SytHIL: A System Level Hardware-in-the-Loop Framework for FPGA, SystemC and QEMU-based Virtual Platforms

A RISC-V example using AWS cloud servers

Lukas Jünger *, Tobias Röhmel †, Mark Burton †, and Rainer Leupers *,
*RWTH Aachen University, {juenger, leupers}@ice.rwth-aachen.de
†GreenSocs SAS, {tobias.roehmel, mark.burton}@greensocs.com

***Index Terms*—ESL, SystemC TLM, Virtual Platforms, FPGA, Simulation, Emulation, QEMU, Hardware-in-the-loop, HIL**

***Abstract*—Using FPGAs to accelerate Virtual Platforms (VPs) has bent the rule within simulation community that you cannot have both accuracy and speed. Having the ability to run a VP at considerable speed, and being sure that the critical IP in question is being emulated totally accurately opens up the scope of functional verification to include the full software stack. Only hitherto this has typically been limited to very specialized and expensive emulators that are often difficult to scale. Being able to make use of this technology in a fashion conducive to continuous integration and test would be a game changer in many embedded systems groups.**

Recently this has become more possible, due to several developments: First, Amazon has made available their FPGA-enabled cloud instances. They are not the only ones to go this route, and using FPGA cards within an in-house server farm is also becoming feasible. Second, the technology needed to make use of these devices within the context of standard simulation frameworks is now better understood.

In this work an activity in this domain is presented: the SytHIL framework. Using SytHIL RTL descriptions emulated on FPGAs, SystemC TLM models and QEMU models can be combined into one unified simulation. This allows to accurately emulate hardware IPs such as processors or accelerators on the FPGA while other system components are executed at rapid speed as simulation models. Through the SystemC layer, the SytHIL simulation can also be connected to the physical world or other simulated environments. In our case study, we demonstrate the capabilities of SytHIL by emulating different RISC-V processors on Amazon FPGA-enabled cloud instances in combination with SystemC TLM and QEMU models via which the simulation is connected to the host network and thus to the internet.

I. Introduction

Over the past decades, the complexity of HW/SW systems has been steadily increasing. Also, embedded systems have become ubiquitous in our every day life in many application. To verify the correct functionality of everything from planes to disc-drives early in the design cycle, full system simulators, so called Virtual Platforms (VPs), are the de facto standard tool.

While VPs are mainly used for functional software verification, the Register Transfer Level (RTL) description of the components is usually verified using either RTL simulation within complex test benches, Field Programmable Gate Array (FPGA) emulation using special hardware or FPGA prototyping. The advantage of RTL simulation is that it is, by construction, entirely faithful to the design, not only in functionality but in measured performance. However, typical RTL simulators are many orders of magnitude slower than is required for typical software development or reasonable functional verification. Emulation using special hardware or FPGA prototyping bridges this gap by enabling the RTL description to be executed at near real-time speeds. While this seems to be an ideal solution, there are two main problems: The availability of the RTL description itself early in the design cycle, and the availability of the FPGA hardware on which to execute the RTL. To this end, standard VP modeling techniques are typically deployed using languages such as SystemC to implement a model of the proposed design, sufficient for the needs of functional verification. But again, this solution is not without its difficulties. Models are often complex to construct, or may not run as fast as the functional verification engineers would like. Hence, often a hybrid approach is preferred, re-using elements of available RTL designs while also using some standard simulation models of components. The degree of flexibility with which different model components can be used in such an environment is critical, but equally important is the availability of the FPGA hardware that will be used.

Fig. 1. Overview of the SytHIL framework.

This work aims to unite the software and hardware testing strategies into a hybrid approach. It uses the IEEE standard language for VPs: SystemC TLM 2.0, the ubiquitous open-source hypervisor/emulator QEMU [17], and FPGAs. Specifically, and critically, the FPGAs used are available in the cloud using the Amazon Web Services (AWS). This means

