2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

Accelerating Complex System Simulation using Parallel SystemC and FPGAs

Stanislaw Kaushanski, MINRES Technologies GmbH, Duisburg, Germany (stas@minres.com)
Johannes Wirth, ESA Group, TU Darmstadt, Darmstadt, Germany (*wirth@esa.tu-darmstadt.de*)
Eyck Jentzsch, MINRES Technologies GmbH, Munich, Germany (*eyck@minres.com*)
Andreas Koch, ESA Group, TU Darmstadt, Darmstadt, Germany (*koch@esa.tu-darmstadt.de*)

I. INTRODUCTION

The development of complex hardware/software systems is a time-consuming and resource-intensive process that requires extensive testing and verification. The growing complexity of the systems has made simulation for verification increasingly difficult and slow. As a result, development teams often face long turnaround times and high costs associated with simulation, which can delay time-to-market and limit innovation.

SystemC/TLM [1] technologies are widely used in the industry for fast system-level simulation. A major limitation of SystemC in terms of performance is the fact that the reference implementation is single-threaded. As the number of cores in computers has increased rapidly over the past decades, the ability to exploit host parallelism during a simulation becomes an important concern. In this paper we introduce RAVEN, a framework that addresses this challenge by providing a new simulation infrastructure. RAVEN allows to speed up SystemC simulations through coarse-grained parallelization.

RAVEN infrastructure not only allows for parallel SystemC simulation but also enables the co-simulation of SystemC and RTL mapped onto FPGA. The development of an embedded hardware project rarely starts from scratch. Typically, there are previous projects with reusable hardware components that should be integrated into the verification of new elements as soon as possible. Addressing this need, RAVEN aims to accelerate project development by enabling hybrid simulation. Hybrid simulation is a parallel simulation of a hardware design partly on FPGA and partly in SystemC. RAVEN enables the integration of existing RTL components using an FPGA, while simultaneously running new system components as a Virtual Prototype (VP) on the host.

The paper is organized as follows: Section II introduces RAVEN and outlines different simulation modes. Section III delves into the implementation of RAVEN's parallelization solution, focusing on the infrastructure. Section IV explains the synchronization mechanism used by RAVEN to ensure temporal alignment between the simulation partitions. Section V introduces RavenDSL, the domain-specific language facilitating integration of an FPGA-based partition. Section VI provides insights into related work, highlighting RAVEN's unique features. Section VII presents the validation and performance analysis results that demonstrate RAVEN's capabilities in various use cases. Finally, Section VIII concludes the paper by summarizing key findings and highlighting the benefits of RAVEN.

II. RAVEN

To enable the parallel simulation using RAVEN, it is necessary to divide the design under test (DUT) into partitions that can be simulated concurrently. The process of partitioning the DUT is dependent on the design's unique characteristics and is a decision made by the user. All the necessary manual adjustments to the DUT are performed just at the top level. Starting from an existing SystemC design, the user has to define the partition boundaries. All interfaces at the partition boundaries have to be attached to the RAVEN inter-thread connectors. More details about the connectors can be found in Section III.

Subsequently, these partitions can be executed in parallel, either in the SystemC environment or using an FPGA. For best results the system should be divided in such a way that individual partitions become as independent of each other as possible, in order to minimize the communication across partition boundaries. The rest of the existing design code remains unchanged.

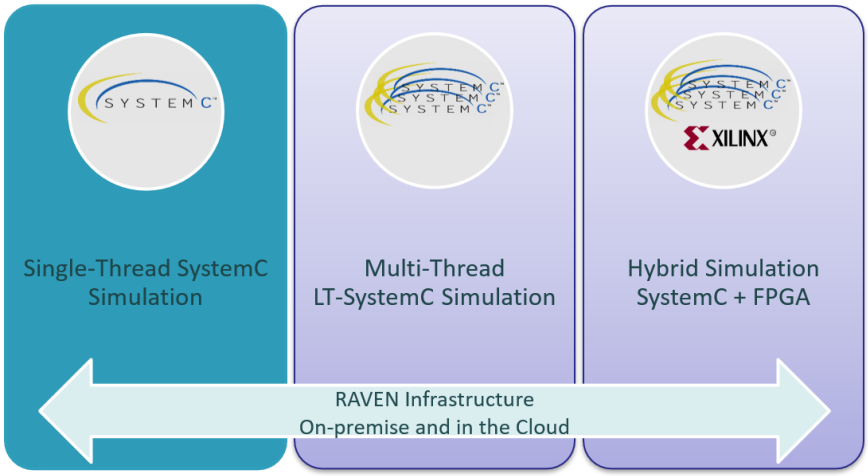

Figure 1: Overview of simulation modes

RAVEN supports different simulation options (Figure 1) to enable more efficient and accurate hardware/software development. These include:

- Single-threaded SystemC simulation

  The adjustments made to the SystemC kernel for parallel SystemC and Hybrid Simulation were carefully designed to ensure backward compatibility, allowing conventional single-threaded SystemC simulations to remain fully supported.

- Multi-threaded parallel simulation

  RAVEN provides an infrastructure that enables multi-threaded parallel SystemC simulation for loosely timed partitions.

- Hybrid simulation of SystemC and FPGA

  The highlight of RAVEN is its hybrid simulation capability, which combines the advantages of both SystemC simulation and FPGA prototyping. This approach achieves the best acceleration and significantly reduces simulation time, but provides only limited debugging options. To circumvent this, RAVEN offers the possibility to generate and simulate the FPGA RTL in a hybrid manner with Verilator instead of the FPGA.

RAVEN offers the flexibility to run parallel and hybrid simulations either locally on-premise or in the cloud. Cloud simulation provides scalable and cost-efficient access to a wide range of computing resources and alleviates the need for upfront hardware investments and maintenance. A graphical user interface assists the user with the setup, configuration and execution of the simulation in the Cloud.

III. IMPLEMENTATION

As described in Section II, the parallelization in RAVEN is based on splitting a complex SystemC design into multiple partitions. These partitions are simulated using separate threads or on an FPGA. For best results, tightly coupled components are located in the same partition in order to minimize the communication across partition boundaries. RAVEN provides an infrastructure that connects all partitions and enables data exchange and synchronization between them.

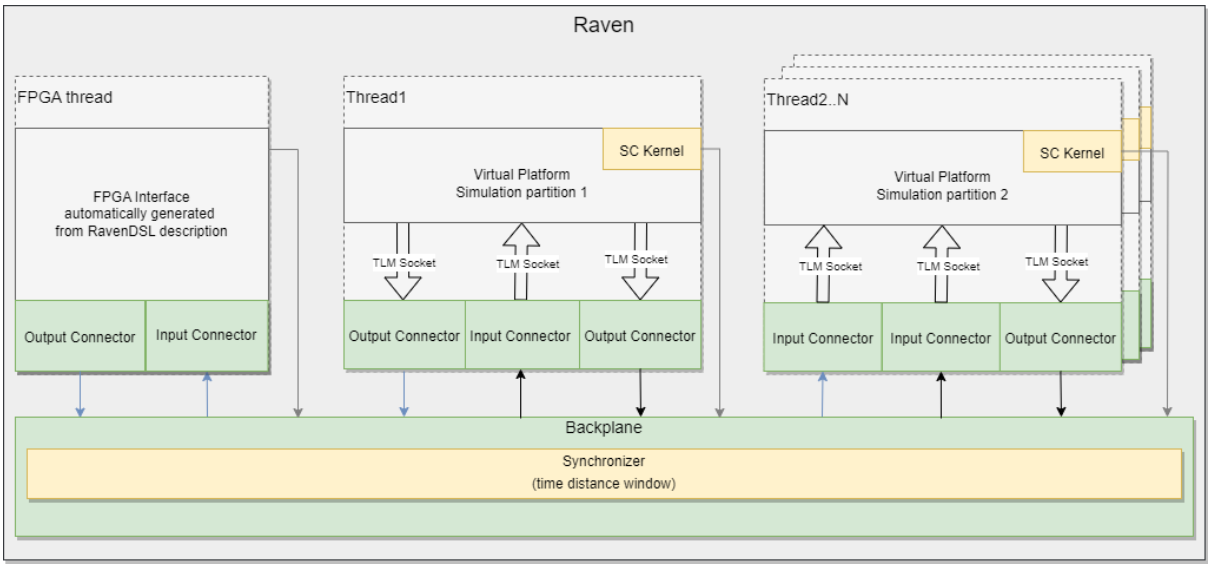

Figure 2: RAVEN infrastructure

The RAVEN infrastructure (Figure 2) is composed of several key components:

