general all-encompassing classification scheme (see [4] and [5]). Therefore, the officially listed 25 simulation solutions from the RISC-V foundation [6] which target the TL abstraction (called VP simulation solutions in this paper) differ greatly in their supported modeling abstractions and the resulting simulation speed.

Thus, in order to find an applicable VP for timing predictions of small RISC-V based SoCs, a systematic comparison of state-of-the-art VP simulation solutions is required. To the best of the authors' knowledge, such a comparison does not exist yet. Furthermore, while this paper focuses on a special class of SoCs, we hope that this paper sets an impulse for further VP simulation solution comparisons.

The rest of this paper is structured as follows. Section II presents related work. In Section III the qualitative comparison is conducted followed by the quantitative comparison in Section IV for a selected set of VP simulation solutions. Finally, Section V concludes this paper and provides an outlook.

II. Related Work

In [7] a qualitative comparison between the two popular simulation solutions OVPsim [8] and QEMU [9] is given. They also quantitatively examine the performance and accuracy when different quantum sizes are employed in the simulation. With a time quantum of $0.08\,\mathrm{ms}$ they achieve an $11.66\,\mathrm{times}$ faster simulation execution compared to their own cycle accurate CPU model while achieving a timing accuracy of $54\,\%$ for an image encoding task.

A very extensive quantitative examination of the timing accuracy for different interconnect abstractions is given in [10]. While the quantitative examination is very extensive, it does not compare the limitations or capabilities of multiple state-of-the-art VP simulation solutions.

The authors in [11] describe a case study regarding the accuracy of TL simulations where an RTL implementation of a watchdog timer is compared to its TL implementation.

Particularly interesting is the examination of a PicoRV32 core based system in [12] since it fits well into the motivation of small RISC-V based SoCs in this paper. The author conducts quantitative accuracy examinations by comparing TL simulation executions with real FPGA based executions of the same system. However, the quantitative comparison does not consider multiple TL abstractions or VP simulation solutions.

This work has been developed in the project VE-VIDES (project label 16ME0243K - 16ME0254) which is partly funded within the research program ICT 2020 by the German Federal Ministry of Education and Research (BMBF).

978-1-6654-7332-3/22/$31.00 ©2022 IEEE

III. Qualitative Comparison

Not all 25 simulation solutions officially listed as RISC-V simulators at [6] target the TL abstraction. However, even the solutions which target the TL differ considerably regarding their specific supported abstractions. Therefore, we choose to examine and compare the simulation solutions in two steps in this section. First, we examine whether the solutions fulfil essential requirements which ensure their practical free-to-use employment for timing predictions in the TL abstraction. After that, we conduct a comparison of the selected solutions regarding their specific supported TL abstractions and support of simulation speed optimizations, most notably temporal decoupling.

A. Essential Requirements

We choose a general differentiation between VP simulation solutions and TL modeling languages/frameworks, such as SystemC TLM-2.0 [5]. A VP simulation solution offers predefined configurable models, e.g., CPU models to ensure a practical usability.

We do not consider so-called host compiled simulation solutions (e.g., see [13]) since no strict hardware/software partition is given. For host compiled simulation solutions the software is annotated with delay constructs to reflect hardware timings rather than being executed by an Instruction Set Simulator (ISS) [13]. Because of practical reasons, we also consider the accessibility of a simulation solution as an important factor. Free-to-use simulation solutions offer a fast and reproducible testability of their features and limitations.

Because of these reasons, we choose to examine the officially listed 25 simulation solutions at [6] with the following essential requirements:

TL Abstraction The solution targets the TL abstraction.

Predefined Models The solution contains at least one configurable RISC-V CPU and interconnect model.

Timed Simulation The solution supports timed simulations with timing synchronization of multiple components in the same time domain.

Extensible The solution is extensible for custom components and platforms.

Free-to-use The solution is free-to-use for non-commercial applications.

We consider a simulation solution for further evaluation if it fulfills these requirements. The results of the examination can be seen in Table I. The first column shows the five essential requirements while the second column lists simulation solutions which miss a corresponding requirement.

Table I shows that many simulation solutions are untimed. Simulation solutions such as [9] follow an untimed approach for high performance and are also often considered as emulators, while solutions such as [22] purely focus on the untimed ISA simulation for educational purposes. We consider the four solutions *OVPsim* [8], *Renode* [32], *gem5* [33], and *RISC-V based Virtual Prototype* [34] (from now on called RISC-V VP) applicable. We did not include the simulation solutions *MARSS-RISCV* (Micro-ARchitectural System Simulator - RISCV) [35] and *DBT-RISE-RISCV* [36] for this examination as it was not possible to sufficiently evaluate their features and limitations for us due to a lack of documentation.

TABLE I
Essential requirements and simulation solutions which do not satisfy them (non-exhaustive)

Requirement	Solutions without requirement support
TL Abstraction	*FireSim* [14]
Predefined Models	
Timed Simulation	*QEMU* [9], *riscv-vm* [15], *TinyEMU* [16], *riscv-rust* [17], *jor1k* [18] *terminus* [19], *Spike* [20], *SweRV-ISS/Whisper* [21], *Ripes* [22], *vulcan* [23], *WebRISC-V* [24], *EmulsiV* [25], *RARS* [26], *Jupiter* [27]
Extensible	*riscvOVPsimPlus* [28]
Free-to-use	*BRVT* [29], *PQSE* [30], *VLAB* [31]

B. Modeling Styles

If an SoC designer deviates from the intended modeling style of a simulation solution, it causes an infeasible amount of modeling work. Naturally, the amount of work which an SoC designer considers infeasible must be defined for different use cases. In the case of this paper, we consider it infeasible to entirely re-implement CPU models or interconnect/communication structures as they generally exhibit a high complexity. We expect a VP simulation solution to provide sufficient configurability for predefined CPU and bus models to be usable for the simulation of different small SoCs.

