Distributed and Reconfigurable Communication System for Open Source Real-Time Simulations of Power Systems

Niklas Eiling, Manuel Pitz, and Antonello Monti, *Senior Member, IEEE*

***Abstract*—Hardware for real-time simulations of power systems is often highly specialized, proprietary, inflexible, and expensive. A consequence is that many systems have low interoperability, are not conducive to innovation, are not extensible, and are inaccessible to many researchers and small companies. While there have been efforts to build open source real-time simulation software, there is a lack of an open implementation of the infrastructure that facilitates interconnecting the hardware and software components of a real-time simulation. In this article, we present an open source implementation of a communication system that empowers real-time simulation software to be used in cosimulations and hardware-in-the-loop simulations that require simulation equipment, measurement devices, and power system hardware to be interconnected using real-time capable communication channels. The system is designed to enable hassle-free reconfiguration without the need for physical rewiring, even when simulation components are distributed across different locations. We show the applicability by connecting the system to the open source simulation software DPsim and performing a latency benchmark, a cosimulation, and a power hardware-in-the-loop simulation. Our results show that our system is capable of achieving a time step of 5 μs and thus has a performance similar to that of commercial real-time simulators. Our communication system is based on established standards and is interoperable with many protocols and data formats. Other researchers can therefore use our system as a basis for interfacing arbitrary components to real-time simulations.**

***Index Terms*—Cosimulation, Field-programmable gate array (FPGA), hardware-in-the-loop, interoperability, power systems, real-time simulation.**

I. Introduction

THIS article addresses the need for open, reconfigurable, and low-latency communication systems to support real-time simulations in power system research and validation. As power systems undergo unprecedented transformation with increasing renewable energy penetration, distributed generation, and smart grid technologies, real-time simulation has become indispensable for ensuring system reliability, stability, and safety [1], [2], [3]. The complexity of modern power system protection schemes, the need for validating advanced control algorithms, and the criticality of hardware-in-the-loop testing for power electronic converters all demand simulation platforms that can accurately reproduce the microsecond-level timing behavior of actual power system events.

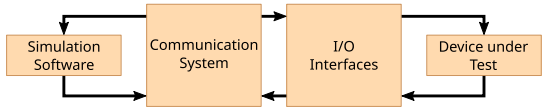

Fig. 1. Conceptual overview of the components of a hardware-in-the-loop real-time simulation.

Real-time simulators are extensively used in power system applications including cosimulation for large-scale system analysis, controller hardware-in-the-loop (CHIL) testing for protection relays and smart grid devices, and power hardware-in-the-loop (PHIL) experiments for validating power electronic equipment under realistic grid conditions. In all cases, a critical component is the *communication system* that transmits signals between simulation software, measurement equipment, and physical devices. The timing precision of the communication system directly impacts the validity of protection studies, the accuracy of converter control validation, and the stability of PHIL amplifier interfaces. Fig. 1 shows an overview of the components involved in a real-time simulation. Unlike offline simulations, real-time simulations must ensure that simulation time remains synchronized with wall-clock time, requiring all computations and communication to complete within a fixed time step of 50 μs or lower [1]. Communication latency therefore becomes a critical bottleneck that can compromise simulation fidelity and stability.

Existing solutions for communication systems for real-time simulations are frequently proprietary, inflexible, and costly [4]. Commercial real-time simulators like OPAL-RT [5], Typhoon HIL [6], and RTDS [3] lock users into vendor-specific ecosystems with limited customization capabilities and implementation transparency. These systems often require manual reconfiguration involving physical rewiring—a time-consuming, error-prone, and potentially unsafe process that hinders rapid experimentation and interoperability. While open source real-time simulation software such as DPsim [7] exists, there remains a critical gap: the lack of open infrastructure to interconnect simulation components with the low latency and flexibility required for time-critical scenarios. This communication infrastructure gap significantly limits the practical deployment of open source real-time simulators. Existing research implementations face

Received 7 January 2025; revised 6 May 2025, 31 July 2025, and 17 December 2025; accepted 12 January 2026. Date of publication 29 January 2026; date of current version 22 April 2026. This work was supported in part by the European Union under the Horizon Europe Research and Innovation Programme under Grant 101173007 and in part by the German Federal Ministry of Education and Research (BMBF) under Grant 03HY128B. *(Corresponding author: Niklas Eiling.)*

Niklas Eiling and Manuel Pitz are with RWTH Aachen University, 52062 Aachen, Germany (e-mail: niklas.eiling@rwth-aachen.de).

Antonello Monti is with RWTH Aachen University, 52062 Aachen, Germany, and also with the Fraunhofer Institute for Applied Information Technology FIT, 53757 Sankt Augustin, Germany.

Digital Object Identifier 10.1109/JSYST.2026.3655017

© 2026 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

TABLE I
COMPARISON OF REAL-TIME SIMULATION APPROACHES

Approach	Min. Time Step	Architecture	Model Flexibility	Communication	Open Source
VTB-RT [4]	$50\,\mu s$	CPU + Analog I/O	High (Software)	Analog	Yes
Difronzo et al. [9]	$70\,ns$	Multi-FPGA	Low (FPGA)	Digital (Parallel)	No
Adler et al. [13]	$1\,\mu s$	FPGA + DSP	Medium (DSP Code)	Digital, Analog	No
Chiandone et al. [14]	$10\,\mu s$	CPU-based	High (Software)	UDP (10ms rate)	No
OPAL-RT OP5700	$5\,\mu s^{\dagger}$	CPU + FPGA	Medium (Proprietary)	Digital, Analog	No
RTDS NovaCor	$3.9\,\mu s^{\dagger}$	Custom CPU + FPGA	Medium (Proprietary)	Digital, Analog	No
This Work	$\mathbf{5\,\mu s}^{\dagger}$	**CPU + FPGA**	**High (Software)**	**Digital, Analog**	**Yes**

Notes: Model Flexibility indicates ease of modifying simulation models and algorithms; Communication shows types of real-time interfaces supported; † Values measured in our laboratory; unmarked entries are self-reported minimum time steps.

fundamental limitations: CPU-based solutions like VTB-RT [4] struggle to meet microsecond-level timing requirements, while field-programmable gate array (FPGA)-only approaches [8], [9] sacrifice the flexibility and maintainability essential for research.

Our approach resolves this by leveraging FPGAs specifically for communication infrastructure while maintaining CPU-based simulation for model execution. FPGAs are particularly well-suited for real-time communication due to their deterministic timing, spatial parallelism, and ability to achieve consistent low-latency performance [10]. By dedicating FPGAs to communication rather than simulation, we preserve the flexibility and maintainability of CPU-based solvers while achieving the performance necessary for demanding real-time applications. This architectural decision enables rapid model development and testing without compromising on communication performance. To address the gap in communication infrastructure for real-time simulation, we present an open source communication system that enables low-latency, distributed, and reconfigurable interconnections specifically designed for demanding power system applications. All hardware designs, FPGA logic, and software components are openly available.[1,2,3] enabling precise latency modeling and fostering innovation in the real-time simulation community. Our system builds on the open source VILLAS-framework [11], extending it with a user-space PMD framework for low-latency communication and the Aurora protocol [12] for FPGA-to-FPGA connections. Additionally, we developed specialized hardware including DINO (for analog signal interfacing) and MIOB (for distributed digital connections) to address typical laboratory requirements.

The main contributions of this article are as follows.

1) *Open communication infrastructure:* An FPGA-based system achieving $5\,\mu s$ time steps with full implementation transparency, democratizing access to advanced HIL testing capabilities for smaller research institutions.
2) *Software-defined laboratory flexibility:* A circuit-switched FPGA design enabling dynamic reconfiguration without physical rewiring, essential for rapid transitions between different testing scenarios.
3) *Modular power system interfaces:* Hardware and software components (DINO, MIOB) addressing diverse connectivity requirements for power system devices in distributed HIL environments.
4) *Validated real-world performance:* Thorough evaluation demonstrating competitive latency with commercial simulators through power system-specific experiments including IEEE WSCC 9-bus cosimulation and distributed PHIL testing.

II. RELATED WORK

Real-time power system simulators require communication interfaces with stringent latency and timing requirements to enable effective HIL testing, cosimulation, and protection scheme validation. Several research groups have developed real-time simulation platforms with varying approaches to the communication challenge. Table I provides a comparison of approaches for real-time power system simulation, highlighting the fundamental tradeoffs between performance, flexibility, and implementation complexity that existing solutions face.