Paper	Goal	Emulation	Simulation type	Synchronization	Flow	SCE-MI
SytHIL	Func. hardware & software verification	FPGA	SytemC	Transaction only	both	No
[1]	Hardware verification	Emulator	C-Program	Clock control	sw→hw	No
[2]	Func. co-verification	FPGA	C-Program / ISS	Transaction only	sw→hw	No
[3]	Func. hardware & software verification	FPGA	SytemC	Cycle accurate	hw→sw	No
[4]	Hardware verification	FPGA	C-Program	Transaction only	sw→hw	No
[5]	Func. hardware verification	FPGA	C-Program / ISS	Transaction only	sw→hw	Yes
[6]	Mixed-signal hardware verification	FPGA	Any HLA	Clock control	both	No
[7]	Hardware verification	HDL simulation	C++ Program	Transaction only	sw→hw	No
[8]	Hardware verification	FPGA	C-Program	Transaction only	sw→hw	No
[9]	Hardware verification	FPGA	SytemC	Clock control	sw→hw	Yes
[10]	Algorithm verification co-design	FPGA	C model/ISS	Transaction only	sw→hw	No
[11]	Co-design	HDL simulation	ISS	Clock control	both	No
[12]	Hardware & Software verification	FPGA	C/C++ code	Clock control	sw→hw	No
[13]	Functional verification	phy. hardware	C-Code	Transaction only	sw→hw	No
[14]	Simulation speed up	FPGA	Simple Scalar	Transaction only	sw→hw	No
[15]	Hardware verification	FPGA	C-Program	Transaction only	sw→hw	No
[16]	Hardware & Software verification	FPGA	C++ Program	Clock control	hw→sw	No

Fig. 2. SytHIL comparison with related work.

that the methodology can be deployed at low cost, and scaled to cover the needs of large teams without the need to buy dedicated hardware. The integrated methodology that has been developed in this work, which allows model components from each of these three environments to work together, will be referred to as the *System Level Hardware-in-the-Loop (SytHIL)* framework.

The diagram in Fig. I aims to give an overview of the technology. At the top, a VP is shown consisting of different components, such as a CPU, a video card, an Ethernet MAC, memory and an UART. With SytHIL this VP can be partitioned into parts for execution inside the QEMU domain, the SystemC domain or the FPGA domain. The components are mapped onto these supports. Also, both the SystemC and FPGA environment themselves are hosted within SystemC. The FPGA environment is represented at the edge of the SystemC environment, as there are certain aspects of the FPGA which are not handled within SystemC, and physically the FPGA allows to interface to the real world. The diagram also shows a memory element that is shared between SystemC and the FPGA domain, the extent to which this is implemented will be examined further in this paper, as it is a critical feature which improves simulation speed. Finally, SystemC provides a view of the world to the simulation environment as SystemC is in control of the notion of time, and inputs/outputs to and from the VP. The purpose of the SytHIL framework is to enable all of this partitioning to be carried out as simply as possible, while all the necessary transactors and adapters are deployed under the hood within the framework.

II. Related Work

Functional hardware-software-co-verification was first proposed in 1998 in [13]. In that paper the emphasis is put on running software in the physical target environment. To do that, a functional software simulator is augmented with an FPGA that can interact with physical devices. The simulator can run target software and the interactions with the outside world are handled by the FPGA by toggling the respective pins. The exact mechanism that allows the transfer of data from the software simulation to the hardware simulation is the core technology in all of the hardware-software-co-verification approaches and will be categorized in the following. Shortly after the first paper, [7] introduced a transaction based scheme which increases abstraction by only keeping track of logical transaction information, e.g. address and data to be transferred. Compared to the older approach, where every logical level of the physical bus had to be accounted for, the new approach can increase throughput significantly. [7] also shifted the focus from functional verification to hardware verification. This means that the C simulation was used to write an abstract test bench that would have previously been written in an Hardware Description Language (HDL). This allowed greater productivity for the engineers and re-usability was increased since the test bench could be used for both a software and an RTL model of the design. Since then most research effort has been focused on the transaction based approach because of the increased simulation performance. Several papers improved the general mechanism in various ways like improved abstraction [1], addition of mixed-signal simulations [6] or exploration of the communication architecture [3] [9]. All these approaches rely on clock cycle accurate synchronization between the simulation and the FPGA which restricts the overall simulation performance.

A popular approach to increase simulation performance is to relax the timing synchronization between the software and the hardware simulation. A simple way to implement this idea is to omit explicit alignment of timing information and only rely on stalling while one part of the simulation depends on the other. To illustrate this concept, imagine a VP with an Instruction Set Simulator (ISS) executing target software and an FPGA attached in some way. Assume the FPGA contains the RTL description of a peripheral that will at some point be accessed by the target software. From the point of view of the target software it will just access a memory mapped register. At this point the software simulation will be stalled until the interaction with the FPGA is completed. The ISS will get the result of the memory access as if no time had passed.