The employed modeling style classification scheme is shown in Fig. 1 where each cell corresponds to a modeling style. As in [4] and [37] we choose a two axes classification scheme. We introduce the Instruction Delay Abstraction (IDA) for the y-axis since we defined the modeling styles, besides the predefined interconnect capabilities, through the predefined CPU model capabilities.

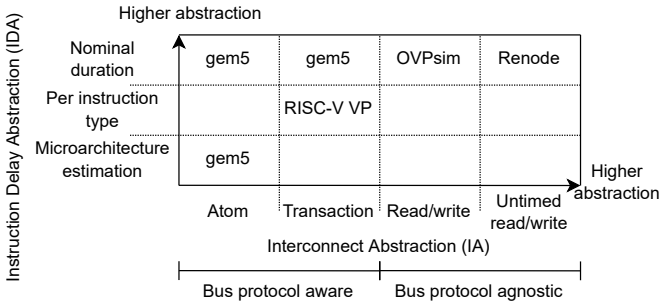

Fig. 1. Two-axes modeling style classification scheme for the VP simulation solutions OVPsim, Renode, RISC-V VP and gem5

In the lowest IDA (*microarchitecture estimation* IDA), CPU models include some microarchitecture details (e.g., a generic pipeline model with execution stage timings) but without constituting a full RTL CPU model. The *per instruction type* IDA only allows fixed instruction execution delays per

instruction type. Finally, the *nominal delay* IDA only allows the configuration of one nominal instruction execution delay for all instructions.

The x-axis of the classification scheme in Fig. 1 describes the interconnect abstraction. The synchronization granularity defines on which boundaries timing information and data is exchanged/synchronized.

The popular SystemC TLM-2.0 coding styles *Loosely Timed* (LT) and *Approximately Timed* (AT) describe programming language idioms rather than pure interconnect abstractions [5]. Thus, we opt to use a similar but more general definition of interconnect abstractions based on [38]. They differentiate non-repeatable phases called *atoms* in a transaction. Typical atoms are, for instance, *Init*, *Data Handshake*, and *Finalization*. We define the *atom* Interconnect Abstraction (IA) and *transaction* IA in the classification scheme on the lower abstraction end in Fig. 1.

Instead of a signal projection to generic transaction attributes which preserves bus protocol specific semantics, one can rely on protocol agnostic read/write accesses. For instance, with such an approach a wrapping burst transfer cannot be modelled as one transaction but must be modelled through multiple bus accesses. Renode and OVPsim use this abstraction. Therefore, we introduce the interconnect abstractions *read/write* IA and *untimed read/write* IA in Fig. 1. The synchronization also takes place on transaction boundaries. However, in the *untimed read/write* IA no timing synchronization is allowed.

Fig. 1 shows that Renode and OVPsim both target modeling styles with a *nominal delay* IDA. In both solutions the predefined CPU models take nominal Million Instructions per Second (MIPS) values for configuration. We notice that both solutions use a Dynamic Binary Translation (DBT) approach. For OVPsim we only consider free usable features and therefore do not include commercial features which can add microarchitecture estimation features to the CPU model.

RISC-V VP offers a 32-bit and 64-bit RISC-V CPU model, which supports the most common RISC-V extensions. As shown in Fig. 1 the solution focuses the *per instruction type* IDA allowing a configuration of instruction execution delays per instruction type. We only consider the officially published RISC-V VP version at [34]. The solution gem5 offers four major CPU models [39] from which the *simple* and *minor* CPU models are important for this paper. The *simple* CPU model targets the *nominal delay* IDA while the *minor* CPU model focuses the *microarchitecture estimation* IDA.

C. Temporal Decoupling

Table II shows the support of temporal decoupling for the VP simulation solutions OVPsim, Renode, RISC-V VP, and gem5. The second column describes the granularity of the decoupled modelled process in a VP. Since RISC-V VP is based on SystemC, the granularity is a SystemC thread. The solution gem5 only supports temporal decoupling for nodes in a distributed (network) system [39]. In OVPsim the granularity is based on the so-called virtual machines, which usually correspond to one modelled component (e.g., CPU or peripheral) [40]. Renode supports temporal decoupling based on hierarchical time sources and sinks which can have different time quantum sizes [41].

TABLE II
TEMPORAL DECOUPLING SUPPORT FOR THE EXAMINED VP SIMULATION SOLUTIONS

Solution	Granularity	Execution
RISC-V VP	SystemC thread	Sequential
gem5	Node in distributed system	Concurrent
OVPsim	Component (OVPsim virtual machine)	Sequential
Renode	Hierarchical time sinks/sources	Selectable

The third column in Table II shows the concurrent execution support for temporal decoupling. While OVPsim in the free version and RISC-V VP only support a sequential execution, Renode and gem5 support a concurrent execution.

Based on Table II we conclude that OVPsim and RISC-V VP have a very similar temporal decoupling support. However, in OVPsim all event executions are statically planned for a time quantum. The concrete time quantum sizes are dynamically adapted based on scheduled events since events can only be executed on quantum boundaries [40].

