2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
OCTOBER 15-16, 2024

Efficient Workflow using Verilator for Processor Benchmarking in SystemC-based Automotive SoC Platforms

Johannes Sanwald, Andreas Mauderer, Mohammad Badawi, Javier Castillo, Jan-Hendrik Oetjens,
Robert Bosch GmbH, Reutlingen, GERMANY

Andreas Wieferink, Maryam Keeley, Tim Kogel, Synopsys Inc.

***Abstract*— This paper proposes an approach to efficiently evaluate processor cores in automotive SoC architectures as part of the design space exploration process, which overcomes obstacles such as limited availability of processor models and inaccurate interface timing, providing an efficient solution for evaluating processor cores from different vendors. The approach encompasses translation of processor RTL into a SystemC model via Verilator, which is then integrated into an easily customizable SystemC platform. A case study demonstrates this, by integrating the verilated open-source RISC-V core CVA6 into a system-level automotive SoC architecture using Synopsys Platform Architect. It shows that accurate performance evaluation with minimal cycle-count deviation and easy integration is achieved.**

Keywords—Design-Space Exploration; Processor; Performance Evaluation; RISC-V; SystemC; Verilator

I. Introduction

The increasing complexity of software in automotive edge devices, coupled with advancements in processing elements that enhance efficiency, scalability, and flexibility, poses a significant challenge in determining the most suitable processor and system architecture [1]. The introduction of processors featuring extensible instruction set architectures, such as RISC-V, further intensifies this challenge and consequently the need for efficient and precise exploration techniques. Such exploration techniques are essential for facilitating rapid decision-making and addressing time-to-market constraints. System-level exploration based on SystemC has become an established approach in addressing the problem of rapid architecture exploration, enabling a fast, yet accurate, sweep of available configurations [2]. However, the integration of processor models into the system-level environment, which is crucial for the success of these approaches, encounters various obstacles. These include the limited availability of cycle-accurate processor models from vendors (Arm for example discontinues the Arm Cycle Models [3]) as well as the absence of standardized interfaces, inaccurate timing, and the utilization of different Transaction Level Modeling (TLM) protocols in models provided by Intellectual Property (IP) vendors.

This paper presents an efficient method for evaluating processor cores from arbitrary vendors within a system-level environment that accurately represents an automotive System on a Chip (SoC). Instead of creating a new standard, we aim to create a new overarching workflow. It combines uniform deliveries of processor IP in the form of Register-Transfer Level (RTL) descriptions, which are translated to SystemC using Verilator. These models are integrated in a SystemC-based environment by using existing tools and workflows to create an efficient and accurate platform for performance evaluation and design space exploration.

We then demonstrate our proposed approach by integrating the verilated open-source RISC-V core CVA6 [4] into a system-level architecture for proof of concept. The efficiency and accuracy of the approach is compared against a pure RTL simulation in an RTL testbench.

II. State of the Art

There are multiple existing ways to assess processor performance in an SoC architecture context, for example full RTL integration, co-simulation-based approaches, and a system-level integration with SystemC. They differ in the effort required to bring up a target platform, in simulation performance and in the timing accuracy of the simulated system.

Fully integrating a processor core's RTL description into an RTL platform, while being inherently cycle-accurate, has drawbacks that include the significant effort required for integration, the slow speed of simulation, and the high maintenance and effort needed for accurately modeling latencies in RTL [5]. Integration demands extensive time and resources for bring-up and verification. Moreover, accurately modeling latencies on the signal level poses a significant challenge, as any inaccuracies can result in performance issues or system failures, necessitating additional maintenance and effort. Another drawback is the increased time consumption of simulating the entire system, leading to longer wait times.

Co-simulating processor core RTL descriptions alongside SystemC components alleviates some of the previously mentioned issues [6], one being the easier modeling of latencies in peripheral components, while still staying accurate. SystemC provides a higher level of abstraction, making it simpler to model complex components and systems. This ease of modeling enhances productivity and reduces the effort required for designing and verifying intricate systems. However, maintaining and setting up a co-simulation can be cumbersome, when two distinct simulation environments need to be coupled and synchronized.