VTB-RT [4] represents an early effort in open real-time simulation, offering an analog interface based on off-the-shelf acquisition hardware and achieving time steps as low as $50\,\mu s$ for HIL simulations involving boost converters. However, VTB-RT's reliance on RTAI [15] requires kernel patching and complex application development, leading to maintenance and portability challenges [16]. Chiandone et al. [14] developed a system that communicates via UDP with controllers to test power management systems in dc systems, achieving $10\,\mu s$ time steps with communication occurring only every 10 ms. While this demonstrates reasonable performance, the limited communication frequency and UDP-based approach cannot meet the stringent timing requirements of modern power system applications. Our approach advances beyond these limitations by leveraging the now mature `PREEMPT_RT` features of the Linux kernel, eliminating the need for kernel modifications while achieving good real-time performance.

The pursuit of ultralow latency has led to using FPGAs for real-time simulation [9] and hybrid FPGA-DSP approaches [13]. Simulations on these specialized hardware solutions achieve lower time steps than CPU-based systems. Difronzo et al. [9] demonstrated 70 ns time steps. FPGAs have also been shown to offer higher energy efficiency compared to CPU and GPU architectures in specific applications [17], [18], [19]. However, the performance and energy-efficiency of FPGA-centric approaches comes at the cost of flexibility and development complexity. As shown in Table I, pure FPGA implementations achieve the lowest latencies but require complete hardware redevelopment for model changes, severely limiting general applicability and hindering the rapid

[1] [Online]. Available: https://github.com/VILLASframework/node
[2] [Online]. Available: https://github.com/RWTH-ACS/miob
[3] [Online]. Available: https://github.com/RWTH-ACS/dino

experimentation. Our work resolves this performance-flexibility tradeoff by strategically dedicating FPGAs to communication infrastructure while preserving CPU-based simulation for model execution.

Commercial real-time simulators, e.g., those of OPAL-RT [5], Typhoon HIL [6], and RTDS [3], represent the current state-of-the-art in terms of performance and industrial adoption. OPAL-RT and Typhoon HIL utilize Linux-based platforms with off-the-shelf hardware, while RTDS employs custom hardware and operating systems optimized specifically for real-time simulation. All three vendors provide FPGA capabilities for fast communication and simulation of highly dynamic system components [1]. However, the proprietary architectures of these commercial solutions limits customization, creates vendor lock-in, restricts interoperability, and the high costs create barriers for smaller research institutions and educational facilities. Furthermore, the closed nature of these systems prevents users from understanding and compensating for communication latencies, which is essential for stable PHIL operations and accurate protection relay coordination studies. Our fully open approach addresses this limitation while delivering competitive performance, enabling precise latency modeling and fostering innovation in the real-time simulation community [20].

Recent research has increasingly focused on the critical role of communication infrastructure in real-time simulation systems. Dong et al. [21] demonstrated the importance of low-latency interconnection by simulating power plants with submicrosecond time steps using OPAL-RT simulators connected via Aurora channels, exchanging hundreds of signals with minimal delay. Similarly, Broadmeadow et al. [22] developed FPGA-based modular control systems for PHIL amplifiers connected to RTDS simulators, highlighting the need for optimized communication to achieve predictable latencies. Barbierato et al. [23] specifically studied communication delays in cosimulations between RTDS and OPAL-RT simulators using Aurora channels, finding delays of two to five time steps while maintaining acceptable performance compared to monolithic simulations. This work underscores the critical importance of minimizing communication delays for accurate and stable cosimulation results. The Energy Lab 2.0 [24] exemplifies a large-scale real-time simulation lab, featuring multiple PHIL setups interconnected via fiber using OPAL-RT OP4510 systems across large areas. While achieving high flexibility and performance, this approach requires substantial investment and creates a dependency on a single vendor. Our work addresses the identified limitations across all categories by providing an open-source communication infrastructure that combines the performance of commercial solutions with the transparency and flexibility essential for research advancement.

III. Communication System for Real-Time Simulations

The development of real-time simulators requires dealing not only with solvers computing the time steps of a simulation model, but also with communication systems interconnecting different hardware components [2]. This includes connecting different components that compute the simulation such as CPUs and FPGAs, connecting different simulators to build cosimulations, and connecting simulators to power system devices.

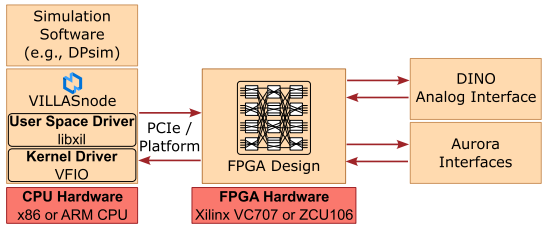

Fig. 2. Overview of hardware, FPGA, and software design.

In this article, we present a novel communication system consisting of software, hardware, and FPGA logic that implements a flexible, adaptable communication system for performing real-time simulations. Fig. 2 shows an overview of our implementation. It can be coupled with simulation software to implement a real-time simulation as shown in the figure, and is capable of running on traditional x86 server hardware and a PCIe FPGA or a more integrated MPSoC platform. When designing our system, we considered the following communication paths to satisfy the needs of even complex simulation scenarios.

1) Communication between simulation software and peripherals, e.g., network interfaces or FPGAs.
2) Communication with devices using digital interfaces, e.g., controllers.
3) Communication with devices using analog interfaces, e.g., simple PHIL amplifiers.
4) Communication between simulators to achieve cosimulation.

We developed our communication system with the goal of interconnecting a wide variety of hardware and software components in a flexible, reconfigurable way that acknowledges the changing requirements of researchers. Fig. 3 shows an example of a deployment of our communication setup and highlights the possibilities such a system can offer. In this deployment, our software and FPGA design are used to connect the existing real time simulation software DPsim to a network of other real-time components, enabling the exchange of simulation signals. The FPGA acts as a reconfigurable switch, allowing realization of arbitrary communication patterns. The same software and FPGA design are used to build MIOB, a more integrated, less powerful but field-deployable implementation. DINO, our novel, isolated, analog interface, connects any device to analog devices, for example, power amplifiers, or relays (shown in the picture). Using the existing VILLASnode software, we enable interfacing a wide variety of digital protocols, some of which are shown in the figure.

The development of such a system requires dealing with a multitude of challenges. Most importantly, the real-time nature of the simulation requires that the simulated time never lags the wall clock time. This makes it necessary for computations and communication to occur predictably and with low latency. In this section, we describe how we address this challenge when building the hardware platform, the FPGA design, and the software interface of our communication system.

A. Hardware Platform

Although desktop computers are widely used for offline simulations of power systems, the constraints on predictability and

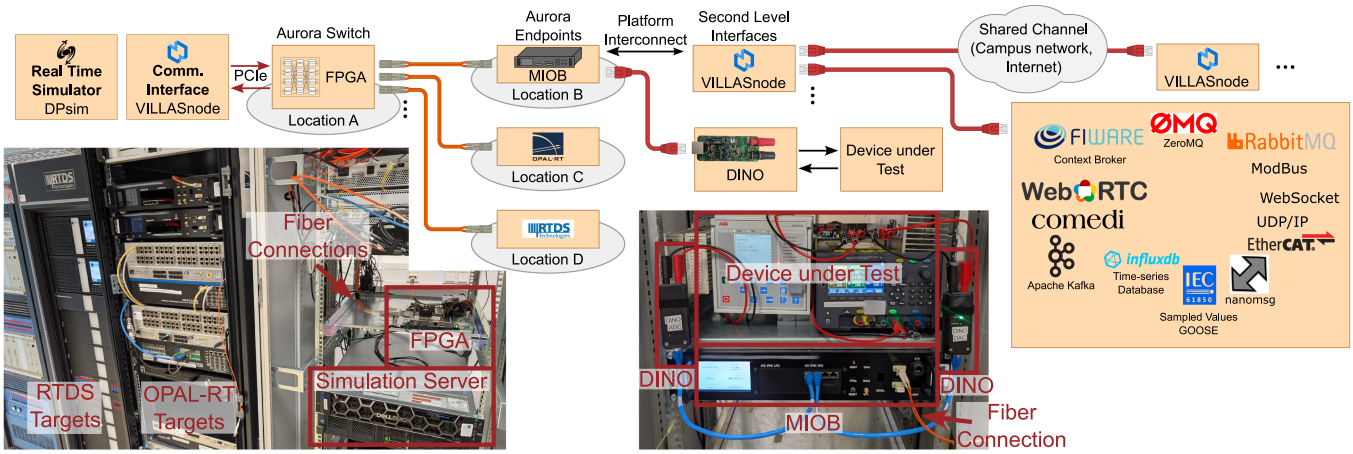

Fig. 3. Reconfigurable real-time communication setup, as deployed in the real-time laboratory of the Institute for Automation of Complex Power Systems at RWTH Aachen University.

latency make them unsuitable for real-time simulations. On the one hand, their computing model favors interactivity for many applications executing in parallel over the predictability of a single application. On the other hand, they require additional peripheral devices to communicate with external components. For these reasons, most commercial real-time simulators use hardware platforms specifically designed for their use in real-time simulations [5]. Although there are both examples of simulators using a CPU-only and an FPGA-only platform, more commonly simulators use CPUs for computations and an FPGA for communication. Typically, the computation of highly dynamic parts of a simulation can also be offloaded to the FPGA. The CPU has the benefit of requiring low development effort, adaptability to a wide variety of problems, and lower costs. Using a real-time capable operating system on a dedicated system that runs only the simulation allows guaranteeing that simulations can run in real-time on a CPU-based computing system [7].