- Backplane - a central component of the RAVEN infrastructure. It facilitates the seamless coordination and communication among different simulation partitions. It functions as a central hub, collecting and managing crucial information from all partitions. Essentially, the backplane acts as a communication backbone that supports the parallel execution of simulations.
- Connectors - are used to connect individual partitions of the multi-threaded simulation to each other, enabling seamless data exchange between them. All inter-partition connections are based on the same principle: the inputs and outputs of the DUT are attached to the RAVEN connectors, which establish essential communication channels between different partitions, ensuring secure and thread-safe data exchange.
- Synchronizer - controls the simulation time across all partitions and prevents individual simulation partitions from diverging in time. For more in-depth details, refer to the Section IV.
- RavenDSL - a domain-specific language and a tool-flow used to automatically create the hardware/software components required for an FPGA-based partition. The goal is to make the task of simulating a part of the design on an FPGA as simple as possible. For more comprehensive details, please consult the Section V.

IV. SYNCHRONIZATION

In order to ensure correct communication between several parallel-running components, the SystemC kernels have to be synchronized. The Accellera SystemC kernel does not provide this functionality. To enable parallel execution and synchronization between multiple SystemC partitions, changes had to be made to the Accellera reference simulator, with a strong emphasis on maintaining backward compatibility with the SystemC standard. This was ensured by using the official regression suite from Accellera [2]. All changes are automatically verified against this regression suite.

The initial step was to ensure thread safety in the SystemC library. A comprehensive analysis of the SystemC library for potential data races was done using the tool ThreadSanitizer [3] and the Static Analyzer, a helper tool which uses Clang AST to find variables with static storage duration. After eliminating data races in SystemC, the focus shifted towards synchronization.

In RAVEN each partition operates with its own local simulation time. However, to prevent excessive divergence between threads, synchronization is crucial. The Synchronizer plays a key role in this process, ensuring that individual partitions remain within a predefined time interval of each other. When a partition seeks to update its simulation time, it must first obtain approval from the Synchronizer, which holds track of the current time for all partitions and regulates the advancement of each partition within the defined limits. This ensures that no partition deviates excessively ahead or behind others, maintaining a controlled level of synchronization.

The allowable synchronization time distance is a user parameter and remains constant for the entire simulation. The value of the time distance shall be defined at the beginning of the simulation. A wider distance increases the simulation speed, while a small distance achieves more accurate timing behavior of the simulation. This mechanism allows the user to decouple the time between components in a controlled way.

V. HARDWARE INTEGRATION

When integrating an FPGA-based partition with a RAVEN simulation, additional hardware and software components are required: A hardware wrapper receives transaction messages from other partitions and applies them to the RTL as signal transitions. In the other direction the wrapper also analyzes the signal transitions of the RTL and creates transaction messages for other partitions. The software part acts as a bridge between the RAVEN infrastructure on one side and the hardware on the FPGA on the other side. It forwards transaction messages between the hardware wrapper on the FPGA and the RAVEN Backplane. It also starts and stops the hardware IP on the FPGA based on the information received by the Synchronizer.

Creating these hardware and software parts by hand would be a time-intensive and error-prone task. Thus, RAVEN aims to automate this process as much as possible.

This is the motivation behind the RavenDSL, which is used to describe the interfaces of the RTL to be simulated. In addition to the interfaces and ports, it also describes how transaction messages (in the form of TLM packages) are translated to signal transitions for the hardware IP and vice versa.

This information is then used by the accompanying tool-flow to automatically create all required hardware and software parts. The tool-flow then uses the TaPaSCo framework [4] to create a bitstream for the FPGA. TaPaSCo provides us an abstraction layer, enabling the use of the same generated hardware and software with different FPGAs (on-premise and in the cloud). A set of CMake scripts helps with integrating the generated software with the rest of the RAVEN simulation.

The following section provides a brief illustration of the RavenDSL through a small example.