IV. Quantitative Simulator Comparison

In this section we conduct a quantitative comparison regarding timing prediction accuracy and simulation speed based on a case study. We choose the simulation solutions OVPsim, RISC-V VP and gem5 because of their timed interconnect abstraction support, which we consider crucial for accurate timing predictions (see Fig. 1).

A. Ground Truth

Fig. 2 shows the structure of the SoC which we use for comparing the timing accuracy and simulation speed of different simulation solutions. The SoC is a modified version of the PicoSoC from [42] and is described in Verilog, thus providing cycle-accurate timings, which we use as the ground truth for the comparison. The microarchitecture of the CPU features no pipelining and only a single speculative linear instruction prefetch per instruction.

As shown in Fig. 2, we use a Serial Peripheral Interface (SPI) based flash device as program memory to resemble a typical microcontroller setup. In contrast to the SRAM, which only has a configured latency of $1\,\mathrm{cycle}$ for any access, flash device accesses have considerably higher latencies of up to $7\,\mathrm{cycles}$ for sequential accesses (depending on prefetch time) and fixed $22\,\mathrm{cycles}$ for random accesses. The hardware-based implementation of the SHA-1 cryptographic hash function as memory-mapped device serves the role of an external peripheral. This allows us to look at VP timing and synchronization behavior with external system components. For the interconnect bus protocol we employ the minimal PicoRV32 native memory interface as the PicoRV32 does not support bus-pipelining capabilities or burst transfers.

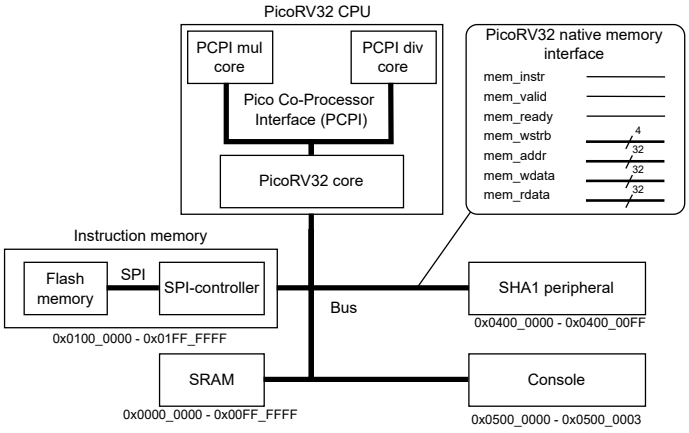

Fig. 2. Ground truth SoC structure based on the PicoRV32

The firmware consists of two alternatingly executed processes and a co-operative round-robin scheduler. We call each alternating process execution an execution cycle. The first process computes the Dhrystone benchmark [43] to simulate a common system workload. The second process sends input vectors to the SHA-1 peripheral device described in the previous section and then polls until results are ready. This firmware resembles a common workload for small SoCs in which we want to monitor the execution cycle timings as described in Section I to detect anomalies. We compile the firmware with the official RV32IMC GCC toolchain, thus supporting compressed instructions.

For the ground truth timings we measure the cycle-accurate execution times of eight execution cycles for each process with continuously and uniformly increasing durations in an RTL simulation (i.e., 16 execution cycles altogether). The increasing cycle times make it possible to evaluate the timing prediction accuracy of the VP simulation solutions for different observation frame sizes. The cycle times of two consecutive process executions (Dhrystone followed by the SHA-1 process) are very similar (relative difference smaller than 6 % for all eight cycle increments). We delimit the execution cycle timings though program counter changes in generated value change dump files.

B. Measurement Approach

1) Variants: We define 16 *variants* for the quantitative comparison based on a trade-off between the following considerations:

Abstraction coverage We want to cover a broad space for the quantitative comparison to evaluate the timing prediction accuracy of VPs on different abstraction levels.

Feasible realization A modeling style may restrict the realizability of a variant to a certain extent. We focus on variants with a broad feasible realizability in the modeling styles of the VP simulation solutions.

Comparability We want to realize similar variants in different simulation solutions so that a direct comparability of the timing prediction accuracies is given.

As a base for all variants, we assume an additive behavior of time induced by instruction execution delays in the CPU model and the transaction communication delays. The communication delay includes the data transfer time and target latency regarding SystemC TLM-2.0 terminology [5]. Such an assumption is realistic for a non-pipelined CPU like the PicoRV32 since memory accesses generally cause a stall.

We organize our benchmark variants in two dimensions: Computation and communication delay abstractions (shown in Fig. 3 where each cell corresponds to a variant). Besides the aforementioned trade-off approach, we chose the abstraction levels based on observations in initial experiments. The abstraction levels represent the most important factors that influence timing behavior and performance. For the computation delay abstraction, we define four classes that differ by how detailed individual instruction delays are modeled. These classes range from a single averaged delay (*1C* class) to individual delays for each instruction type (*AC* class). Intermediate classes group instructions depending on whether they are multiplication, memory access, branching (for *4C* class but not for *3C*), or other instructions.

For the communication delay abstraction we define four classes that differentiate between types of peripheral accesses. The *DIFF* class is the most detailed class and has separate delays for program memory accesses (sequential and random) and SRAM/peripheral accesses. The *DEL* class uses a single delay for program memory while the *MEMO* class merges program memory delays into the computation delays (average). Finally, the ZERO class uses a single global average delay merged into the computation delay value(s) of all instructions.