When coupling the simulation with external devices or other simulators, the real-time capability of the communication system has to be considered as well. Communication in a CPU-based computing system occurs by software interacting with the communication hardware via operating system interfaces. The operating system uses a device-specific driver to interact with the hardware. This typically involves communication with the hardware via system internal interconnects, such as PCIe. For common network interface cards, hardware, drivers, and interfaces are readily available. This makes basing a communication system for real-time simulation on them cost-effective and straightforward. Using one of the ubiquitous network protocols of the IP family allows interconnecting via existing wide area networks, even across continents. Such geographically distributed real-time simulations have previously been successfully implemented [25]. However, in such cases, the communication channel is shared among many users, making the performance of the channel unreliable and only allowing statistical latency and jitter guarantees. The latencies in such setups are typically in the range of milliseconds.

When the simulation scenario requires exchanging real-time data that is highly dynamic, jitter-sensitive, or safety-critical, statistical latency and jitter guarantees are not sufficient. Some examples where a real-time simulation requires stricter guarantees for the communication are listed below.

1) When the interaction between simulators or with real hardware includes high-frequency components, the simulation becomes highly dynamic and requires small communication latencies, e.g., when studying hardware that includes switching converters [2], or hybrid ac/dc grids.
2) The simulation becomes jitter-sensitive, when the simulation is not tolerant to changing delays across communication channels, e.g., because the delayed arrival of messages can lead to time step overruns or because the used interface algorithms are not tolerant to larger delays.
3) When a communication failure may cause harm or damage to personnel or equipment, the simulation becomes safety critical, e.g., a PHIL amplifier that stops receiving set-points, thus producing a dc output thereby leading to damage to connected hardware or the amplifier itself.

In such scenarios, hard real-time performance is required, i.e., latency and jitter constraints need to be guaranteed in all cases [16]. This confidence can be achieved by using real-time capable protocols, such as CAN, EtherCAT, or TSN, which can guarantee certain performance while still allowing a shared channel, or by not sharing channels at all. Although under these considerations, communication hardware, such as a network interface card, is still an option for some real-time simulation applications, the fact that they are usually designed for bandwidth rather than latency and predictability leads to unacceptable delays when using them for real-time simulations with time steps in the microsecond range.

FPGAs are better suited for real-time communication interfaces, offering consistent performance, spatial parallelism, and optimizable designs for specific latency and bandwidth goals [10]. Unlike other hardware, we can optimize an FPGA design to meet specific internal latency and bandwidth goals. However, designing a system using FPGAs is more complex than traditional software development or using off-the-shelf communication hardware.

1) FPGA- and CPU-Based Simulation Platform: Because only FPGAs allow for meeting performance comparable to commercial solutions, we build our communication system using an FPGA connected to a CPU. We use a Xilinx VC707 FPGA board connected to an off-the-shelf Dell 740xd server. The server is

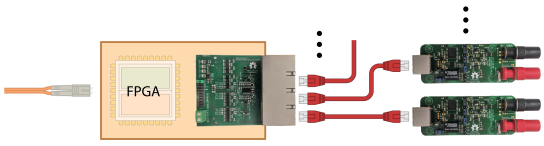

Fig. 4. Overview of the DINO hardware.

equipped with two Intel Xeon Gold 6132 CPUs and 256 GB of memory. The standardized PCIe interface allows connecting the FPGA to any compatible computing system and upgrading to more powerful servers if needed. Likewise, we can extend the connectivity of the FPGA board using interfaces commonly used with FPGAs: Two VITA 57 compliant FMC interfaces and several PMOD interfaces. Using only one FMC interface, we support up to eight Aurora channels for the connection to compatible devices.

2) MIOB. A Self-Contained FPGA Platform for Interconnecting Real-Time Simulation Components: Although using an FPGA connected to an off-the-shelf server via PCIe offers high flexibility, our approach also works on more specialized hardware, e.g., on the Xilinx MPSoC architecture, which integrates an ARM CPU and FPGA on a single chip. This platform enables building self-contained platforms for interconnecting and running real-time simulations in small form factors. We use a Xilinx ZCU106 FPGA board in a 1U rack-mountable case with peripheral hardware, FPGA design for communication routing, and custom Linux distribution to form the Mobile Input Output Box (MIOB). MIOB enables hassle-free, fast real-time interconnection of devices and simulators for distributed HIL and cosimulation setups. It features two fiber ports for Aurora compatible devices, Ethernet, and extensibility via PMOD and FMC interfaces for GPS synchronization, eight additional Aurora channels, and the analog interfaces. To enable remote management when an MIOB is installed in a location that is not easily accessible, it includes a Raspberry Pi that can control, program, and reset the MPSoC. The Linux distribution for MIOB is based on the Xilinx Yocto distribution, modified so that the Linux kernel is aware of the peripherals programmed in the FPGA design and extended by the software for interfacing with our communication system and dependencies.

3) DINO. Isolated Analog In- and Output: Converting between digital and analog signals is often necessary when connecting devices to real-time simulations, as most devices interface with the grid by measuring or injecting voltages and currents. Although CHIL using only digital interfaces and PHIL using a digitally interfaced grid emulator are possible, interfacing with devices using analog signals remains widespread in real-time simulations.

To connect components requiring analog signals, we developed digital input and output (DINO), a low-latency, modular, and isolated analog interface that connects to an FPGA using an FMC connection. Fig. 4 shows DINO analog-to-digital and digital-to-analog hardware connected via RJ-45 twisted pair cables to an FPGA via a custom FMC card.

The input range for analog-to-digital conversion handles up to 1 kV peak, and the digital-to-analog output a maximum voltage swing of ± 10 V. Both devices feature 100 kHz analog bandwidth and built-in isolation barriers for hassle-free deployment. To ensure high timing accuracy, conversion signals are generated directly by the connected FPGA and transmitted via RJ-45 cable. A special digital circuit decodes conversion signals and synchronizes communication to prevent desynchronization due to delays of isolation ICs. DINO is a powerful, modular, and open hardware solution for converting between the digital signals of simulations and the analog signals necessary for interfacing with real devices.

TABLE II
FPGA RESOURCE UTILIZATION OF THE FPGA DESIGNS ON THE VC707 AND ZCU106 FPGAS

FPGA	LUTs	Registers	GTX/GTH Channel
VC707	37411 (12.32 %)	46893 (7.72 %)	5 (17.86 %)
ZCU106	14923 (6.48 %)	24722 (5.37 %)	1 (5 %)

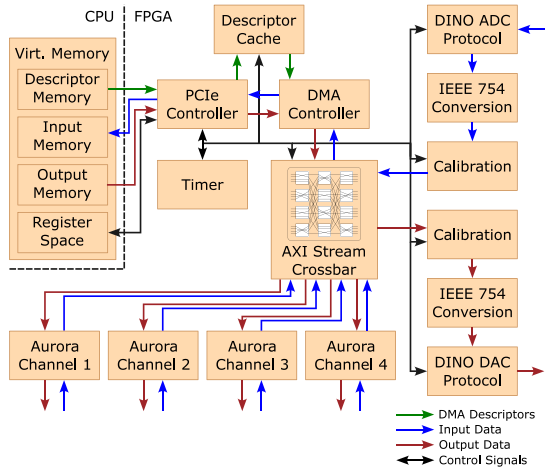

Fig. 5. Overview of the FPGA design for the VC707 with DINO interface and four Aurora interfaces.

B. FPGA Design Enabling a Software-Configurable Simulation Data Path

Building on top of the previously discussed hardware platform, our communication system comprises an FPGA design that enables communication from applications running on a CPU to external devices in a low latency and reconfigurable way.

