Late Breaking Results: CHESSY: Coupled Hybrid Emulation With SystemC-FPGA Synchronization

Late Breaking Results: CHESSY: Coupled Hybrid Emulation with SystemC-FPGA Synchronization

Lorenzo Ruotolo*, Giovanni Pollo*, Mohamed Amine Hamdi*, Matteo Risso*, Yukai Chen†, Enrico Macii*, Massimo Poncino*, Sara Vinco*, Alessio Burrello*, Daniele Jahier Pagliari*

*Dept. DAUIN, Politecnico di Torino, Italy, name.surname@polito.it

† IMEC, Leuven, Belgium

***Abstract*—The growing complexity of cyber-physical systems (CPSs) calls for early prototyping tools that combine accuracy, speed, and usability. Virtual Platforms (VPs) provide fast functional simulation, but hybrid co-emulation solutions, in which key digital components are deployed on FPGA, become necessary when accurate timing modelling is required and RTL simulation is too costly. However, existing hybrid emulation tools are mostly proprietary, and rely on vendor-specific FPGA features. To address this gap, we introduce an open-source framework that connects SystemC-based VPs with FPGA emulation, enabling full-system co-emulation of digital and non-digital components. The FPGA accelerates the execution of main digital subsystems, while a wrapper coordinates timing and communication with the VP through JTAG, maintaining synchronization with simulated peripherals. Evaluations using a RISC-V SoC, with an example in the biosignals processing domain, show up to 2500× speedup compared to RTL simulation, while maintaining less than 2× total simulation time relative to pure FPGA emulation.**

***Index Terms*—FPGA, SystemC, hybrid emulation**

I. Introduction and Background

Early prototyping helps shortening time-to-market and reducing costs of Cyber-Physical Systems (CPS) design [1]. A key challenge lies in accurately modeling interactions between digital systems and their surrounding environment (e.g., sensors, actuators, power sources) within a unified framework. Standard approaches rely on high-level SW Virtual Platforms (VPs) such as QEMU or Renode, which enable fast, functional simulation and facilitate multi-domain full-system modeling through frameworks like SystemC AMS [2]–[8]. However, pure SW VPs trade timing fidelity for speed and require separately maintained high-level abstractions of each component.

Conversely, FPGA emulation executes the actual RTL code of digital blocks with cycle accuracy, without incurring the huge costs of RTL simulation. However, it complicates the modelling of interactions with components not available at RTL, or non-digital [9]. VP-FPGA hybrid co-emulation frameworks [10] address this duality by combining a SW VP with FPGA-accelerated modelling of key digital blocks (e.g., processors), but existing implementations either are proprietary/not openly available [11]–[15], rely on vendor-specific features [16], [17] or require RTL instrumentation (e.g., transactors), as for SCE-MI [10] and others [14], [17]–[19].

To address these limitations, we propose **CHESSY** (Coupled Hybrid Emulation via SystemC–FPGA Synchronization), a *fully open-source*, flexible, hybrid prototyping framework that combines FPGA-accelerated RTL execution with loosely timed SystemC models [20]. CHESSY leverages JTAG for VP-FPGA communication and timing synchronization, through lightweight Board Support Package (BSP) stubs, thus being *FPGA-agnostic and requiring no changes to firmware or RTL*

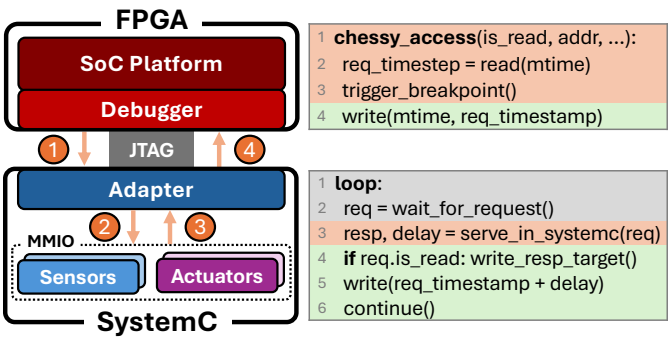

Fig. 1. Overview of CHESSY. The background shades in the pseudocode (right) indicate the timestamp observed by the corresponding hardware (left): *gray* = previous request, *orange* = current request, *green* = current request + simulated peripheral delay.