RISC-V VP offers a feasible realizability of all variants while OVPsim only supports two variants with a high abstraction (depicted through the colored rectangles in Fig. 3). In gem5 we differentiate between gem5 with the *simple* CPU model and the *minor* CPU model as the *minor* CPU model supports more variants. The defined variants target an abstraction level comparable to the LT coding style in SystemC TLM-2.0 because of the trade-off approach (feasible realizability and comparability).

2) Comparison Workflow: Fig. 4 describes the comparison workflow. First, we select all supported variants as shown in Fig. 3 for a simulation solution and iterate over them. The next instantiation step constructs a concrete VP from the timing characterization data and the current selected variant.

After this, we conduct a search over a range of time quantum values (quantum search) to find the biggest value which does not influence the timing accuracy in order to receive realistic simulation speed measurements. We run each variant a hundred times in a Linux-based virtual machine and calculate the average to determine the simulation time. The repeated executions are only required for the reliable measurement of simulation time since the simulated execution cycle durations are deterministic. Finally, we calculate the timing accuracy by comparing the simulated execution cycle timings in the VP with the ground truth. Note that we examine the variants for *gem5(simple)* and *gem5(minor)* separately.

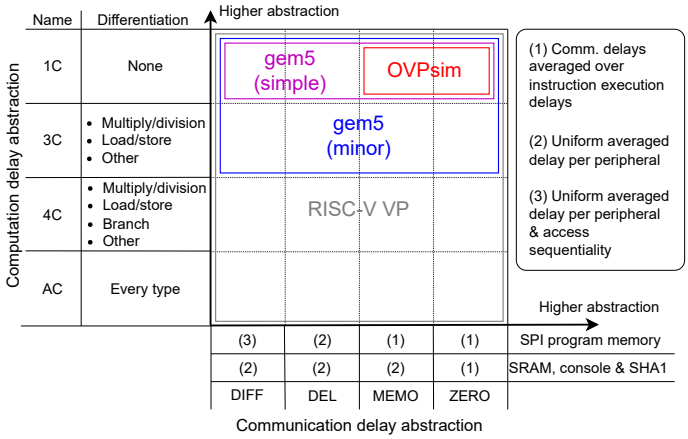

Fig. 3. Variants and their realizability in simulation solutions shown in a classification matrix

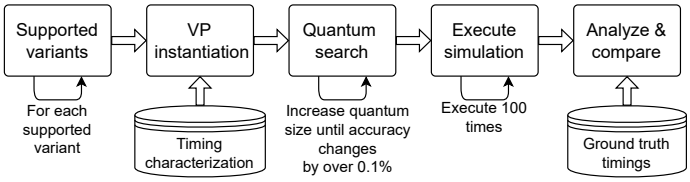

Fig. 4. Comparison workflow

We define the Step Percentage Error (SPE) of execution cycle step timings to be the relative error, i.e., the relative execution duration difference of an execution cycle in a TL simulation and the ground truth. Moreover, we employ an error aggregation of all SPE values generated by a VP simulation through a Root Mean Squared Error (RMSE) function. We denote this aggregation by writing $rmse(SPE)$.

In the quantum search step, beginning from a quantum size of $10\,\mathrm{ns}$ (clock duration of the ground truth SoC), we increase the quantum size by $10\,\mathrm{ns}$ in each step. We increase the quantum sizes until the $rmse(SPE)$ changes by over $0.1\,\%$ in comparison to the initial error value in which the quantum size of $10\,\mathrm{ns}$ was used.

3) Deviation Measurement: We try to separate the observed accuracy error into its systematic components. The first part is inherent to the abstraction level, while the second part corresponds to the implementation part, i.e., an inaccuracy we introduced during the construction of a specific measured VP.

System abstraction The chosen variant in which a system is modelled has an intrinsic theoretical upper bound for the accuracy. We will refer to a timing inaccuracy induced through this reason as intrinsic abstraction error.

Platform limitations An actual implementation of a benchmark variant in a given simulation solution might have less accuracy. We refer to an inaccuracy induced through this reason as realization error.

We are mainly interested in timing inaccuracies caused by realization limitations of the examined three simulation solutions. For instance, a variant may inherently yield a bad timing accuracy for predictions because of its high abstraction. However, we want to know how well a simulation solution supports the realization of a variant. In other words, we want to know the deviation between the theoretical best-case SPE of a variant versus the actual measured SPE of a benchmark in a specific VP. We call this deviation ΔSPE.

We make a-priori estimations of the SPE values for a variant and denote these values as SPE^{est}. These estimations use the instruction and transaction execution counts of the ground truth by calculating these with the parametrization values for a variant and summarizing these (assuming the timing behavior outlined in Section IV-B1). However, this estimation does not account for the interdependency of multiple execution threads. Such an interdependency occurs between the CPU execution thread and the SHA-1 peripheral through the polling behavior. Therefore, a ΔSPE value may include small intrinsic abstraction errors, which we, however, consider negligible for the most cases.

Fig. 5 summarizes this deviation-based measurement approach where SPE^{mes} describes a measured SPE value. It describes the approach for single SPE values. However, to evaluate the deviations for all simulated execution cycle steps in a VP simultaneously, we allow the RMSE aggregation of ΔSPE values.