For this article, we implement two typical external communication interfaces: an Aurora interface for digital communication, e.g., with other simulators, and an interface to DINO, e.g., to supply analog signals to an amplifier to perform an HIL simulation. Additionally, our FPGA design provides a time-step signal for simulation software running on the CPU. Generating the time step on the FPGA yields higher accuracy and lower jitter than is possible in software. The design currently supports the Xilinx VC707 FPGA board using PCIe to connect to a CPU and the Xilinx ZCU106 FPGA board using the ARM Platform interconnect to connect to the MPSoC CPU. Table II shows the FPGA resource utilization of our designs.

Our implementation comprises the following components.

1) An FPGA design that facilitates flexible communication between the CPU and arbitrary FPGA IPs using the AXI Stream bus.
2) A user space polling mode driver that allows interacting with the FPGA with very low latencies.
3) A generic and modular software interface based on the VILLASframework enabling connection to arbitrary simulation software.

Fig. 5 shows an overview of the components of the FPGA design. It allows for real-time capable interconnection of any FPGA IP that supports the AXI Stream protocol to any software that can use the VILLASframework API. However, to showcase the capabilities of the communication system, we focus on specific components typically found in real-time simulators.

1) Aurora: Although some applications require interfacing via analog signals, for many applications, digital interfaces have superior characteristics. Digital signals can be transferred over much larger distances without losing signal quality. Their bandwidth is only limited by the rate of transmission and not by analog effects. Digital transmission also avoids latencies from digital-analog conversions. Because of this, digital interfaces play an important role in real-time simulations. One popular interface for the communication of digital signals in real-time simulations is the Aurora interface [12]. Aurora is a serial interface based on optical fiber channels. We implement an Aurora 8b10b interface compatible with the implementations of the commercial simulators from RTDS and OPAL-RT on our FPGA to allow our simulation to communicate with the wide variety of simulation hardware that supports this protocol. Fig. 5 shows four instances of the Aurora interface at the bottom.

2) AXI Stream Crossbar: As simulation scenarios can exhibit a multitude of communication patterns, rewiring of devices that are part of a simulation is often necessary. To avoid the error-prone and labor-intensive task of physically changing connections, enabling the connection to be changed purely in software was a key requirement for our implementation. Our FPGA design uses an AXI Stream Crossbar IP that connects to all communications endpoints, i.e., analog interfaces, Aurora channels, the CPU, and any other AXI Stream compatible IP. The AXI Stream Crossbar IP allows the routing between these communications endpoints to be configured via a register from an application running on the CPU. This implements a circuit switched communication network, where fixed connections between endpoints are established before a simulation is started. Although this does not always optimally use the available bandwidth of connected channels, fixed connections are very predictable and incur minimal latency overhead, making them ideal for real-time communication.

3) Direct Memory Access Controller: Our design uses a direct memory access (DMA) controller for reading and writing signals communicated with external devices directly to and from main memory. This offloads much of the communication between the FPGA and the CPU to the FPGA, allowing the simulation running on the CPU to spend more time on computations rather than communication. To further optimize the latency of communication, we use the scatter-gather feature of the DMA Controller. This allows the transmissions to be controlled using descriptors stored in main memory, rather than requiring slower register accesses. A read or write transmission consists of building a descriptor that describes the transmission, i.e., the memory address and length to read from or write to, and setting a register of the controller to the address of the prepared descriptor. When the transmission is finished, the controller sets a status flag in the descriptor. An application can poll this flag in main memory to synchronize with the communication, rather than having to poll a device register, or use interrupts, both of which we found to be significantly slower. The memory segment used for communication with the DMA controller does not change during a simulation. Therefore, the DMA descriptor does not change, except for the status flag. To avoid the DMA controller having to retrieve the same descriptor from main memory every time step, we implemented a cache for the descriptors (shown at the top of Fig. 5). In this way, descriptors that have previously been read, can be retrieved much faster than when they are read from main memory. The cache implements a write-through policy, where it forwards writes to the cached memory immediately to main memory. This ensures that updates of the status flag in the descriptor are immediately visible to the CPU. With our implementation, writing or reading signals from any of the interfaces connected to the AXI Stream Crossbar requires only a single register access to start the transaction in the DMA Controller. Afterward, an application can poll the descriptor status flag to wait for the transaction to finish.

4) DINO IP: For the DINO analog interface, we implement FPGA IP that manages control and data flow to the ADCs and DACs used in our hardware implementation. The right of Fig. 5 shows how data flows to and from the DINO ADC and DAC via protocol logic, IEEE 754 floating-point conversion, and calibration logic. The data is exchanged via AXI Streams using floating-point values directly corresponding to voltages seen at the DINO input and output. Conversions are triggered using a timer programmable from the CPU. For the DINO DAC, conversion can optionally also be triggered immediately when new data has been received.

C. Low-Latency Modular CPU to FPGA Communication Interface

The hardware and FPGA design provide a foundation for flexible, low-latency communication with external devices. How simulation software executing on the CPU accesses the FPGA is of similar importance to guarantee optimal performance. Similarly to other peripheral devices, such as network interfaces or GPUs, our FPGA requires drivers, which provide application interfaces for configuration and communication. Drivers are typically implemented as kernel modules running in kernel space, which gives them higher privileges and isolates them from applications running in user space. However, this requires an additional interface between the user space application and the kernel space driver, typically also involving latency-intensive context switches between kernel and user space. To avoid this, we leverage VFIO, an interface allowing user space applications to directly access peripheral devices without involving kernel space code. We use a PMD instead of interrupt-driven patterns for FPGA synchronization. In this way, our drivers can achieve significantly lower latencies than traditional drivers. The drivers are based on the Xilinx embedded drivers [26], modified for use with VFIO and optimized for latency.

Although in this article, we focus on using FPGAs to implement the communication interfaces of a real-time simulator, FPGAs are fundamentally highly flexible, allowing implementation of a multitude of IPs. Our software design takes this into account by providing a driver interface for arbitrary IPs implemented on the FPGA and automatically loading the drivers necessary to configure and communicate with them. This contrasts with other implementations of real-time simulators, where the external and CPU interfaces are fixed so that drivers do not need to change. While making the CPU-to-FPGA interface less complex, this leads to the periphery being more inflexible.

We integrate our software into the VILLASframework to benefit from the existing communication management and protocol translation functionalities. To enable the use of our communication system, we added a plugin to the VILLASnode component, which enables communication with FPGAs. Using the hardware handoff file generated alongside the FPGA bitstream, we generate a JSON description of the FPGA design. The plugin uses this description to load the appropriate drivers for the design and initialize all IPs. Using VFIO [27] the drivers run within the VILLASnode process but have full access to the memory space of the IPs. The DMA controller additionally requires the use of the IOMMU that allows it to access the virtual address space of the VILLASnode application. Our implementation interacts only with the VFIO kernel interface when the drivers are loaded and, during the simulation, does not interact with the kernel at all. This is an important property of our implementation, as it means that no context switches are necessary during the execution of critical real-time code. Furthermore, driver implementation and customization is less complex without requiring kernel driver development. Due to the modular nature of our implementation, it enables arbitrary AXI4 Stream IP to be connected to VILLASnode.

D. Integration With DPsim

Any real-time capable software that needs to be connected to the communication system can use the VILLASnode API to exchange signals with another connected device or software. To show the capabilities of the communication system, we integrated it with DPsim, an open source real-time-capable simulation software for power grids. By coupling our communication system with DPsim, we create a real-time simulator with similar capabilities as the commercial simulators from OPAL-RT and RTDS. While DPsim offers models and solvers for multiple simulation domains, in this article we focus on using it for EMT simulations, as this has the highest requirements on communication latency. The accuracy of DPsim simulations has previously been validated in various publications [7], [28]. DPsim already had an interface to VILLASnode. However, this was based on queues and threads, and was built for communication in geographically distributed simulations where latency requirements are relaxed, because signals are not exchanged as instantaneous values [7]. This is why we built a lower latency interface to VILLASnode. With this interface, any simulation signal in DPsim can be exchanged directly with any plugin of VILLASnode. Furthermore, we added DPsim time step synchronization to FPGA signals and time step overrun detection. For this, DPsim receives a counter-value from either an FPGA-internal counter or via Aurora from a remote device. DPsim then compares the value with that of the previous time step and reports an overrun if the difference is larger than one. At the end of a time step, DPsim issues a read transaction to the DMA Controller and waits for the transaction to finish. Receiving a transaction from the DMA Controller marks the beginning of the next time step.