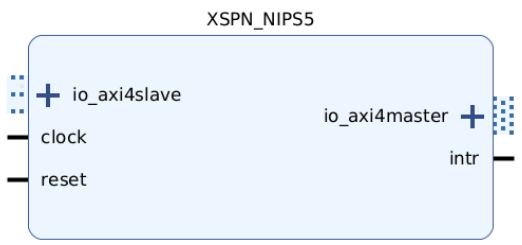

Figure 3: Hardware IP

```
1  dut XSPN_NIPS5;
2      clocks {
3          Clock#(100 MHz) clock;
4      }
5      resets {
6          ResetAgent#()(ACTIVE_HIGH) reset;
7      }
8      AXI4LiteSlaveAgent#(32,32) io_axi4slave;
9      AXI4MasterAgent#(32,512,1,0) io_axi4master;
10     InterruptAgent#() intr;
11 }
```

Figure 4: RavenDSL description

The Figure 3 shows a sample IP block to be simulated using the FPGA, while the Figure 4 shows the corresponding description in the RavenDSL.

The first part (lines 2-4) defines the clocks used by the RTL, in this case a single 100MHZ clock. The second part (lines 5-7) accordingly define the reset inputs, here again a single, active-high reset signal.

The rest of the description (lines 8-10) defines all other protocols. The RavenDSL uses the term "Agent" from the UVM [5] terminology. An Agent internally describes the ports of the interface, but also the behavior: how transaction messages are applied to the RTL via signal transitions and vice versa.

The sample IP block e.g., uses an Agent for AXI4-Lite (line 8) and for AXI4 (line 9).

This example only uses existing Agents, the definitions of those Agents is not shown. These existing Agents are part of a library provided with RAVEN and can easily be re-used. The RavenDSL also allows to define new Agents using state machines to support new protocols. This of course also allows to use non-standard protocols with RAVEN.

VI. Related Work

The FPGA-accelerated full-system hardware simulation platform FireSim [6] focuses on accelerating hardware verification using FPGA, with a strong focus on Chisel-based designs. Its primary goal is to provide a scalable and efficient solution for simulating complex hardware systems on FPGAs on-premise and in the cloud. In contrast, RAVEN enables hybrid simulation, allowing a design to be executed partially on FPGA and partially in a VP.

Commercially available emulation platforms such as Mentor Veloce, Synopsys ZeBu [7] and Cadence Palladium [8] require specialised hardware and are therefore extremely expensive. RAVEN Hybrid simulation, on the other hand, offers a low-cost simulation option that utilizes standard Xilinx FPGA boards and can be deployed in the cloud, using commercial cloud FPGA offerings such as AWS F1.

The SCE-MI [9] standard specifies all the components needed for communication between the software side and the hardware side. However, developers have to implement the necessary components themselves. With RavenDSL, developers can describe hardware interfaces in a simple and abstract DSL, and the RavenDSL compiler automatically generates HW drivers and monitors that stimulate hardware design interfaces and observe hardware output data, respectively. RavenDSL simplifies the integration of a HW part into the RAVEN simulation. In contrast to existing mechanisms for incorporating FPGA emulation into SystemC-based simulations, such as SCE-MI, the level of abstraction in RAVEN is significantly increased, while the integration effort has been truly reduced. The interface description in RavenDSL is not only shorter compared to BlueSpec or Verilog, but it is also easier to understand for software engineers, making it a more accessible option for a wider range of developers.

VII. Validation and Performance Analysis Results

The RAVEN infrastructure has been validated using various designs consisting of a RISC-V core subsystem and one or more HW ML accelerators. A family of sum-product-network-based ML hardware accelerators called XSPN [10] developed at the TU Darmstadt has been chosen for this evaluation, as the IP blocks have identical interfaces but can easily be scaled in terms of complexity and size. This family minimizes integration effort and allows for varying computational effort, which is useful for performance analysis of the RAVEN infrastructure.

We use the Verilator tool to integrate the HW accelerator in single- and multi-threaded simulations. In hybrid simulation, the HW accelerator is simulated on an FPGA.

The performance analysis of the RAVEN infrastructure involves several design variants, summarized below:

- Design A: This design consists of a RISC-V core subsystem in one partition and one ML HW accelerator in another partition, in single-threaded, multi-threaded and hybrid variants. During the execution of calculation tasks by the ML HW accelerator, the core subsystem remains idle, resulting in an unbalanced distribution of workloads between the design partitions.

- Design B: This design includes a core subsystem and two HW accelerators running in parallel, in both single-threaded and multi-threaded simulation modes. The design has three partitions, which are better balanced because the two accelerators are equally complex and compute the equal amount of data.