By integrating a timing accurate SystemC processor model [7] or Instruction Set Simulator (ISS) [8] into a SystemC platform, we can circumvent the intricacies of mixed-level simulation and enhance simulation and integration speed. However, the accuracy of the simulation heavily relies on the fidelity of the processor model, or the ISS employed in this simulation scheme, which can vary significantly among different vendors. Moreover, the interfaces of such timing-accurate processor models often use proprietary protocols that are either not timing accurate or incompatible with other SystemC bus or memory IP. Similarly, not every ISS is cycle accurate. For example, Qemu uses dynamic binary translation (DBT) and is not cycle accurate, but can be enhanced to be more cycle accurate. But this is a very complex process, requiring much effort, and needs to be redone for each new processor or configuration [9].

To overcome the shortcomings of the previously described approaches, we use an approach that combines RTL processor descriptions translated to SystemC in SystemC architectures. The open-source tool Verilator compiles RTL code into highly optimized C++ or SystemC models, resulting in accurate and accelerated simulation performance [10], mitigating the lack of accuracy in the previous SystemC and ISS approach. Another option to translate RTL to SystemC is the commercially available tool Carbon [11], which has since been purchased by Arm and renamed to Arm Cycle Models [12]. The Cycle Models fulfill the requirements of accuracy and fast and seamless integration into a system level platform, but they have been discontinued by Arm and are not publicly available anymore. This makes Verilator a more attractive choice since it is openly available and benefits from continued development. While many publications exist, demonstrating the efficiency of simulating verilated processors in RTL architectures, in this work, we focus on combining verilated processors with a SystemC architecture. Using verilated RTL design and simulating it in a SystemC platform offers distinct advantages, including even faster simulation speed compared to co-simulation and higher accuracy compared to ISSs. This speed advantage allows for quicker verification and testing of the design. Additionally, RTL translated with Verilator into SystemC models enable seamless integration with other SystemC components, which are readily provided by IP suppliers. However, the problem of software debuggability will transfer from RTL to the verilated processor.

III. APPROACH

Our proposed systematic workflow, as illustrated in Figure 1, starts with an in-house or vendor-supplied RTL processor description that incorporates standardized bus interfaces, which will be integrated into a SystemC platform. To overcome the challenge posed by the unavailability of standardized cycle-accurate SystemC processor models from IP vendors, we generate a processor SystemC model using the open-source tool Verilator [13]. We configure Verilator to output a C++ description of the processor, along with a SystemC wrapper. This generated processor model is still at RTL abstraction level with cycle-accurate timing and pin accurate interfaces to the outside. This SystemC model is then imported into Synopsys Platform Architect [14]. A new refinement framework is analysing the pin interfaces and automatically detecting and grouping interface pins that belong to standardized AMBA interfaces like Advanced eXtensible Interface (AXI), AXIStream, Advanced High-performance Bus Lite (AHBLite) and Advanced Peripheral Bus (APB). Current precondition is that the pins are available as individual SystemC pins and named according to AMBA standard and conventions. By automatically connecting and parameterizing suitable pin transactors in a superior block hierarchy, a library block is generated by the new refinement framework which provides Fast Timed (FT) interfaces to the outside. The FT protocol is based on TLM2 but usually still allows for cycle accurate communication modelling. This library block is then seamlessly integrated with other IP models into a SystemC TLM2 FT platform, with FT models for memories, buses, and other peripherals. Therefore, the approach benefits from the easier connection, parametrization, design, and modeling of peripheral modules in SystemC at different levels of abstraction. Synopsys Platform Architect provides a wide range of such SystemC IP components. To enable software debugging, we can then temporarily replace the verilated core with an ISS that provides a software debug interface via a GDB-server. Another approach for debugging can be postmortem software debug solutions such as Synopsys Verdi HW/SW Debug [15], which rely on instruction traces.