In addition to the implementation of the interface to VILLASnode, several latency improvements of the DPsim code were necessary. This included reducing logging and unnecessary file system interactions, moving memory allocations to the initialization, and implementing a lower latency buffered result writer. While we modified the interface to VILLASnode, we made no changes to models or solvers. Our modifications significantly improved the real-time performance at small time steps, allowing EMT simulations with microsecond-level time steps.

IV. Evaluation

To evaluate the system, we performed several experiments to investigate specific performance figures and applicability in general. We measured the end-to-end frequency response from the analog input to the analog output via a DPsim simulation of a wire, finding a -3 dB gain at 86.7 kHz and a -10° phase shift at 2.729 kHz. The phase shift stems from communication and computation latencies, while the analog gain error of DINO only becomes relevant at significantly higher frequencies. Because this result highlights latency as the dominant limitation, we focus on the metrics that govern real-time capability: the achievable simulation time step and the latency and jitter with which the simulator reacts to inputs. We compare the time-step quality of our DPsim-coupled communication system to the commercial RTDS and OPAL-RT simulators and quantify end-to-end reaction latency to assess the dynamics each system can capture. Finally, we demonstrate the broader applicability of the approach through a joint cosimulation experiment involving all three simulators and a distributed, scaled PHIL setup.

All experiments are conducted in the real-time laboratory of the Institute for Automation of Complex Power Systems at RWTH Aachen University (see Fig. 3). For the experiments, we use an RTDS NovaCor simulator, an OPAL-RT OP5700 simulator, and our DPsim-based system based on a Dell 740xd server using two Intel Xeon Gold 6132 CPUs, 256 GB memory and a VC707 FPGA connected via PCIe. To evaluate the time step quality we use a Stanford Research DS345 signal generator and a Rohde & Schwarz MXO44 oscilloscope to externally provide and measure signals. For our cosimulation and PHIL experiments, we interconnect the three simulators and MIOB using multimode optical fiber cables.

A. Time Step Benchmarks

In a real-time simulator, the time step is defined by a time step signal that describes the beginning of a new time step and the end of the previous time step. The time step signal is also used to detect overruns, where the simulation cannot calculate a time step before the next one begins. The time step signal can be thought of as a clock in a digital circuit, that is, it is generated by a base clock and will be out of phase and drift with respect to other clocks, most importantly with respect to the real time. To limit the compounding error the mismatch of simulation time and real time will have on the simulation results, the time step signal must be of high accuracy and low jitter. The interfaces with which real-time simulators are connected to the real world also introduce latency, consisting of communication and conversion latencies. For analog interfaces, there is an additional latency of the analog front end, i.e., filters, amplifiers, and safety components. In an HIL simulation, the real-time simulator receives input from devices via an interface, calculates the next simulation time step, and sends the reaction of the simulation back to the device via the interface. Ideally, this happens in a defined number of time steps that are independent of the time step duration. The bandwidth of the simulation thus increases linearly with smaller time step duration until the minimum time step or the bandwidth of the interface is reached.

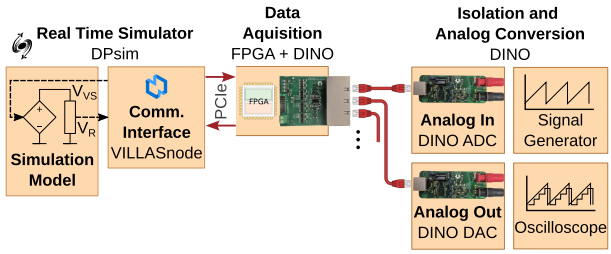

Fig. 6. Latency evaluation setup for DPsim.

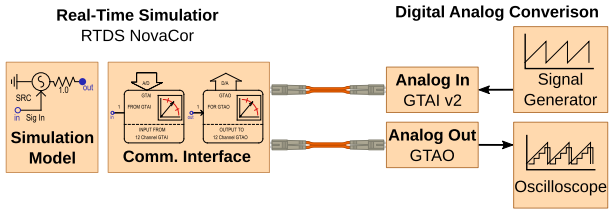

Fig. 7. Latency evaluation setup for RTDS.

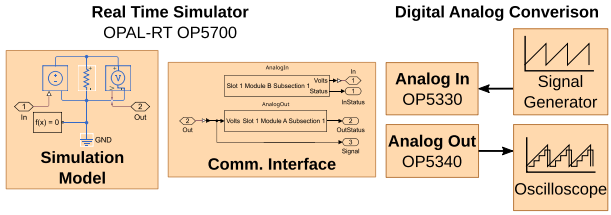

Fig. 8. Latency evaluation setup for OPAL-RT.

In summary, to accurately quantify the performance of a real-time simulator, the quality of the time step and the minimum latency of the reaction to external events are important figures. For the evaluation of the time step quality, we cannot rely on simulator-internal timing measurements because they will not capture the latencies of the interfaces. Furthermore, the clocks of each simulator, an external signal source, and any measurement equipment will drift in respect to each other and, even if synchronized, have an unknown phase relationship. To avoid inaccuracies caused by this, we set up our communication system coupled with DPsim, an RTDS NovaCor simulator, and an OPAL-RT OP5700 simulator as shown in Figs. 6, 7, and 8. We connected a Stanford Research DS345 signal generator generating a 5 kHz sawtooth signal to the analog input of the simulators. This signal is input in the simulation model to a voltage source, which in turn connects to a resistor. The voltage over the resistor is output via the analog output to a Rohde & Schwarz MXO44 oscilloscope. Using a sawtooth signal with a frequency similar to the time step of the simulator means that the value read at the analog input is related to the time of measurement. In this setup, the signals at the analog outputs of the simulators show a staircase pattern due to the sampling of the input signal as visible in Fig. 9. The length of the steps corresponds to the time step of the simulation. Calculating the difference between the time a step change in the output signal occurs and the time the input signal had the signal value of the new step yields the complete latency of the analog interface and simulation. For example, in the figure, the arrow shows a complete latency of 8.55 μs for the particular

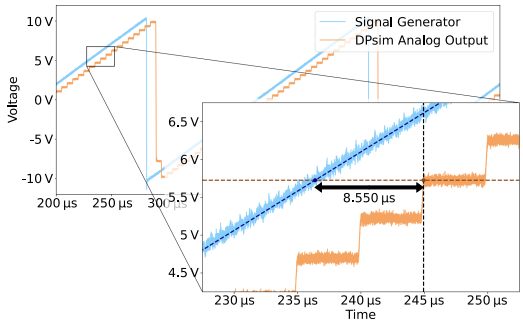

Fig. 9. Methodology to derive latencies of real-time simulations.

TABLE III
COMPARISON OF TIME STEP JITTER FOR THE DPSIM, OPAL-RT, AND RTDS SIMULATIONS

Simulator	Mean Time Step	Max. Jitter	Standard Deviation
DPsim	4.9998 μs	0.08 μs (1.6 %)	0.0142 μs (0.28 %)
OPAL-RT	4.9989 μs	0.42 μs (8.4 %)	0.0734 μs (1.47 %)
RTDS	5.0004 μs	0.36 μs (7.2 %)	0.0695 μs (1.30 %)

time step. Compared to this methodology, using sinusoidal signals and deriving the latency from phase differences calculated from zero crossing of the signals has a larger error due to sampling effects and is prone to inaccuracies if the analog interfaces show nonlinearities. Using the described evaluation setup, we connect the three simulators to the same signal source, start all simulations concurrently, and collect oscilloscope data for ten phases of the input signal and the output of the three simulators. By further processing the data, we derive various metrics on time step performance. We documented our data processing using Jupyter Notebooks to allow retracing our calculations.[4] To quantify the quality of the time step, we calculated the length of each step in the output signal (see Fig. 9). Because the steps are a result of the sampling occurring in each time step, the length of the steps directly corresponds to the time step. By collecting the length of the steps in the output signal, we can derive the mean, the jitter as the difference between the maximum and minimum value, and the standard deviation for the time step. Table III shows the time step quality of each simulator at a time step of 5 μs.

The results show high time step accuracy for all simulators. Our DPsim-based simulator shows the lowest time step error, with RTDS having twice the error and OPAL-RT four times higher. Jitter and standard deviation results also show best performance for our DPsim-based simulator. Our simulator implementation not only matches commercial simulator time step quality but performs better in this benchmark. Using the same data, we can also derive an average latency for retrieving analog input signals, calculating time steps, and outputting analog signals. Fig. 10 shows these latency results and standard deviations.