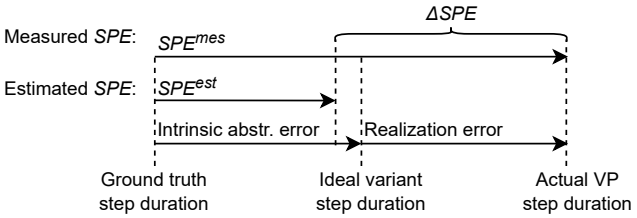

Fig. 5. Correlation between intrinsic abstraction error/realization error and different SPE types

C. Results

1) Estimated Intrinsic Abstraction Errors: Fig. 6 shows the $rmse(SPE^{est})$ values for each variant defined in Fig. 3. We denote the variants realized in a VP simulation solution with the following general syntax *solution[computation delay abstraction][communication delay abstraction]*. Since the depicted values in Fig. 6 are estimated and not generated through a concrete simulation in a simulation solution, we leave the *solution* part on the x-axis empty.

A higher instruction class granularity through a computation delay abstraction level does not necessarily yield better accuracy results (see Fig. 6) when the averaged instruction delays for an abstraction level are heterogenic. This is especially the case for *ZERO* and *MEMO* based variants as they average communication delays for the program memory over an entire instruction class. For instance, an instruction of type *a* might execute many sequential program memory accesses in the DHRY process while it only makes random program memory accesses in the SHA-1 process.

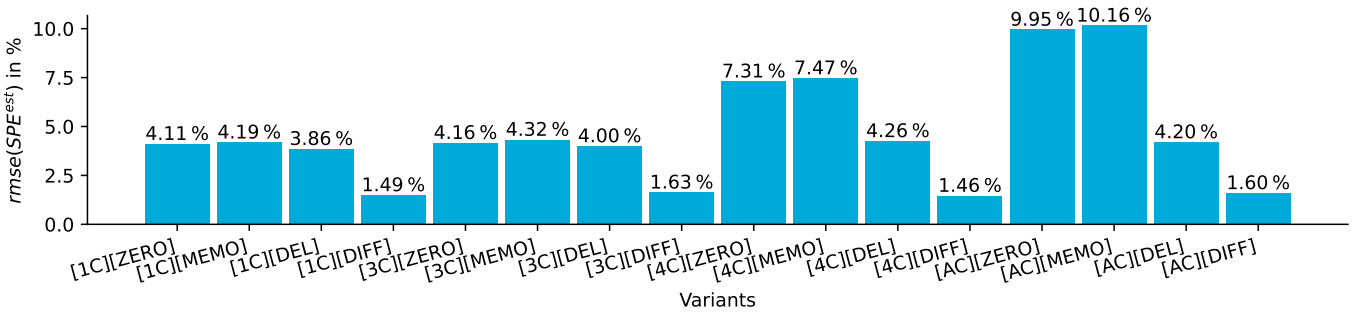

Fig. 6. Aggregated estimated intrinsic abstraction errors for all defined 16 variants

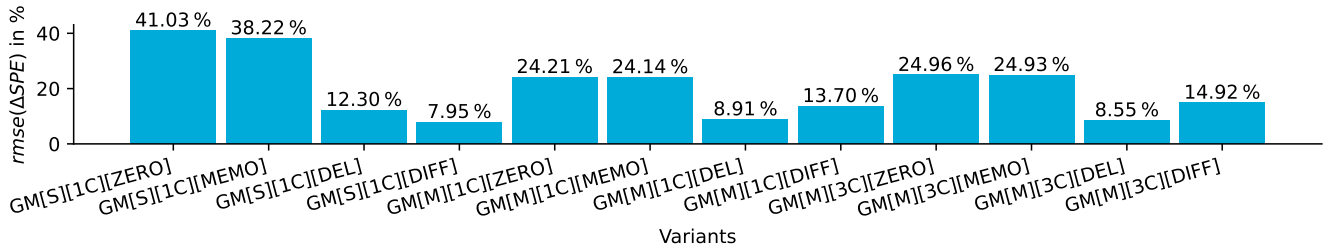

Fig. 7. Aggregated deviations for all variants realized in gem5

As should be expected, the *DIFF* based variants are the most promising variants for timing predictions since the aggregated estimated intrinsic abstraction error values are the lowest (around $1.5\,\%$). The following subsections examine the error added by the candidate platforms on top of this best-case error.

2) OVPsim Realization Errors: As shown in Fig. 3 OVPsim only supports two variants. Therefore, we describe the results for OVPsim in text form. We notice that for *OVP*[*1C*][*ZERO*] the deviation is only $1\,\%$, i.e., OVPsim is very close to the theoretical accuracy limit for this variant. For *OVP*[*1C*][*MEMO*] we notice a high additional error of $23\,\%$.

The likely reason is the fact that communication delays are implemented with two calls in OVPsim to a memory-mapped device. The first call blocks the ISS. After the specified delay the CPU component is notified and retries the load/store instruction with zero delay [40]. This means that the communication delays can only be accounted for on a rounded instruction length basis.

3) gem5 Realization Errors: Fig. 7 shows the aggregated deviations for each realized variant as VP in gem5 (abbreviated as *GM*). The [*M*] prefix denotes variant realizations with the *minor* CPU model while the [*S*] prefix denotes realizations with the *simple* CPU model. Because we could not find any direct state-of-the-art documentation regarding the 32-bit ISA support for RISC-V in gem5, we opted to use a 64-bit ISA realization. This decision causes significant realization errors because the firmware must be adapted in the realizations.