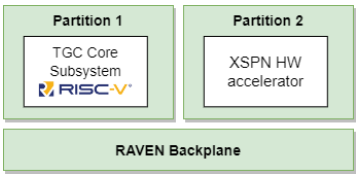

Figure 5: RAVEN Validation Design A

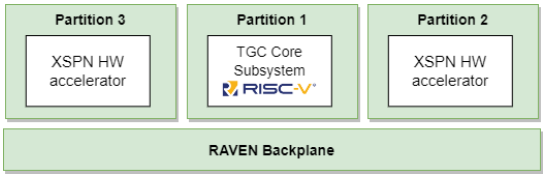

Figure 6: RAVEN Validation Design B

For this evaluation, two configurations of the XSPN accelerators are used: a smaller configuration called NIPS5 and a bigger one named NIPS80. Table 1 shows the resource usage of the accelerator in the column DUT. The column RAVEN of Table 1 shows the resource usage of the generated RAVEN infrastructure around the accelerators, as introduced in Section V. We use TaPaSCo [4] to create the FPGA-specific wrapper, e.g. containing host and memory connections, and building the bitstream. The third column of Table 1 consolidates the resource consumption, combining the accelerator, the RAVEN infrastructure and the FPGA-specific wrapper created by TaPaSCo.

	NIPS5			NIPS80			Available
	DUT	RAVEN	Total	DUT	RAVEN	Total	
LUT	17k	25k	167k (12.8%)	99k	25k	250k (19.2%)	1304k
Register	7.7k	24k	214k (8.19%)	159k	24k	361k (13.9%)	2607k
CLB	3.3k	6.0k	36k (21.8%)	20k	4.3k	52k (31.8%)	163k
BRAM	11	103	222 (11.0%)	71	103	282 (14.0%)	2016
DSP	23	0	26 (0.25%)	736	0	739 (8.16%)	9024

Table 1: FPGA Implementation results. Created using Vivado 2022.1 for the AMD Alveo U280.

The performance evaluation (Figure 7) shows that for Design A, multi-threaded SystemC simulation does not provide significant speed-up due to unbalanced partitions. For particularly small systems, hybrid simulation is not recommended because the overhead due to serialisation, de-serialisation of data and data transfer becomes a bottleneck.

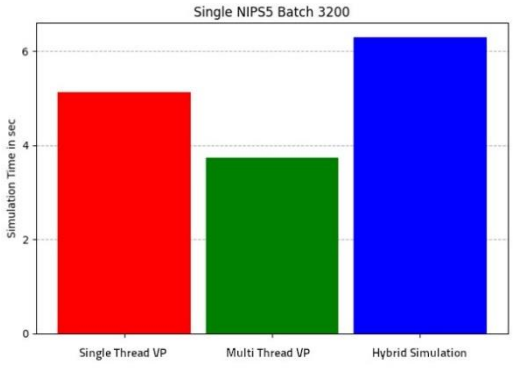

Figure 7: Design A: Core Subsystem, 1 XSPN NIPS5

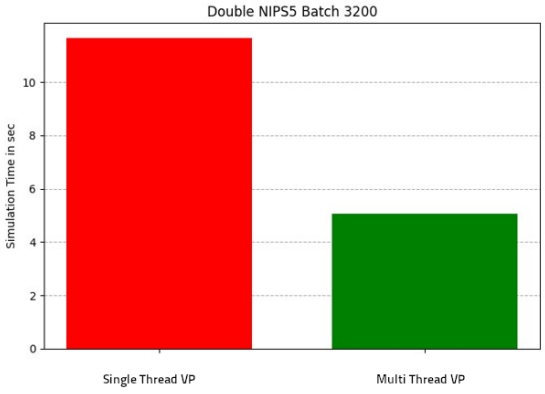

Figure 8: Design B: Core Subsystem, 2 XSPNs NIPS5

With a more complex HW accelerator with about four times more LUTs and eight times more registers, the speed gain with RAVEN becomes very clear (Figure 9). Note that the time of the hybrid simulation hardly changes,

while the conventional SystemC simulation has become slower by about a factor of 7. This is explained by the fact that a larger accelerator uses more resources on the FPGA, but it requires the same time to execute (assuming the same clock frequencies are used). This differs from simulating a more complex accelerator using SystemC, which takes more time compared to a simple accelerator.