For all three simulators, we chose the minimally possible time step before they experienced overruns. While with our simulator, we can disable the synchronization of the analog output to the time step signal, this is not possible with the simulators of RTDS and OPAL-RT. For these simulators, the minimal latency is thus related to the time step. For the OPAL-RT and DPsim-based

[4][Online]. Available: https://git.rwth-aachen.de/acs/public/publications/2024-oshil

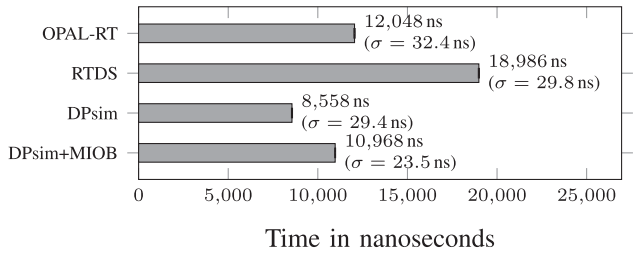

Fig. 10. Latency and standard deviation for reaction from analog in via simulation to analog out based on at least 300 measurments.

experiments, we used a time step of 5 μs. RTDS was able to use a time step of 3.9 μs. We also measure the latency for a distributed simulation setup, where, compared to the setup in Fig. 6, we additionally pass the signals via Aurora to an MIOB to which we connect the DINO analog interface.

The results show that although we expected the output of the RTDS and OPAL-RT simulators to be synchronized with the beginning of a time step, the measured latency is not an integer multiple of the configured time step. For larger time steps, our simulator has a latency of one time step, the OPAL-RT simulator has a latency of two time steps, and the RTDS simulator has a latency of three time steps.

However, for the smaller time steps considered here, the interface latency and analog effects have a significant influence on the overall latency. This is visible in Fig. 10, where none of the simulators achieves a latency the same multiple of the time step as for larger time steps. Although configured with the lowest time step, the RTDS simulator shows the largest latency of more than four times the time step. The OPAL-RT simulator has a latency of approximately 2 μs more than the expected two time steps. Our DPsim-based simulator has the lowest latency, but shows a larger, approximately 3.5 μs, increase over the expected single time step latency. These results indicate that the latency of the interfaces is not accounted for when detecting overruns. This means that, from the point of view of devices connected to the analog interface, an overrun can occur, even when the simulator does not detect or report one. Regarding simulation performance, our DPsim-based simulator has the lowest latency, meaning it can reproduce the highest dynamics and largest bandwidth, making the HIL simulation with it more accurate than when using the RTDS and OPAL-RT simulators. The distributed setup including an MIOB shows an approx. 2.4 μs higher latency, due to the signals having to pass via an Aurora channel and an additional FPGA. Although for the time steps considered here, this represents a significant additional latency, this setup still performs better than the RTDS and OPAL-RT simulators.

Because of its open source nature, we can inspect all parts of the simulation of our DPsim-based simulator. In this way, we can measure not only the overall latency, but also the individual parts of communication, simulation, and synchronization. For this, we Integrated Logic Analyzers (ILAs) on the FPGA that allow us to collect waveforms of FPGA-internal signals. Using data collected from ILAs, we can derive how long it takes to communicate with DINO, to perform PCIe communications, to compute the simulation, and to wait for the beginning of the next time step. Fig. 11 shows the results for a time step of 5 μs and a time step of 10 μs, where the analog output is synchronized with the time step signal. The figure shows that communication accounts for approximately 70.5% of the 5 μs time step. Of the communication, a large part of the latency comes from the ADC and DAC protocols, which require serial communication using a comparably slow clock. PCIe communication adds significantly more latency after the DPsim calculation than before it. This is because the first PCIe communication consists only of polling the main memory for the DMA descriptor status flag, while the second PCIe communication also starts both read and write transactions by writing to a DMA controller register (see Section III-B).

Fig. 11 also shows the timings of a previous and next time step. Fig. 11(a) shows that the time steps overlap. With the larger time step in Fig. 11(b), this does not happen. This is because our simulator only detects overruns when new data arrives from the FPGA before the last results have been written to it. In other words, only the DPsim computations (yellow), and the PCIe communication and VILLASnode code (blue) are captured by the overrun detection. Whether simulations such as in Fig. 11(a) should be allowed is debatable. On the one hand, they might lead to unintended behavior of the simulation. On the other hand, lower time steps can improve simulation accuracy even when the latency across communication interfaces has a larger latency. While the measurements in Fig. 10 were made externally, the measurements in Fig. 11 were made internally within the FPGA. The second measurements therefore do not capture latencies of the analog front ends of analog input and output. The difference in the total time step latency between both figures is due to the latency of these front ends. Their effect of an additional latency of approximately 1.76 μs means that additional tuning of the analog components has the potential to significantly improve the latency.

B. Cosimulation

One use case for real-time simulation is coupling multiple simulators for cosimulation. This increases computation power [11] and leverages individual simulator strengths [29], [30], e.g., proprietary or existing models, or specialized communication interfaces. In our setup (see Fig. 3), simulators connect via Aurora to our communication system, so cosimulation requires only configuring the FPGA crossbar for appropriate communication channels without physical rewiring. To demonstrate cosimulation capabilities, we implemented the IEEE WSCC 9 Bus benchmark model on the RTDS, OPAL-RT, and DPsim simulators. Fig. 12 shows our configuration. Most components run on RTDS, while the bus 5 load runs in DPsim and the bus 6 load in OPAL-RT. We use ideal voltage and current sources to implement the electrical interface between the simulators. The voltages at buses 5 and 6 are sent to DPsim and the OPAL-RT simulator, respectively, where they are fed into voltage sources. The resulting currents through the loads are sent back to RTDS, where we use them as set points for current sources. To synchronize the simulators, we use the communication packages received via Aurora as the time step signal for DPsim and the OPAL-RT simulator. To achieve this with the OPAL-RT simulator, we implement a custom RT-XSG FPGA bitstream that synchronizes the time step to an Aurora channel. For RTDS, this is not possible, which is why RTDS generates the time step for the entire simulation. To start the simulation, we first start the RTDS model. It will use zero Amperes as the set points for the current sources until the other simulators start communicating. Afterward, we start the DPsim and OPAL-RT simulations, which

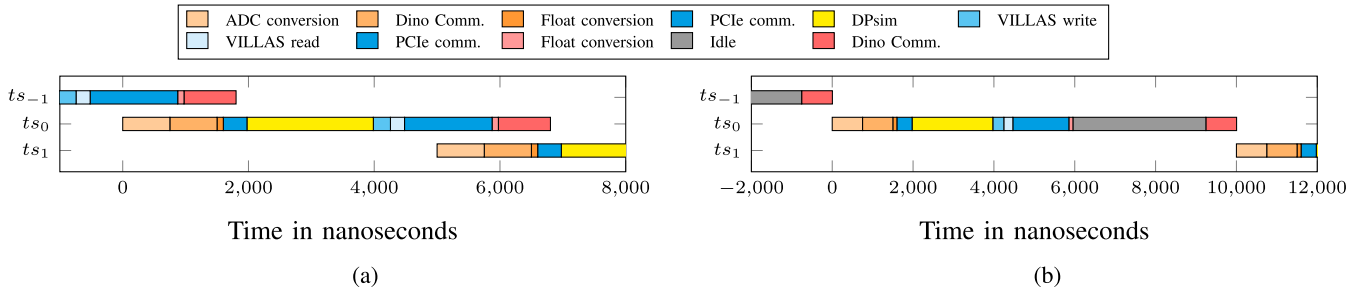

Fig. 11. Detailed breakdown of the time step of the DPsim-based simulator, derived using the FPGA integrated logic analyzers. (a) Breakdown of a 5 μs DPsim time step. (b) Breakdown of a 10 μs DPsim time step with DAC trigger synchronized to the time step signal.