[13]. The result is a portable and transparent co-emulation environment that bridges cycle-accurate RTL execution of key digital modules with high-level simulation of the rest of the system, including non-digital components.

We demonstrate CHESSY on a RISC-V System-on-Chip [21], [22] for a biosignal-based robot control use case. The results show that we achieve more than three orders of magnitude speedup over RTL simulation while maintaining a total simulation time of less than 2× that of pure FPGA emulation. CHESSY is available at https://github.com/eml-eda/chessy.

II. Methodology

Figure 1 illustrates CHESSY, which connects a SystemC VP to an FPGA-based target through a debugger-mediated link.

The FPGA runs an unmodified design containing a processor and (a subset of) its fully-digital on-chip peripherals, while SystemC, running on the host, simulates the rest of the Memory-Mapped I/O (MMIO) peripherals, especially those interacting with the external world, such as sensors and actuators. A host-side adapter links the two environments by intercepting I/O requests at dedicated breakpoints and maintaining time synchronized through controlled updates to the FPGA's timer registers (e.g., `mtime`). This lets the software running on the FPGA perceive realistic peripheral latencies, even though the latter are simulated on the host.

Figure 1 captures this process: ① whenever the FPGA wishes to communicate with a simulated device, it issues a `SystemCRequest` via a wrapper function `chessy_access`; ② this request is intercepted by the host adapter, a SystemC module, that dispatches it to the appropriate peripheral; ③ the peripheral SystemC model computes the response, after a specified `simulated_delay`; finally, ④ the adapter returns the result to the FPGA and updates timer

registers accordingly. The only FPGA-side requirement to realize this behavior is a debugger server accessible through standard physical connections such as JTAG, providing basic breakpoint handling and memory/register access. This avoids RTL modifications, reduces hardware integration effort, and keeps the framework portable across FPGA vendors, processor models, debuggers, phy links, and simulation environments.

A. Communication and synchronization details

CHESSY synchronizes FPGA and VP executions at each interaction (e.g., a read/write request from the FPGA-emulated core to a peripheral virtualized in SystemC). This minimizes synchronization overheads while maintaining timing accuracy equal to the resolution of the timer register present in the FPGA-emulated system (a requirement of our approach). Namely, the interaction works as follows.

The `chessy_access` function is implemented as a SW stub running on the FPGA, and it packages each communication request into a transaction structure containing operation type (`is_read`), target address (`addr`), data pointer (`data_ptr`), transfer size (`size_bytes`), and local timestamp (`timestamp_us`, read from `mtime`). When the request is ready, a breakpoint notifies the host.

On the host, the adapter controls the debugger to monitor for breakpoints on `chessy_access`, and reads the request transaction data structure; for write operations, it fetches the payload from the FPGA memory pointed to by `data_ptr`; for read operations, it prepares to write the response back after processing. Data transfer occurs over JTAG via the GDB `restore` (write) and `dump` (read) commands.

Next, the SystemC global simulation time is advanced until `timestamp_us`. The adapter then encapsulates the request into a high-level `SystemCRequest` and forwards it to the appropriate SystemC peripheral, determined by the target address. Peripheral models compute the functional response and advance the simulated time to model the peripheral's latency.

When a read reply is ready, the adapter writes the result into FPGA memory and advances time (`mtime`) to `timestamp_us` + `simulated_delay`, ensuring that the FPGA's perception of time reflects the simulated behavior.