The gain in simulation performance lies in the fact that while the FPGA is calculating the result of the memory access it is not synchronized with the software simulation. This approach lets both simulation run at full speed and only stall when they interact. This approach has been utilized often in literature ([2], [4], [5], [7], [8], [10], [14], [15]).

Another important distinction in this space is whether or not the design can handle transactions that are initiated by the hardware simulation. For now only the case where data bus masters or bus initiators are in the software simulation was considered. This is a considerable disadvantage if peripherals that have a Direct Memory Access (DMA) port or even whole CPUs are to be verified. The concept of having a device on the FPGA that can write data back through a DMA port has been explored in [12] and [11]. [12] describes a mechanism that allows for data to be transferred in both directions by synchronizing hardware and software on every clock cycle. While this approach allows the desired DMA transfer, it suffers from low simulation performance because of the synchronization overhead. In [6] and [3] an entire CPU was instantiated on the FPGA to verify the interplay between target software and RTL design. [3] ran target software for a NIOS 2 softcore in a custom scheduler environment and connected SystemC to simulate other peripherals. Cycle accurate synchronization is taken care of by the scheduler that runs underneath the target software, which ensures accuracy but incurs a performance penalty as mentioned previously. The authors of [6] rely on the HLA/RTI standard to connect all simulations which implement it. The FPGA is connected by controlling the Design Under Test (DUT) with a debugging subsystem and a software layer that implements the required HLA/RTI interfaces. This approach is also cycle accurate. A different angle is approached by [16], which puts emphasis on CPU design and architecture exploration. In their paper the RTL description of the CPU is transformed in a way that allows fine grained control of the hardware simulation while every module's clock is decoupled. They also provide software adapters that transfer the logic level of peripheral outputs to their software simulation. There is also the Accelera Standard Co-Emulation Modeling Interface (SCE-MI) which standardizes a modeling interface that is supposed to enable transactor models to be easily migrated from simulation to emulation [18]. Few of the mentioned approaches implement this standard.

With SytHIL we propose a framework that allows the connection of all types of peripherals and CPUs in whichever domain, hardware or software simulation, in a transaction based and decoupled manner. Fig. 2 provides a table which gives a comparison of our work and previous publications.

III. The SytHIL Framework

Our framework consists of two major parts, first the QEMU integration into SystemC and second the SystemC-FPGA integration. The QEMU domain is handled using GreenSocs's Qbox [19]. This allows wrapping QEMU CPUs into SystemC modules that have standard Transaction Level Modeling (TLM) ports that connect to the rest of the system. It also allows non-CPU QEMU devices to be wrapped and similarly exposed as SystemC modules so that it is possible to reuse all the devices that QEMU provides. For the FPGA-SystemC integration Xilinx's libsystemctlm-soc library was extended [20]. There are two types of connections, one to transfer SystemC initiator requests to the FPGA and the other to transfer RTL initiator requests from the FPGA to SystemC. The main difference between the two is in which domain the transaction initiator lies. The basic structure of both connections is similar, there is an RTL part, which is called the RTL bridge, and a software module interacting with it. The RTL bridge is connected to the user RTL design on the FPGA and the SystemC module, that acts as a user space driver for the RTL bridge, provides a TLM interface to it. In principle, the RTL bridge acts as a mailbox that the initiator side can write into and the target side will read out of, once it is ready or as soon as all relevant information has arrived. The RTL bridge also supports the forwarding of interrupts in both directions.

Xilinx's library supports the RTL design to be simulated in software with Verilator or to run it on an FPGA that is connected via Peripheral Component Interconnect Express (PCIe) port. To also support different bus protocols in the RTL design, such as AXI, ACE, and CHI, the software has to account for protocol-specific parameters. The back-end and the protocol-specific configuration are encapsulated in different modules to increase abstraction. The same back-end is used in both connection directions. An overview of the general architecture can be seen in Fig. 3. Notice that the AWS shell is specific to the AWS setup and facilitates the conversion from PCIe to AXI. In non-AWS setups this block can be substituted by Xilinx's PCIe to AXI converter which does not pose any issues since no other features of the AWS shell are being used.

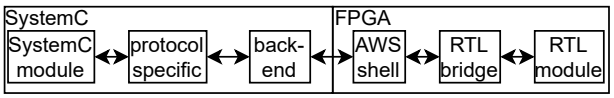

Fig. 3. FPGA to SystemC transaction state machine.