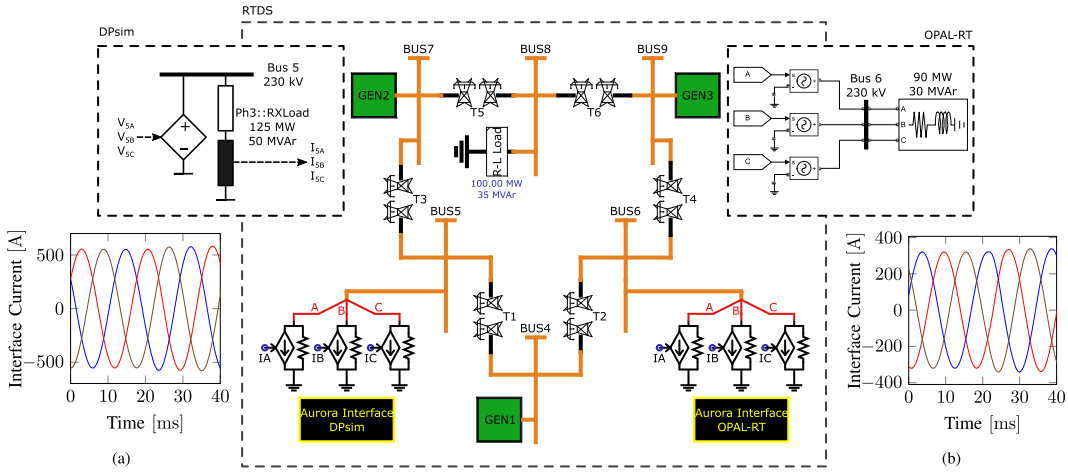

Fig. 12. Cosimulation of IEEE WSCC 9 bus with load at bus 5 simulated in DPsim, load at bus 6 simulated in OPAL-RT and remaining components simulated in RTDS. We observe the interface currents shown on the left and right sides during load step at bus 8 (300 MW, 110 MVAr to 100 MW, 110 MVAr). (a) DPsim Aurora Interface. (b) OPAL-RT Aurora Interface.

will start receiving data from RTDS and synchronize to the time step signal. We can then activate switches in the RTDS model that begin feeding the received current signals from DPsim and the OPAL-RT simulator to the current sources in the RTDS model. To quantify the delay introduced by the communication, we also communicate the time step number from RTDS to the other two simulators. These echo the number back to RTDS, where we compare the received value with the then current time step number. For DPsim, we find a communication delay of one time step, for OPAL-RT it is two time steps and for RTDS it is three time steps. Although there are techniques to compensate for these delays to avoid inaccuracies and instabilities, their implementation is beyond the scope of this article. To verify the cosimulation setup under dynamic behavior, we added a load step to the simulation. Fig. 12(a) and (b) shows that the current signals increase due to the load step without causing instabilities. The cosimulation behaves similar to the monolithic solution, despite the delays introduced by the communication interfaces.

C. Power Hardware-in-The-Loop Simulation

Another typical use case for real-time simulations is PHIL. In a PHIL simulation, a real-time simulation of an electrical grid is connected to real hardware using a power transmitting interface. The interface signals in a PHIL simulation are voltages and currents that represent power flow.

Fig. 13 shows the distributed scaled PHIL simulation setup that we implemented using our communication system. We use

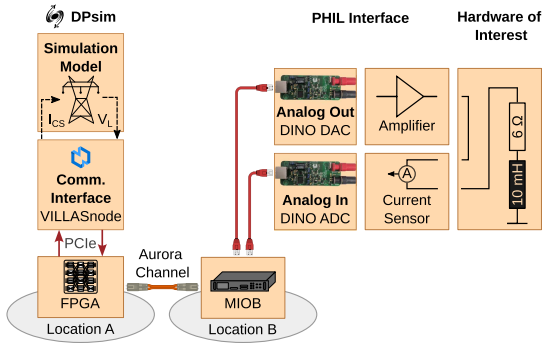

Fig. 13. Distributed PHIL setup.

the same hardware as previously described to run a simulation of the IEEE WSCC 9 bus in DPsim. We modified the 9 bus model to include a current source at phase A of bus 6, where we inject a current measured at a real load acting as a device under test. Because the simulation server is located in a different room from where the real load is located, we connect the simulation server to an MIOB located close to the real load. To convert between digital signals and real voltages and currents, we use DINO, a power amplifier, and a current sensor. The real load consists of a series *R-L* load. In this experiment, we scale the real voltage and current because we lack access to transmission grid level power amplifiers. We scale the current by a factor of $\frac{1}{100}$ and

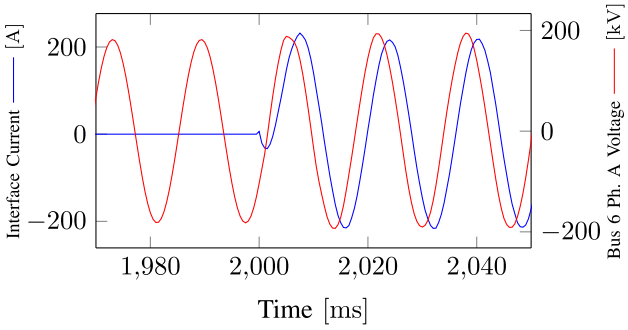

Fig. 14. Distributed PHIL experiment with real *R-L* load connected to simulated WSCC 9-bus at $t = 2$ s.

the voltage by a factor of $\frac{15}{230000}$. This results in the simulation experiencing a load of $P = 57.5$ MW and $Q \approx 91.514$ MVAr at the PHIL interface.

Fig. 14 shows the the voltage and current at the PHIL interface in the DPsim simulation when the real load is connected to the simulation. From the simulation results, we derive a phase angle of $\phi = \Delta t \cdot 360° \cdot f = 44.9°$, while analytically the real load has a phase angle of $\phi = \tan^{-1}(\frac{2\pi f L}{R}) \approx 32.14°$. This discrepancy represents approximately 590 μs delay and is due to the single time step delay of the communication system (50 μs), a delay of the amplifier and current sensor ($< 0.4\,\mu$s) and tolerances of the real hardware (up to 210 μs at 15% tolerance).

V. Conclusion

We introduced an open source communication system for real-time simulators that is fully reconfigurable in software, eliminating physical rewiring requirements. We demonstrated the feasibility of our communication system using cosimulation and distributed PHIL experiments, achieving performance competitive with commercial solutions while providing greater flexibility and openness. With a hardware cost for the communication system used in this article (consisting of computing server, FPGA, and DINO hardware) of well below $10 000, the communication system is many times more affordable than commercial alternatives. The price of many commercial real-time simulators also includes support services that help power system engineers to focus on their studies without having to become experts in real-time simulation technology. We acknowledge that our communication system may be significantly more challenging to set up and maintain than commercial solutions. However, we believe in the significant practical benefits of open source products for power systems research and education. By enabling sub-10 μs real-time simulations with open-source components, our system makes advanced HIL testing accessible to smaller research institutions and educational facilities that cannot afford expensive commercial simulators. The software-reconfigurability allows researchers to quickly adapt their experimental setups for different power system scenarios, from microgrid studies to transmission-level PHIL experiments. Furthermore, the demonstrated interoperability with commercial simulators (RTDS, OPAL-RT) provides a migration path for existing research facilities while enabling hybrid simulation setups that leverage the strengths of different platforms. Unlike commercial solutions, our approach of separating communication system and simulation software allows researchers to choose or develop simulation software best suited for their needs, without being locked into the ecosystem of one specific vendor. Using VILLASnode, researchers can specify communication patterns in JSON. VILLASnode then configures the communication system and performs protocol translation. This moves laboratory setups toward machine-readable, declarative descriptions, improving reproducibility while hiding device-specific configuration details.

The communication system supports complex simulations beyond those demonstrated. Fig. 3 shows how we have deployed the presented communication system in the real-time laboratory of the Institute for Automation of Complex Power Systems at RWTH Aachen University. We have connected numerous devices and simulators to a common Aurora network. We imagine expanding the communication systems to allow for much larger simulation models, interconnecting devices and simulators over a larger geographic area, and including a variety of components. Due to the open source nature of our work, researchers can adapt our implementation to new protocols, add features, and experiment with future technologies.

References

[1] M. D. Omar Faruque et al., "Real-time simulation technologies for power systems design, testing, and analysis," *IEEE Power Energy Technol. Syst. J.*, vol. 2, no. 2, pp. 63–73, Feb. 2015.

[2] A. Benigni, T. Strasser, G. De Carne, M. Liserre, M. Cupelli, and A. Monti, "Real-time simulation-based testing of modern energy systems: A review and discussion," *IEEE Ind. Electron. Mag.*, vol. 14, no. 2, pp. 28–39, Jun. 2020. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9127176

[3] P. Forsyth and R. Kuffel, "Utility applications of a RTDS simulator," in *Proc. Int. Power Eng. Conf.*, 2007, pp. 112–117.

[4] B. Lu, X. Wu, H. Figueroa, and A. Monti, "A low-cost real-time hardware-in-the-loop testing approach of power electronics controls," *IEEE Trans. Ind. Electron.*, vol. 54, no. 2, pp. 919–931, Feb. 2007.

[5] J. A. Martinez-Velasco, *Transient Analysis of Power Systems: Solution Techniques, Tools and Applications*. Hoboken, NJ, USA: Wiley, 2015.