III. Results

To validate the proposed approach, we emulated the RISC-V-based Astral platform [21] on a AMD VCU118 FPGA, connected to a Linux workstation running the MESSY [20] SystemC-VP for peripherals simulation. The workstation is equipped with a quad-core CPU, 32GB of RAM, an RV64 GNU GCC / GDB toolchain, and an RTL simulator (Siemens QuestaSim 2022.3) for baseline comparisons.

The simulation time overhead introduced by CHESSY was evaluated against a pure FPGA setup (i.e., without VP communication), using a periodic read–compute–write benchmark with a configurable delay to emulate a generic CPS workload. The benchmark was run while sweeping the computation-to-I/O ratio by varying: **(i)** the data transfer size, from a few bytes up to several kilobytes, and **(ii)** the computation delay, from zero to millions of cycles, thereby controlling the access frequency to VP-simulated models. The results, reported on the left side of Fig. 2, show that the overhead is nearly insensitive

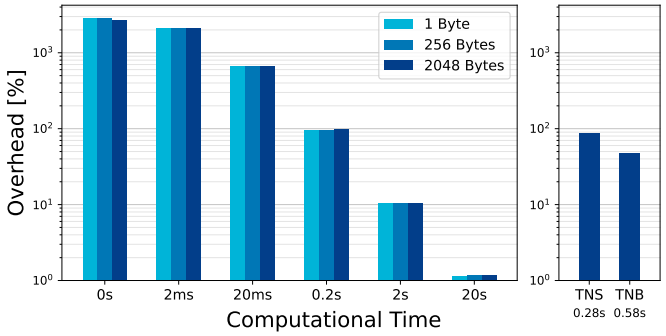

Fig. 2. Simulation time overhead [%] for different transfer sizes and interaction intervals. The two rightmost bars show overheads for 14 M (TNS) and 28 M (TNB) cycle variants of TempoNet (280 ms and 560 ms at 50 MHz).

to transfer size, with relative variation always below 4%, since data exchanges are implemented as host memory dump/restore operations. The total overhead clearly depends on the frequency of FPGA-VP interactions, but reduces to acceptable values for compute-intensive workloads, which are those that necessitate FPGA-acceleration in the first place. On average, each access incurs a nearly constant overhead of less than 100 ms.

We then assessed CHESSY's effectiveness on a realistic biomedical use case, including an Electromyography (EMG) sensor and a robotic arm, both modeled in SystemC, connected to the RISC-V SoC mapped on the FPGA. The application SW reads EMG data, performs gesture recognition via the TempoNet Neural Network (using either a small $\approx$ 4k-parameter / 14 M-cycles variant or a larger $\approx$ 9k-parameter / 28 M-cycles variant), and sends commands to the robotic arm [23]. As with the synthetic benchmark, we measure the average overhead relative to an FPGA-only baseline, reported on the right side of Fig. 2. The 14 M-cycle network exhibits an overhead of about 86%, which reduces to 47% for the 28 M-cycle one. Hence, for realistic applications with similar compute-to-access ratios, the total time remains below $2\times$ that of pure FPGA execution.

Larger and more compute-intensive workloads, common in modern embedded ML, benefit even more: the relative overhead drops to nearly 1% for workloads longer than 20 s. For extremely access-intensive use cases, the protocol remains usable, although simulation may become up to $30\times$ slower. Overall, CHESSY adds modest performance overhead, especially for compute-bound applications, while retaining practical efficiency even in I/O-heavy scenarios.

Finally, to relate CHESSY's performance to a traditional cycle-accurate RTL simulation, the biomedical workload was also executed on QuestaSim. As expected, the FPGA-accelerated setup (including the HIL-based CHESSY protocol) achieves speedups greater than $2500\times$ over RTL.

IV. Conclusions

We presented CHESSY, an open-source, FPGA-agnostic hybrid prototyping framework that does not require RTL modifications. Through a lightweight JTAG-based mechanism, it allows FPGA-emulated hardware to interact with SystemC-modeled peripherals while preserving realistic timing behavior. Experiments show that CHESSY provides over three orders of magnitude speedup compared to RTL simulation, with total execution time remaining within $2\times$ that of pure FPGA emulation for realistic applications.