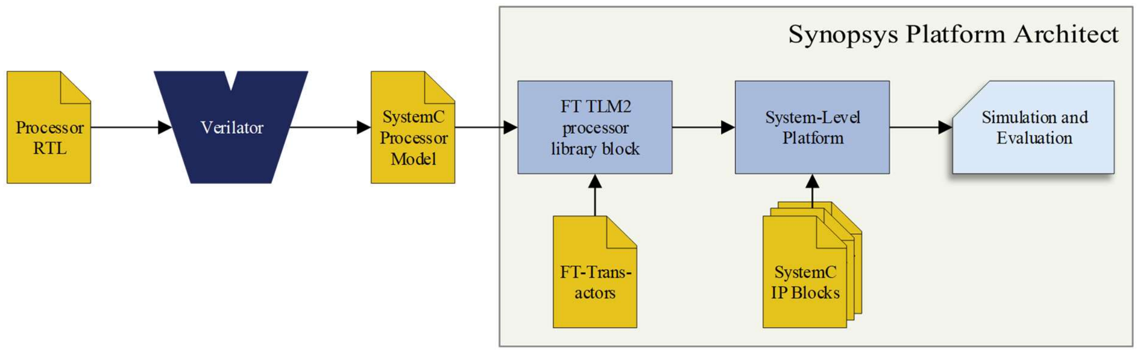


Figure 1. Integration flow from processor RTL to system-level automotive SoC

IV. CASE STUDY

To assess the effectiveness of our proposed flow, we apply it to the open-source processor CVA6, which is a six-stage, single-issue, in-order pipeline RISC-V core [16]. CVA6 offers many configuration options including bit width, instruction and data caches, an interface for custom instructions, and type of bus interface among others. We configured the CVA6 to have 64-bit data-width and instruction extensions for multiplication and division, atomic memory operations, single and dual precision floating point operations, and compressed instructions, as well as an AXI bus interface, which is suitable for demonstrating the applicability of the approach, an instruction cache, and a write-through cache for data. The core subsystem of the CVA6 is wrapped in a SystemVerilog wrapper, which splits the struct of AXI pins into separate Verilog in- and output signals and attaches an instruction trace module. The core is then translated via Verilator into C++ with a SystemC wrapper, which is in turn getting imported into Synopsys Platform Architect. The new refinement framework detects the set of RISC-V interface signals that belong to the AXI bus interface, and it picks a suitable FT-pin transactor from the Platform Architect library to attach them to. At simulation runtime, the FT pin transactors translate the detailed pin wiggling on the verilated processor model into a sequence of nonblocking TLM2 transport calls on the platform side, and vice versa. It is then connected directly via said AXI TLM interface to a generic SystemC TLM memory, while using pin connectors for reset- and clock-generators, as illustrated by Figure 2. The clock generator provides a clock signal for the processor and rest of the platform and the reset generator resets and starts the processor. The memory is used as unified instruction and data storage, separated into distinct regions through linker scripts, and attached directly to the processor without the use of a bus crossbar, to closely resemble an RTL testbench, which is used for validation and as reference for timing measurements. We decided for a generic TLM memory model to showcase the seamless integration and applicability of the approach. To determine the performance of the composite system and measure the execution time in cycle-counts of the CVA6 core, we execute the widely recognized benchmark Dhrystone.

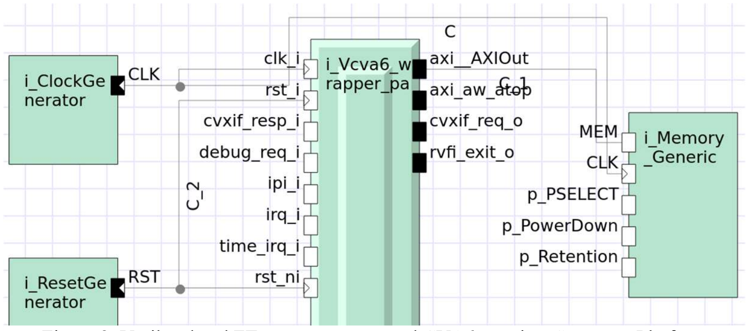


Figure 2. Verilated and FT-transactor wrapped CVA6 core in a Synopsys Platform Architect system-level architecture as used in the case study

A. Validation

To show functional correctness, we employ the Dhrystone benchmark to compare the functional behavior of the verilated CVA6 in a SystemC environment with its counterpart in an RTL testbench. By generating and comparing processor instruction traces from both platforms, we can verify that both cores execute the same order of instructions and have the same register contents, thereby establishing functional equivalence between the two implementations. Furthermore, Figure 3 presents two sets of AXI waveforms. The top section of the figure shows the read and write accesses of the SystemC platform with white background and RTL platform with black background for the entire execution of the Dhrystone benchmark with ten runs. One can notice that the same patterns emerge between the two platforms. The lower section of Figure 3 focuses on one of those patterns, specifically the fourth run of the Dhrystone benchmark. Because of the write-through cache, no read transfers are taking place during this run, which is why only the signals for write transfers are displayed. Again, similar patterns