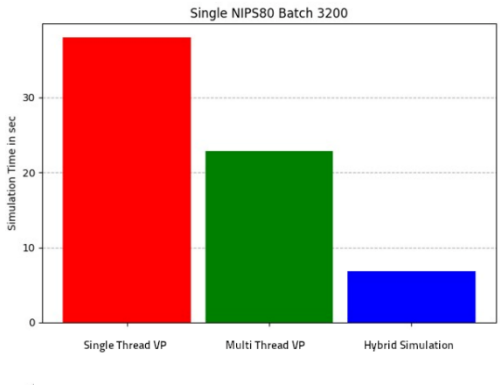

Figure 9: Design A: Core Subsystem, 1 XSPN NIPS80

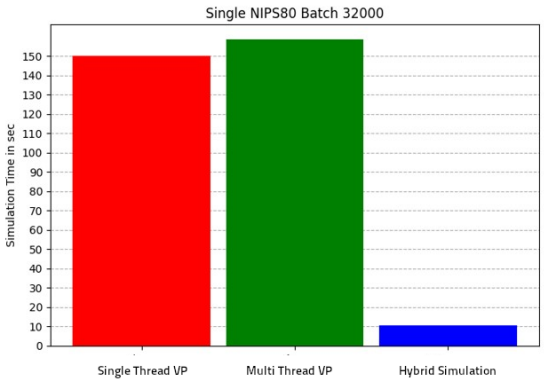

Figure 10: Design B: Core Subsystem, 2 XSPNs NIPS80

If we further increase the computing load of the HW accelerator (Figure 10), the hybrid simulation can really shine. In this case, the multi-threaded simulation with a verilated XSPN is no longer worthwhile, because with so much data being copied back and forth between the threads, the MT simulation becomes even slower than ST.

VIII. Conclusion

The performance evaluation showed that hybrid simulation may not be the best option for small designs. However, the effectiveness of hybrid simulation improves as the computational effort increases. For complex HW accelerators, RAVEN provides significant speed gains, and hybrid simulation time remains relatively constant, while conventional SystemC simulation slows down significantly.

The results of this performance analysis confirm that different types of RAVEN simulation are useful depending on the design layout. Hybrid simulation can be very effective in certain cases, especially for complex and computationally expensive design.

IX. References

[1] Accellera Systems Initiative, "Core SystemC Language 2.3.4," Accellera Systems Initiative, [Online]. Available: https://www.accellera.org/downloads/standards/systemc. [Accessed 07 08 2023].

[2] Accellera Systems Initiative, "SystemC Regression Test Suite 2.3.3," Accellera Systems Initiative, [Online]. Available: https://www.accellera.org/downloads/standards/systemc. [Accessed 07 08 2023].

[3] ThreadSanitizer , "ThreadSanitizerCppManual," Google Inc., [Online]. Available: https://github.com/google/sanitizers/wiki/ThreadSanitizerCppManual. [Accessed 07 08 2023].

[4] Heinz, Hofmann, Korinth, Sommer, Weber and Koch, "The TaPaSCo Open-Source Toolflow," *Journal of Signal Processing Systems*, 2021.

[5] Accellera Systems Initiative, "Standard Universal Verification Methodology," [Online]. Available: https://www.accellera.org/downloads/standards/uvm. [Accessed 16 08 2023].

[6] FireSim, "Fast and Effortless FPGA-accelerated Hardware Simulation with On-Prem and Cloud Flexibility," 07 08 2023. [Online]. Available: https://fires.im/. [Accessed 07 08 2023].

[7] Synopsys Inc., "ZeBu Server 5," [Online]. Available: https://www.synopsys.com/content/dam/synopsys/verification/technical-papers/zebu-server5-spec-mar2023.pdf. [Accessed 07 08 2023].

[8] Cadence Design Systems Inc., "Protium S1 FPGA-Based Prototyping Platform," [Online]. Available: https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/tools/system-design-verification/protium-s1-fpga-based-prototyping-platform-ds.pdf. [Accessed 07 08 2023].

[9] Accellera Systems Initiative, "Standard Co-Emulation Modeling Interface (SCE-MI)," 2016. [Online]. Available: https://www.accellera.org/images/downloads/standards/sce-mi/SCE-MI_v24-Nov2016.pdf. [Accessed 07 08 2023].

[10] L. Sommer, L. Weber, M. Kumm and A. Koch, "Comparison of Arithmetic Number Formats for Inference in Sum-Product Networks on FPGAs," 2020. [Online]. Available: https://www.esa.informatik.tu-darmstadt.de/assets/publications/materials/2020/2020_FCCM_LS.pdf.