Fig. 7 shows that the results for the *1C* and *3C* computation abstraction levels are very similar. Only the communication delay configurations cause significant changes for the *minor* CPU based variants. The noticeable deviations ($8.55\,\%$ to $24.96\,\%$) are caused through the pipeline behavior of the *minor* CPU, which can only be partially configured to resemble the non-pipelined PicoRV32 execution behavior.

For all variant realizations in Fig. 7 based on the *simple* CPU the aggregated deviations are also noticeable ($7.95\,\%$ to $41.03\,\%$). For unaligned instructions, which occur due to the compressed ISA usage, the CPU model issues two transactions. While our custom buffer simulates the buffering behavior of the PicoRV32 CPU, the *simple* CPU model simulates a full instruction delay for each instruction fetch nevertheless.

4) RISC-V VP Realization Errors: Fig. 8 shows the aggregated deviations for each realized variant as VP in RISC-V VP (abbreviated as VP). We notice that the *ZERO* and *MEMO* based variant realizations with a *4C* or *AC* computation abstraction level show a deviation around $0.53\,\%$ to $0.58\,\%$. We consider these deviations to be intrinsic abstraction errors not captured by the SPE^{est} because we could not find any realization errors that may explain this behavior. However, for the *DEL* and *DIFF* based variant realizations Fig. 8 shows noticeable deviations. This is because the employed predefined CPU model in RISC-V VP does not feature the same instruction prefetch behavior as the PicoRV32 CPU.

5) Simulation Speed: As described in [39], gem5 only supports temporal decoupling for nodes in distributed (network) systems. We therefore only conducted the quantum search for the realized variants in RISC-V VP and OVPsim. According to the search process described in Section IV-B2, for RISC-V VP the quantum sizes are all between $140\,\mathrm{ns}$

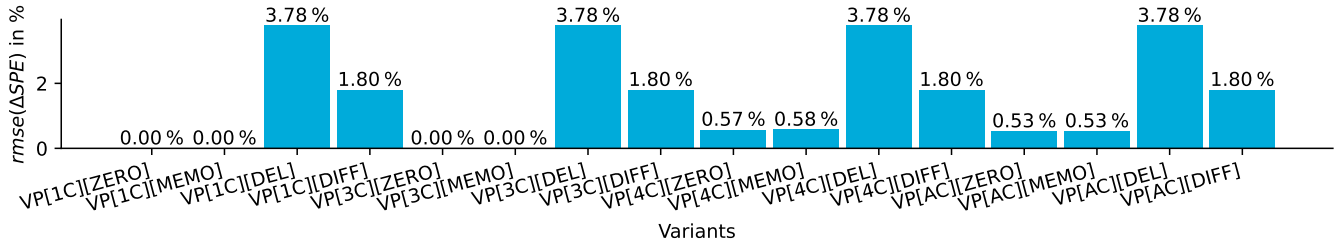

Fig. 8. Aggregated deviations for all variants realized in RISC-V VP

and 390 ns but there is no systematic correlation between quantum sizes and variants. However, for OVPsim the selected quantum sizes are 1150 ns for *OVP[1C][ZERO]* and 110 ns for *OVP[1C][MEMO]*. The quantum sizes directly relate to the highest employed peripheral delays being used.

Fig. 9 shows the averaged simulation durations for each simulation solution and variant. The figure shows that the RISC-V VP and gem5 variant realizations with the *simple* CPU model have a low execution duration of around 1 s to 1.25 s. The variant realizations in OVPsim require more time with around 2.5 s because of the DBT based ISS overhead, which is not amortized in such a short simulation. The longest execution durations are caused through variant realizations in gem5 with the *minor* CPU. Especially the realizations based on the *ZERO* and *MEMO* communication delay abstractions yield long execution durations because they cause a very high speculative prefetching overhead in the CPU model.

D. Evaluation and Discussion

Based on the results in the previous section, we conclude that the RISC-V VP simulation solution is the best selection for realizing small RISC-V based SoCs as VPs in the TL abstraction for most use cases since we assume that small SoCs often have a simple structure (e.g., single-master bus and low CPU microarchitecture complexity). The realized variants in this simulation solution exhibit low aggregated deviations (around 0 % to 3.78 %) and therefore low realization errors.

Furthermore, the simulation durations for the realized variants in the RISC-V VP solution are around 1.25 s and therefore 175 times as fast as the RTL simulation. The variants realized in gem5 with the *simple* CPU model have comparable durations but exhibit considerably higher aggregated deviations around 7.95 % to 40.3 %. The *OVP[1C][ZERO]* variant realization also exhibits a low aggregated deviation, but its execution duration is significant longer than for the RISC-V VP based realizations because of the DBT based ISS approach, which is problematic for short simulations.

The simulation solution RISC-V VP is simple and small because of its conformance to the SystemC TLM-2.0 standard and its RISC-V ISA focus. It supports the *transaction* IA, which is sufficient for most use cases of TL modeling for small SoCs – including multi-master systems with low contention levels (see [10]). For more specific cases where a pipelined CPU microarchitecture or a multi-master bus system with high contention levels should be simulated, we consider gem5 with the *minor* CPU a promising alternative. For long-running simulations OVPsim can offer a good performance gain while allowing a sufficient timing accuracy.