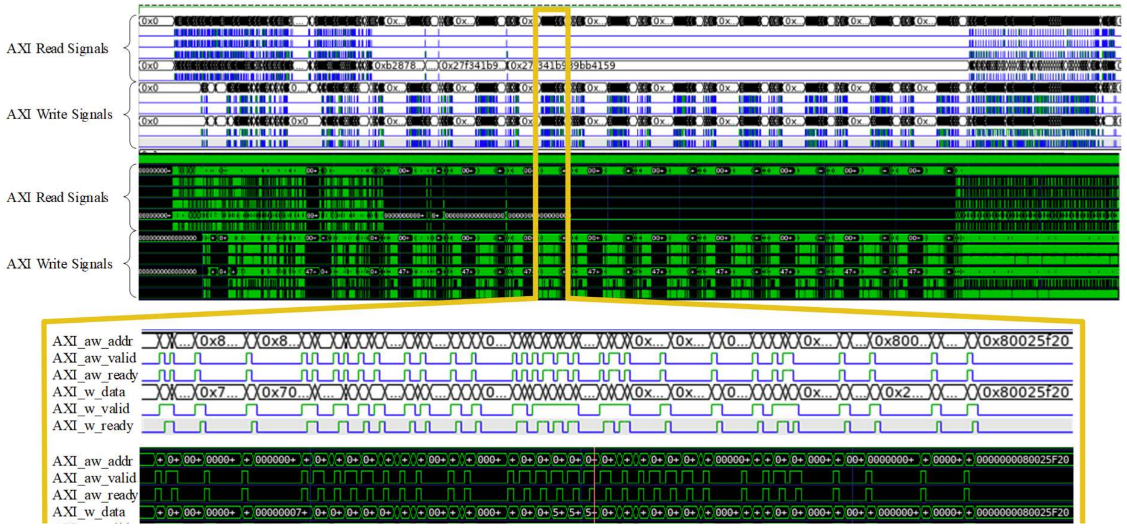


Figure 3. Waveforms of read and write AXI bus transfers for SystemC (white background) and RTL platform (black background) of Dhrystone benchmark with 10 runs;
Bottom: Zoomed in waveforms of read and write AXI bus transactions for SystemC (white background) and RTL platform (black background) of a single Dhrystone run;

can be observed, between the two platforms, with only slight deviations in transfer timings. Ultimately, the same addresses are accessed, and the same data values are written.

B. *Evaluation*

The evaluation of our proposed approach encompasses both quantitative and qualitative indicators. We measure the simulation speed and compare the accuracy of the exploration results, like execution time measured in clock-cycles, against an RTL testbench that resembles the SystemC platform closely. Additionally, we consider integration efforts, capability, and debuggability. Regarding integration effort, we assess the time required, in person-days, to generate the CVA6 core system using Verilator and integrate it into the system-level architecture.