As an example the following paragraph will explain the sequence of a transaction from a functional point of view. To simplify the explanation details of the back-end interactions are neglected. The steps involved in a SystemC to FPGA transaction are depicted in Fig. 4. Generally, the transaction starts with a SystemC initiator that calls the b_transport function. The meta information of the transaction is programmed into the memory mapped registers of the RTL bridge on the FPGA. Once all information arrives, the RTL bridge carries out the AXI transaction according to the configuration and makes the result available in its registers. The software counterpart reads the results, converts them to a TLM payload, and returns them to the TLM initiator.

The FPGA to SystemC transaction works very similarly but is initiated on the FPGA. When a transaction arrives at the RTL

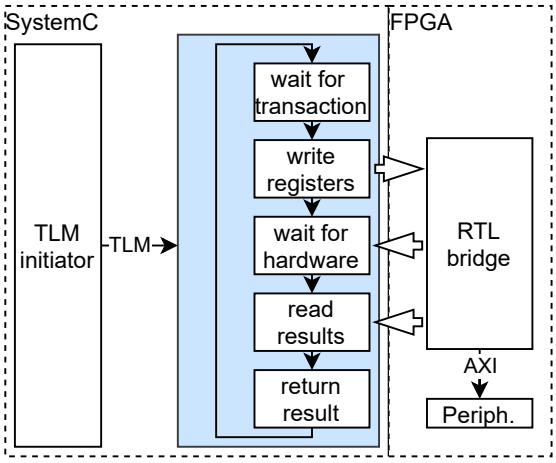

Fig. 4. SystemC to FPGA transaction state machine.

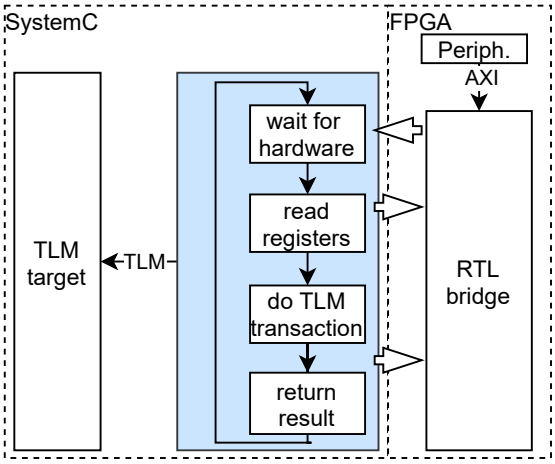

Fig. 5. FPGA to SystemC transaction state machine.

bridge, it notifies the SystemC module. This modules reads the transaction information from the RTL bridge registers and carries out the respective TLM transaction. The results are again written to the RTL bridge registers. This is shown in Fig. 5.

To increase the throughput of the SystemC to FPGA connection a mechanism that can optionally bypass all transaction related modules was implemented. The idea is to give the initiator module direct access to the FPGA through a memory pointer that is mapped to the FPGA. All accesses to that pointer result in AXI accesses on the FPGA. This pointer is initialized by the back-end by interacting with an AWS specific library and is then passed on to the initiator by using SystemC's Direct Memory Interface (DMI) mechanism. On the FPGA, Xilinx AXI interconnects are used to create two paths to the RTL peripheral. The first goes through the RTL bridge as before, while the second one bypasses the RTL bridge and directly connects to the target RTL module. A graphical representation of this setup is shown in Fig. 6. Since

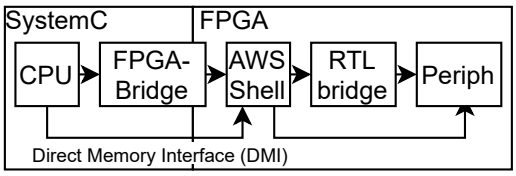

Fig. 6. AWS DMI setup.

the RTL bridges are still in the memory space the RTL target peripheral needs to be mapped in a special address range. The driving software expects the RTL bridges to occupy the first addresses up to 0x20_0000 which means the peripheral address needs to be bigger than that.

IV. Results

In this section, an overview of the experimental evaluation results is presented. An AWS f1.2xlarge instance was used as the simulation host for all benchmarks. An overview of its technical specification is provided in Table I.