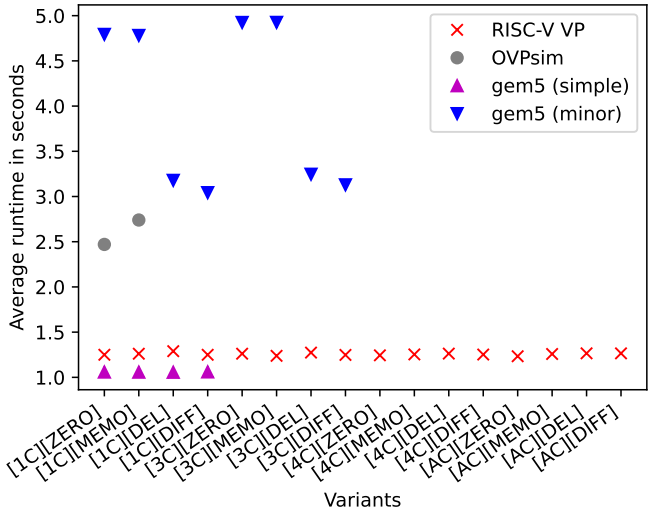

Fig. 9. Averaged execution durations for all simulation solutions and variants

As major limitations of this work we consider the following points: Our SoC exhibits a low hardware parallelism. Employing an estimation based approach for accuracy comparison as in this paper might be problematic for systems with a higher degree of parallelism. Such further examinations would also help to better validate advantages of gem5 and OVPsim (e.g., through long-running simulations). Furthermore, our case study only considers two representative CPU workloads. Finally, as described in Section IV-C3 we opted for a 64-bit realization in gem5 because of its unclear 32-bit support. This causes an inherent high realization error in the gem5 variants.

V. Conclusion

We showed through a qualitative comparison that the four state-of-the-art VP simulation solutions gem5, RISC-V VP, OVPsim, and Renode are applicable for timing predictions through VPs in early design phases. Through a subsequent quantitative comparison of the solutions gem5, OVPsim, and RISC-V VP, we found that RISC-V VP offers good timing

accuracy and performance results with little modeling work. The quantitative comparison conducted in this paper is based on a case study with a representative small RISC-V based SoC which features a low CPU microarchitecture complexity and a single-master bus interconnect. Furthermore, we showed that a more detailed computation delay abstraction level in a VP does not necessarily enhance the timing accuracy.

While we think we gave a good quantitative comparison in this paper, further case studies are required based on other SoC designs, CPU workloads, and VP realizations. We hope that this paper provides an impulse for such work in the future.

REFERENCES

[1] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 5th ed. Waltham, MA: Morgan Kaufmann/Elsevier, op. 2012.

[2] F. Kesel, *Modellierung von digitalen Systemen mit SystemC: Von der RTL- zur Transaction-Level-Modellierung*. München: Oldenbourg Wissenschaftsverlag, 2012.

[3] D. Patterson and J. L. Hennessy, *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*, RISC-V edition ed., ser. Morgan Kaufmann series in computer architecture and design. Boston, MA: Elsevier, 2018.

[4] S. Sfar, I. Bennour, and R. Tourki, "Transaction level models' structuring: From idioms to tlm-2," *Journal of Theoretical and Applied Information Technology*, vol. 76, pp. 178–201, 06 2015.

[5] "IEEE Standard for Standard SystemC Language Reference Manual," *IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005)*, pp. 1–638, 2012.

[6] "RISC-V Exchange," RISC-V International, Accessed: 16.02.2022. [Online]. Available: https://riscv.org/exchange

[7] P. E. S. Bomfim and A. Gerstlauer, "Integration of Virtual Platform Models into a System-Level Design Framework," Tech. Rep., August 2010, Accessed: 13.07.2021. [Online]. Available: https://repositories.lib.utexas.edu/handle/2152/ETD-UT-2010-05-1257

[8] *Imperas Installation and Getting Started Guide*, Imperas Software Limited, 2021, Accessed: 24.07.2022. [Online]. Available: https://www.ovpworld.org/documents/Imperas_Installation_and_Getting_Started.pdf

[9] "QEMU - a generic and open source machine emulator and virtualizer," Accessed: 16.02.2022. [Online]. Available: https://www.qemu.org/

[10] G. Schirner and R. Dömer, "Quantitative analysis of transaction level models for the AMBA bus," in *Proceedings of the Design Automation Test in Europe Conference*, vol. 1, 2006, pp. 1–6.

[11] S. Jayadevappa, R. Shankar, and I. Mahgoub, "A comparative study of modelling at different levels of abstraction in system on chip designs: a case study," in *IEEE Computer Society Annual Symposium on VLSI*, 2004, pp. 52–58.

[12] R. Scheffel, "Simulation of RISC-V based systems in gem5," M.S. thesis, Technische Universität Dresden, Dresden, 2018.

[13] D. Mueller-Gritschneder and A. Gerstlauer, "Host-compiled simulation," in *Handbook of Hardware/Software Codesign*, S. Ha and J. Teich, Eds. Dordrecht: Springer Netherlands, 2017, pp. 1–27. [Online]. Available: https://doi.org/10.1007/978-94-017-7358-4_18-1

[14] S. Karandikar *et al.*, "Firesim: FPGA-accelerated cycle-exact scale-out system simulation in the public cloud," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 29–42.