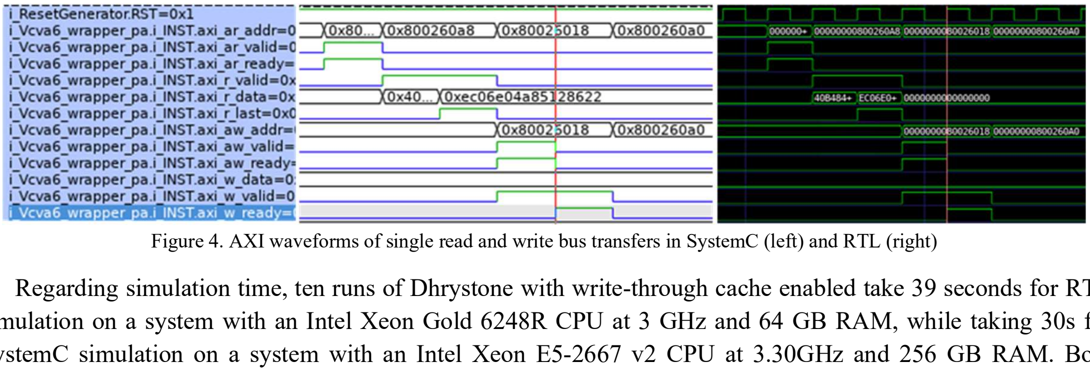
1) Timing Accuracy

To measure timing accuracy, we use the *mcycle* and *mcycleh* Control and Status Registers (CSRs), which are provided in the RISC-V Instruction Set Architecture (ISA). We execute the Dhrystone benchmark on both platforms and use the exact same binaries for configurations with write-through cache enabled and disabled. Table I presents the results of the benchmark. For ten runs in Dhrystone, the CVA6 with cache in RTL platform finishes in 10424 cycles, while the verilated CVA6 with cache in SystemC completes in 10444 cycles. This constitutes a 0,19 % deviation in cycle counts. We repeat the test with the cache deactivated, to examine the system under different bus loads. For the same benchmark, the CVA6 with deactivated cache in RTL platform finishes in 34310 cycles, while the verilated CVA6 without cache in SystemC completes in 34330 cycles. This constitutes a 0,06 % deviation in cycle counts. For both variants with different load on the bus, this approach produces reliable results with minimal deviation. The remaining deviation is due to a slight difference in modelled memories and not cycle accurate behavior in a small number of AXI transactions.

Table I. Cycle counts of Dhrystone (10 runs) benchmark on CVA6 in RTL and SystemC

	Cycle Count		
Processor Configuration	***SystemC***	***RTL***	***Relative Error***
Write-Through Cache	10444	10424	0,19 %
No Cache	34330	34310	0,06 %

Figure 4 depicts a single read and single write bus transfer in SystemC on the left and RTL on the right. The top six signals pertain to read transactions, while the bottom six belong to write transfers. One can detect the beginning of a write transfer, when an address is supplied in *axi_ar_addr* and *axi_ar_valid* and *axi_ar_ready* are asserted. The response comes one cycle later, when *axi_r_valid* is asserted and and data is supplied to *axi_r_data*. Once *axi_r_last* is pulled high, the transfer is complete. One can see that this process matches cycle for cycle between the two platforms. The same is true for write transfers, which start with an asserted *axi_aw_valid* and *axi_aw_ready* signal and conclude when the signal *axi_w_ready* is pulled high one cycle later. This determines that transfers in the TLM bus are modelled cycle accurately and match the ideal RTL bus transfers.


i_ResetGenerator.RST=0x1
i_Vcva6_wrapper_pa.i_INST.axi_ar_addr=0
i_Vcva6_wrapper_pa.i_INST.axi_ar_valid=0
i_Vcva6_wrapper_pa.i_INST.axi_ar_ready=
i_Vcva6_wrapper_pa.i_INST.axi_r_valid=0x
i_Vcva6_wrapper_pa.i_INST.axi_r_data=0x
i_Vcva6_wrapper_pa.i_INST.axi_r_last=0x0
i_Vcva6_wrapper_pa.i_INST.axi_aw_addr=
i_Vcva6_wrapper_pa.i_INST.axi_aw_valid=
i_Vcva6_wrapper_pa.i_INST.axi_aw_ready=
i_Vcva6_wrapper_pa.i_INST.axi_w_data=0
i_Vcva6_wrapper_pa.i_INST.axi_w_valid=0
i_Vcva6_wrapper_pa.i_INST.axi_w_ready=
0x80…
0x800260a8
0x80025018
0x800260a0
0x40…
0xec06e04a85128622
0x80025018
0x800260a0
000000+
00000000800260A8
0000000080026018
00000000800260A0
40B484+
EC06E0+
0000000000000000
0000000080026018
00000000800260A0
Figure 4. AXI waveforms of single read and write bus transfers in SystemC (left) and RTL (right)

Regarding simulation time, ten runs of Dhrystone with write-through cache enabled take 39 seconds for RTL simulation on a system with an Intel Xeon Gold 6248R CPU at 3 GHz and 64 GB RAM, while taking 30s for SystemC simulation on a system with an Intel Xeon E5-2667 v2 CPU at 3.30GHz and 256 GB RAM. Both simulations are executed on different systems and are therefore not directly comparable. They are only listed here for completion. However, it can be seen, that the simulation time is in the same order of magnitude.