[6] Typhoon HIL documentation - system architecture basics, Typhoon HIL Inc., Accessed: Nov. 6, 2024. [Online]. Available: https://www.typhoon-hil.com/documentation/typhoon-hil-software-manual/concepts/system_architecture_basics.html

[7] M. Mirz, S. Vogel, G. Reinke, and A. Monti, "DPsim—A dynamic phasor real-time simulator for power systems," *SoftwareX*, vol. 10, 2019, Art. no. 100253.

[8] R. Delgado, J. Park, and B. W. Choi, "MPSoC: The low-cost approach to real-time hardware simulations for power and energy systems," IFAC-PapersOnLine, vol. 52, no. 4, pp. 57–62, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896319304914

[9] M. Difronzo, H. L. Ginn, and A. Benigni, "A low latency parallel bus interface for high-speed multi-FPGA RT-simulations," in *Proc. IEEE Electric Ship Technol. Symp.*, 2021, pp. 1–7.

[10] S. Biookaghazadeh, M. Zhao, and F. Ren, "Are FPGAs suitable for edge computing?," in *Proc. USENIX Workshop Hot Topics Edge Comput.*, Jul. 2018. [Online]. Available: https://www.usenix.org/conference/hotedge18/presentation/biookaghazadeh

[11] M. Stevic et al., "Multi-site European framework for real-time co-simulation of power systems," *IET Generation, Transmiss. Distrib.*, vol. 11, no. 17, pp. 4126–4135, 2017. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-gtd.2016.1576

[12] Xilinx, Inc., "Aurora 8b/10b protocol specification," Tech. Rep., Aug. 2014. [Online]. Available: https://web.archive.org/web/20260123084429/https://docs.amd.com/v/u/en-US/aurora_8b10b_protocol_spec_sp002

[13] F. Adler, A. Benigni, H. Stagge, A. Monti, and R. W. De Doncker, "A new versatile hardware platform for digital real-time simulation: Verification and evaluation," in *Proc. IEEE 13th Workshop Control Model. Power Electron.*, 2012, pp. 1–8.

[14] M. Chiandone, A. Vicenzutti, D. Bosich, A. A. Tavagnutti, N. Barbini, and G. Sulligoi, "Open source hardware in the loop real-time simulation of zonal DC systems," in *Proc. Open Source Model. Simul. Energy Syst.*, 2022, pp. 1–6.

[15] RTAI - Real, Time application interface official website, the RTAI team, 2021, Accessed: Nov. 5, 2024. [Online]. Available: https://www.rtai.org/index.html
[16] F. Reghenzani, G. Massari, and W. Fornaciari, "The real-time Linux kernel: A survey on PREEMPT_RT," *ACM Comput. Surv.*, vol. 52, no. 1, Feb. 2019, Art. no. 18. [Online]. Available: https://doi.org/10.1145/3297714
[17] J. M. Rodriguez-Borbón, A. Kalantar, S. S. R. K. C. Yamijala, M. B. Oviedo, W. Najjar, and B. M. Wong, "Field programmable gate arrays for enhancing the speed and energy efficiency of quantum dynamics simulations," *J. Chem. Theory Computation*, vol. 16, no. 4, pp. 2085–2098, 2020. [Online]. Available: https://doi.org/10.1021/acs.jctc.9b01284
[18] M. Véstias and H. Neto, "Trends of CPU, GPU and FPGA for high-performance computing," in *Proc. 24th Int. Conf. Field Program. Log. Appl.*, 2014, pp. 1–6.
[19] D. Weller, F. Oboril, D. Lukarski, J. Becker, and M. Tahoori, "Energy efficient scientific computing on FPGAs using OpenCL," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2017, pp. 247–256. [Online]. Available: https://doi.org/10.1145/3020078.3021730
[20] J. Klimt, N. Eiling, F. Wege, J. Baude, and A. Monti, "The role of open-source software in the energy sector," *Energies*, vol. 16, no. 16, 2023. [Online]. Available: https://www.mdpi.com/1996-1073/16/16/5855
[21] Z. Dong, S. Debnath, W. Li, Q. Xia, P. R. Marthi, and S. Chakraborty, "Real-time simulation framework for hardware-in-the-loop testing of multi-port autonomous reconfigurable solar power plant (mars)," in *Proc. IEEE Energy Convers. Congr. Expo.*, 2021, pp. 3160–3167.
[22] M. A. H. Broadmeadow, G. R. Walker, and G. F. Ledwich, "Modular and scalable control and data acquisition system for power hardware in the loop (PHIL) amplifiers," *J. Eng.*, vol. 2019, no. 17, pp. 3655–3659, 2019. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/joe.2018.8035
[23] L. Barbierato et al., "Exploring stability and accuracy limits of distributed real-time power system simulations via system-of-systems cosimulation," *IEEE Syst. J.*, vol. 17, no. 2, pp. 3354–3365, Feb. 2023.
[24] G. De Carne and D. Kottonau, "Power hardware in the loop laboratory testing capability for energy technologies," in *Proc. AEIT Int. Annu. Conf.*, 2022, pp. 1–5.
[25] A. Monti et al., "A global real-time superlab: Enabling high penetration of power electronics in the electric grid," *IEEE Power Electron. Mag.*, vol. 5, no. 3, pp. 35–44, Mar. 2018.
[26] X. Inc., Baremetal drivers and libraries - Xilinx wiki, Accessed: Dec. 6, 2024. [Online]. Available: https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841745/Baremetal+Drivers+and+Libraries
[27] T. kernel development community, "VFIO - virtual function i/o," Accessed: Dec. 7, 2024. [Online]. Available: https://docs.kernel.org/driver-api/vfio.html
[28] M. Mirz, J. Dinkelbach, and A. Monti, "Dpsim—advancements in power electronics modelling using shifted frequency analysis and in real-time simulation capability by parallelization," *Energies*, vol. 13, no. 15, 2020, Art. no. 3879.
[29] C. Gomes, C. Thule, D. Broman, P. G. Larsen, and H. Vangheluwe, "Co-simulation: A survey," *ACM Comput. Surv.*, vol. 51, no. 3, May 2018. [Online]. Available: https://doi.org/10.1145/3179993
[30] S. M. Mohseni-Bonab, A. Hajebrahimi, I. Kamwa, and A. Moeini, "Transmission and distribution co-simulation: A review and propositions," *IET Generation, Transmiss. Distrib.*, vol. 14, no. 21, pp. 4631–4642, 2020. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-gtd.2020.0244



Niklas Eiling received the M.Sc. degree in computer engineering from RWTH Aachen University, Aachen, Germany, in 2018.

From 2018 to 2025, he was a Research Associate with the Simulation Infrastructure and High-Performance Computing Team, Institute for Automation of Complex Power Systems (ACS), E.ON Energy Research Center, RWTH Aachen University. His research interests include GPU and FPGA virtualization, checkpoint/restore mechanisms for heterogeneous computing, and real-time communication systems for power system simulations.



Manuel Pitz received the M.Sc. degree in electrical engineering and information technology from KIT Karlsruhe, Karlsruhe, Germany, in 2016.

He is working as a Research Fellow with the Institute for Automation of Complex Power Systems, RWTH Aachen University, Aachen, Germany. He has been working in the field of renewable technologies, grid monitoring, measurement equipment for grid applications, and communication for grid monitoring since 2016.



Antonello Monti (Senior Member, IEEE) received the M.Sc. (*summa cum laude*) and Ph.D. degrees in electrical engineering from Politecnico di Milano, Milano, Italy, in 1989 and 1994, respectively.

He started his career with Ansaldo Industria and then moved in 1995 to Politecnico di Milano as an Assistant Professor. In 2000, he joined the Department of Electrical Engineering, University of South Carolina, Columbia, SC, USA, as an Associate and then Full Professor. Since 2008, he has been the Director of the Institute for Automation of Complex Power System within the E.ON Energy Research Center, RWTH Aachen University, Aachen, Germany. Since 2019, he holds a double appointment with Fraunhofer FIT where he is developing the new Center for Digital Energy, Aachen. Currently, he is leading ERC Synergy Project SAFEr Grid (Store-and-Forward Energy Grid). He has authored or coauthored more than 400 peer-reviewed papers published in international Journals and in the proceedings of International conferences.

Dr. Monti is an Associate Editor of the *IEEE System Journal,* an Associate Editor of *IEEE Electrification Magazine*, a Member of the Editorial Board of the *Elsevier Journal SEGAN*, and a Member of the founding board of the *Springer Journal Energy Informatics*. He was the recipient of the 2017 IEEE Innovation in Societal Infrastructure Award.