REFERENCES

[1] J. Shi *et al.*, "A survey of cyber-physical systems," in *2011 International Conference on Wireless Communications and Signal Processing (WCSP)*, 2011, pp. 1–6.

[2] R. Leupers *et al.*, "Virtual platforms: Breaking new grounds," in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2012, pp. 685–690.

[3] QEMU, "QEMU - a generic and open source machine emulator and virtualizer," https://www.qemu.org/.

[4] P. Adelt *et al.*, "QEMU support for RISC-V: Current state and future releases," *International Workshop on RISC-V Research Activities*, 2019.

[5] antmicro, "Develop your iot product with renode," https://renode.io/.

[6] F. Speiser *et al.*, "Embedded system simulation using Renode," *Engineering Proceedings*, vol. 79, no. 1, 2024.

[7] "Ieee standard for standard systemc® language reference manual," *IEEE Std 1666-2023 (Revision of IEEE Std 1666-2011)*, pp. 1–618, 2023.

[8] Accellera Systems Initiative, "SystemC analog/mixed-signal extensions," https://systemc.org/overview/systemc-ams/, 2024.

[9] D. Chen *et al.*, "Fpga design automation: A survey," *Foundations and Trends® in Electronic Design Automation*, vol. 1, no. 3, pp. 195–330, 2006. [Online]. Available: http://dx.doi.org/10.1561/1000000003

[10] Accelera, "Sce-mi, standard co-emulation modeling interface," https://www.accellera.org/downloads/standards/sce-mi.

[11] M. G. Seok *et al.*, "An hla-based formal co-simulation approach for rapid prototyping of heterogeneous mixed-signal socs," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 100, no. 7, pp. 1374–1383, 2017.

[12] Aldec, "Hw/sw co-verification environment for hybrid systems using qemu," https://www.aldec.com/en/company/blog/176–hwsw-co-verification-environment-for-hybrid-systems-using-qemu.

[13] L. Benini *et al.*, "Systemc cosimulation and emulation of multiprocessor soc designs," *Computer*, vol. 36, no. 4, pp. 53–59, 2003.

[14] L. Junger *et al.*, "Sythil: A system level hardware-in-the-loop framework for fpga, systemc and qemu-based virtual platforms."

[15] D. Chiou *et al.*, "Fpga-accelerated simulation technologies (fast): Fast, full-system, cycle-accurate simulators," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, 2007, pp. 249–261.

[16] J. Ou and V. K. Prasanna, "Matlab/simulink based hardware/software co-simulation for designing using fpga configured soft processors," in *19th IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2005, pp. 8–pp.

[17] C. Fu *et al.*, "Chimera: A co-simulation framework combining with gem5 and fpga platform for efficient verification," in *2024 34th International Conference on Field-Programmable Logic and Applications (FPL)*, 2024, pp. 133–139.

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

[19] Y. Y. Tan *et al.*, "Emunoc: Hybrid emulation for fast and flexible network-on-chip prototyping on fpgas," in *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2022, pp. 334–341.

[20] M. A. Hamdi *et al.*, "Integrating SystemC-AMS power modeling with a RISC-V ISS for virtual prototyping of battery-operated embedded devices," in *Proc. of ACM International Conference on Computing Frontiers: Workshops and Special Sessions*, 2024, p. 51–54.

[21] PULP, "Astral, a space computing platform built around cheshire," https://github.com/pulp-platform/astral.

[22] D. Rossi *et al.*, "Energy efficient parallel computing on the PULP platform with support for openmp," in *Proc. of Convention of Electrical & Electronics Engineers in Israel (IEEEI)*, 2014.

[23] M. Zanghieri *et al.*, "Robust real-time embedded emg recognition framework using temporal convolutional networks on a multicore iot processor," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 14, no. 2, pp. 244–256, 2020.