2) Effort of Integration

The integration effort for our proposed approach is very low compared to RTL integration. The process involves wrapping the core in a custom wrapper to split the AXI-struct into separate channels and verilating the resulting core subsystem, which takes half a person day. The resulting SystemC model is wrapped with FT-transactors around the AXI channels and integrated into a simple SystemC platform. This process is automated and can be completed in a matter of hours, resulting in a low integration effort overall. Finally testing and evaluating the platform with benchmarks consumed 5 person days. The effort distribution is shown in Figure 5. Once this approach is established a team will need considerably less effort to integrate a processor core into a system-level architecture compared to the integration into an RTL platform. A similar effort of half a person day for wrapping the processor RTL is expended. However, integration into an RTL-based system required more effort. An experienced digital designer needed ten person days to integrate the processor core in a reference system and testbench. Testing and evaluation took a similar effort of 5 person days as with the SystemC platform. Highlighted in the yellow rectangle is the system integration effort. The large difference in integration effort shows the flexibility of the proposed approach.

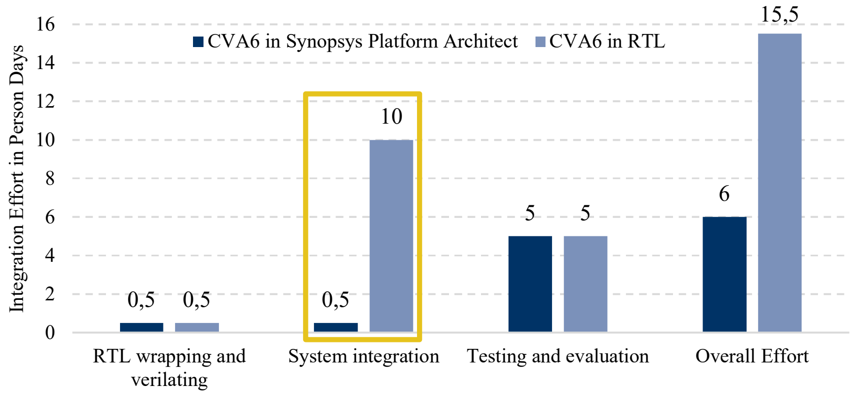


Figure 5. Integration effort in person days compared between CVA6 in Synopsys Platform Architect and in RTL

3) Debuggability

The inclusion of the instruction trace module in conjunction with the verilated core provides convenient access to instruction traces, significantly simplifying the debugging process compared to traditional waveform debugging. Additionally, the incorporation of the trace module, which must be provided by the RTL description but is readily available in most processor designs, enables the application of postmortem software debug solutions such as Synopsys Verdi HW/SW Debug [15], further enhancing the debuggability of the system. This capability facilitates efficient debugging and analysis.

V. Summary

This paper describes an approach to efficiently evaluate processor cores in a system level platform. The approach uses Verilator to translate RTL processor descriptions into SystemC models, for fast and seamless integration into SystemC platforms. This offers several advantages, including easier modelling and parametrization of peripheral modules, use of existing SystemC IP libraries, and faster integration compared to RTL. The paper demonstrates the successful implementation of an efficient approach for evaluating processor cores in automotive SoC architectures, showcasing the integration of the verilated open-source RISC-V core CVA6 into a system-level SoC architecture using Synopsys Platform Architect. The results illustrate that accurate performance evaluation and easier integration can be achieved, addressing challenges such as limited availability of processor models and inaccurate interface timing. Overall, our findings provide a promising solution for efficiently and rapidly assessing processor cores from different vendors.

VI. Future Work

The integration of CVA6 into SystemC was validated successfully, but it is important to widen the scope and study further processor variants and different instruction set architectures. Furthermore, it remains to show the flexibility of this approach by modelling a full automotive system-architecture on system-level and compare the approach to an equivalent, more complex RTL simulation. Additionally, further efforts need to be put into modelling and assembly of a comprehensive platform, on both RTL and SystemC, to examine the processor in both environments under similar conditions. With matching behavior between RTL and TLM, cycle accuracy can be achieved. The integrated trace unit provides adequate debuggability through instruction traces for software bring up, but to be able to use the platform extensively for software development, breakpoints and instruction stepping offers large benefits. This could be added in the future, by swapping out the verilated core for an ISS which provides a GDB-server.