[15] A. Dodds, "A Small RISC-V Virtual Machine," Accessed: 31.08.2021. [Online]. Available: https://github.com/bit-hack/riscv-vm

[16] F. Bellard, "Tinyemu," Accessed: 24.08.2021. [Online]. Available: https://bellard.org/tinyemu/

[17] T. Aoyagi, "riscv-rust," Accessed: 24.08.2021. [Online]. Available: https://takahirox.github.io/riscv-rust/index.html

[18] S. Macke, "Online OR1K Emulator running Linux ," Accessed: 01.09.2021. [Online]. Available: https://github.com/s-macke/jor1k/

[19] Y. Li, "A riscv isa simulator in rust," Accessed: 01.09.2021. [Online]. Available: https://github.com/shady831213/terminus

[20] "Spike, a RISC-V ISA Simulator," RISC-V International, Accessed: 01.09.2021. [Online]. Available: https://github.com/riscv/riscv-isa-sim

[21] "chipsalliance/swerv-iss," CHIPS Alliance, Accessed: 31.08.2021. [Online]. Available: https://github.com/chipsalliance/SweRV-ISS

[22] M. B. Petersen, "A graphical processor simulator and assembly editor for the RISC-V ISA," Accessed: 01.09.2021. [Online]. Available: https://github.com/mortbopet/Ripes

[23] V. M. de Morais Costa, "RISC-V Instruction Set Simulator (Built for education)," Accessed: 01.09.2021. [Online]. Available: https://github.com/vmmc2/Vulcan

[24] R. Giorgi and G. Mariotti, "WebRISC-V: a web-based education-oriented RISC-V pipeline simulation environment," in *ACM Workshop on Computer Architecture Education (WCAE-19)*, Phoenix, AX, (USA), jun 2019, pp. 1–6. [Online]. Available: http://www.dii.unisi.it/ giorgi/papers/Giorgi19-wcae.pdf

[25] G. Savaton, "A visual simulator for teaching computer architecture using the RISC-V instruction set ," Accessed: 01.09.2021. [Online]. Available: https://github.com/Guillaume-Savaton-ESEO/emulsiV

[26] B. Landers, "RARS – RISC-V Assembler and Runtime Simulator ," Accessed: 01.09.2021. [Online]. Available: https://github.com/thethirdone/rars

[27] A. Castellanos, "RISC-V Assembler and Runtime Simulator," Accessed: 01.09.2021. [Online]. Available: https://github.com/andrescv/Jupiter

[28] "Imperas RISC-V riscvOVPsim reference simulator and architectural validation tests," Open Virtual Platforms/Imperas Limited, Accessed: 26.08.2021. [Online]. Available: https://www.ovpworld.org/riscvOVPsimPlus/

[29] "RISC-V Vector Environment," Barcelona Supercomputing Center, Accessed: 10.02.2022. [Online]. Available: https://repo.hca.bsc.es/gitlab/epi-public/risc-v-vector-simulation-environment

[30] "Post-Quantum Crypto IP: Software Support," PQShield, Accessed: 01.09.2021. [Online]. Available: https://pqsoc.com/software/

[31] "VLAB Virtual Platform Products," ASTC, Accessed: 01.09.2021. [Online]. Available: https://vlabworks.com/products/

[32] "Renode - Antmicro's virtual development framework for complex embedded systems," Antmicro Ltd, Accessed: 27.08.2021. [Online]. Available: https://github.com/renode/renode

[33] "gem5/gem5," Accessed: 24.01.2022. [Online]. Available: https://github.com/gem5/gem5

[34] V. Herdt, D. Große, P. Pieper, and R. Drechsler, "RISC-V based virtual prototype: An extensible and configurable platform for the system-level," *Journal of Systems Architecture*, vol. 109, p. 101756, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762120300503

[35] G. Kothari and G. Yuksek, "TinyEMU based full system cycle-level micro-architectural research simulator for single-core RISC-V systems," Computer Architecture and Power-Aware Systems Research Group, Accessed: 30.08.2021. [Online]. Available: https://github.com/bucaps/marss-riscv

[36] "Minres/dbt-rise-core," Minres, Accessed: 26.08.2021. [Online]. Available: https://github.com/Minres/DBT-RISE-Core

[37] L. Cai and D. Gajski, "Transaction level modeling: an overview," in *First IEEE/ACM/IFIP International Conference on Hardware/ Software Codesign and Systems Synthesis (IEEE Cat. No.03TH8721)*, 2003, pp. 19–24.

[38] W. Klingauf, R. Gunzel, O. Bringmann, P. Parfuntseu, and M. Burton, "Greenbus - a generic interconnect fabric for transaction level modelling," in *2006 43rd ACM/IEEE Design Automation Conference*, 2006, pp. 905–910.

[39] J. Lowe-Power *et al.*, "The gem5 simulator: Version 20.0+," 2020, Accessed: 28.05.2022. [Online]. Available: https://arxiv.org/abs/2007.03152

[40] *OVP Peripheral Modeling Guide*, Imperas Software Limited, 2021, Accessed: 27.08.2021. [Online]. Available: https://www.ovpworld.org/creating-behavioral-peripheral-components-using-bhmppm-apis-and-adding-them-to-platforms

[41] "Time framework," Antmicro Ltd., Accessed: 27.08.2021. [Online]. Available: https://renode.readthedocs.io/en/latest/advanced/time_framework.html

[42] C. Wolff, "PicoRV32 - A Size-Optimized RISC-V CPU ," Accessed: 23.08.2021. [Online]. Available: https://github.com/cliffordwolf/picorv32

[43] R. P. Weicker, "Dhrystone: A synthetic systems programming benchmark," *Commun. ACM*, vol. 27, no. 10, pp. 1013–1030, Oct. 1984. [Online]. Available: https://doi.org/10.1145/358274.358283