Acknowledgement

The TRISTAN project, nr. 101095947 is supported by Chips Joint Undertaking (CHIPS-JU) and its members Austria, Belgium, Bulgaria, Croatia, Cyprus, Czechia, Germany, Denmark, Estonia, Greece, Spain, Finland, France, Hungary, Ireland, Israel, Iceland, Italy, Lithuania, Luxembourg, Latvia, Malta, Netherlands, Norway, Poland, Portugal, Romania, Sweden, Slovenia, Slovakia, Turkey and including top-up funding by Federal Ministry of Education and Research, BMBF (Germany).

References

[1] M. Gries, "Methods for evaluating and covering the design space during early design development," *Integration,* vol. 38, no. 0167-9260, pp. 131-183, 2004.

[2] L. Pomante, V. Muttillo, M. Santic and P. Serri, "SystemC-based electronic system-level design space exploration environment for dedicated heterogeneous multi-processor systems," *Microprocessors and Microsystems,* vol. 72, no. 0141-9331, p. 102898, 2020.

[3] G. Platt, "IP exchange and Cycle Models end-of-life update," Arm Limited, 25 May 2022. [Online]. Available: https://community.arm.com/arm-community-blogs/b/soc-design-and-simulation-blog/posts/ip-exchange-and-cycle-models-end-of-life-update. [Accessed 9 April 2024].

[4] F. Zaruba and L. Benini, "The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 27, no. 11, pp. 2629-2640, 2019.

[5] Y. S. Shao, B. Reagan, G.-Y. Wei and D. Brooks, "Aladdin: a Pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, Minneapolis, Minnesota, USA, 2014.

[6] S. S. Abrar, M. Jenihhin, J. Raik, S. Kiran A. and C. Babu, "Performance analysis of cosimulating processor core in VHDL and SystemC," in *2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Mysore, India, 2013.

[7] L. Bogdanov and R. Ivanov, "A Cycle-accurate Template Microprocessor Model of a Von Neumann Architecture Based on SystemC," in *2022 XXXI International Scientific Conference Electronics (ET)*, 2022.

[8] M. Monton, "A RISC-V SystemC-TLM simulator," *CoRR,* vol. abs/2010.10119, 2020.

[9] V. Herdt, D. Große and R. Drechsler, "Fast and Accurate Performance Evaluation for RISC-V using Virtual Prototypes," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, 2020.

[10] Y. Chi, X. Lin and X. Zheng, "Design of High-performance SoC Simulation Model Based on Verilator," in *Proceedings of the 2022 5th International Conference on Algorithms, Computing and Artificial Intelligence*, New York, 2023.

[11] R. Goering, "Carbon, CoWare link RTL to SystemC," EDN, 29 June 2004. [Online]. Available: https://www.edn.com/carbon-coware-link-rtl-to-systemc/. [Accessed 1 July 2024].

[12] A. Winstanley, "ARM to Offer Cycle-Accurate Virtual Prototyping for Complex SoCs Through an Asset Acquisition from Carbon Design Systems," Arm Limited, 20 October 2015. [Online]. Available: https://www.arm.com/company/news/2015/10/arm-to-offer-cycle-accurate-virtual-prototyping-for-complex-socs-through-an-asset-acquisition-from. [Accessed 1 July 2024].

[13] "Verilator," [Online]. Available: https://github.com/verilator/verilator. [Accessed 12 April 2024].

[14] "Synopsys Platform Architect," Synopsys, 25 June 2020. [Online]. Available: https://www.synopsys.com/verification/virtual-prototyping/platform-architect.html. [Accessed 12 April 2024].

[15] "Verdi HW/SW Debug," Synopsys, Inc., [Online]. Available: https://www.synopsys.com/verification/debug/verdi-hw-sw-debug.html. [Accessed 2 July 2024].

[16] "CVA6 RISC-V CPU," OpenHW Group, [Online]. Available: https://github.com/openhwgroup/cva6. [Accessed 22 June 2024].