First the speed of data movement between the FPGA and the SystemC framework was evaluated using different methodologies to identify the performance bottlenecks and boundaries. In the first experiment, a VP consisting of a RISC-V QEMU instance, a SystemC UART model and a memory is partitioned using our SytHIL framework. QEMU and the SystemC UART model are executed on the simulation host, while the memory is placed on the FPGA and connected using SytHIL transactors. Linux is booted on the RISC-V core running on QEMU. A benchmark program is executed in the RISC-V Linux that reads and writes data to the memory located in the FPGA.

Transfer throughput is measured against wall-clock time. A second partitioning in which the memory is placed in the SystemC domain instead of the FPGA is used for comparison. Two different methodologies for the connection between the host and the FPGA were evaluated. In the first, which is the standard approach used by e.g. [20], data is transferred via a system of letterboxes and interrupts using SystemC's blocking transport interface (b_transport). Note that both read and write operations require signaling from SystemC to the FPGA and back. Even in the case of a write transaction, the return path is used to indicate the completion and success or failure of the operation. In the second approach, which has been implemented in SytHIL, a bridge between the PCIe interface and the AXI internal bus fabric on the FPGA is used. This is implemented using the previously described DMI mechanism,

TABLE I
F1.2XLARGE INSTANCE SPECIFICATION.

CPU	Intel Xeon E5-2686 v4 (8 vCPUs)
RAM	122 GB
OS	CentOS 7
FPGA	1 Xilinx Virtex UltraScale+ VU9P
Price	1.65 USD/h

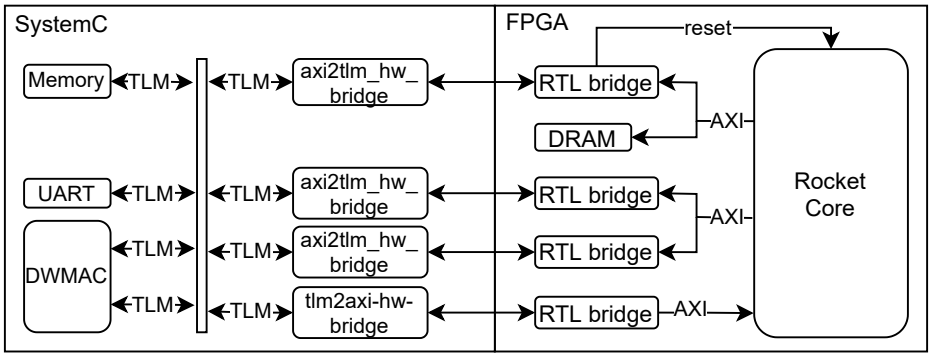

Fig. 7. Platform with Rocket Core and DWMAC.

which is part of the SystemC TLM2.0 standard, and also supported by the QEMU-based CPU model. The measured values are summarized in Table II. While the initial 'NON DMI'-based approach is slower than the standard SystemC TLM2.0 'b_transport' methodology, the DMI approach is 1 order of magnitude faster. The difference between the non-DMI and DMI approaches is stark. This improvement in bandwidth enables SytHIL to offer a more flexible distribution of components between the FPGA and SystemC environment, and it does so with no loss of simulation quality. However, when executing in 'NON DMI' mode, all accesses to memory from the FPGA domain can be traced, while this is not possible in 'DMI' mode. In terms of the DMI approach, it can be observed that writes to the FPGA memory are $4.4x$ faster than reads. This is a property of the PCIe and DRAM memory system itself. Even though the improvements are considerable, a pure SystemC platform outperforms the FPGA mechanism by a factor of 18-154 depending on the access type as moving data to and from the FPGA incurs an unavoidable overhead. One has to keep in mind that DMI is mostly used for memory accesses while SytHIL can perform DMI accesses to peripherals that update their state between transactions.

Next, the results of a more complex benchmarks are examined, specifically looking at the flexibility that the framework gives to place CPUs on the FPGA. In this case Linux boot time is used as a measure of the simulation performance. The first case consists of a VP containing a Rocket core [21] RISC-V CPU and a memory, both placed on the FPGA, and an UART in the SystemC domain, all connected using SytHIL transactors. In the second benchmark the Rocket core was replaced with the BOOM core [22] highlighting the adaptability of the framework. In both cases Linux boot time was measured. Results are shown in Table III, which indicates the boot time for the BOOM core is marginally faster than that of the Rocket core. These preliminary result of approximately 20 s compare favourably with those found by [16] who report equivalent BOOM core boot times of 3.68 minutes. It has to be taken into account that their FPGA platform (ZC706) can only run an unmodified BOOM core at 50 MHz while the AWS FPGA runs at 125 MHz.

TABLE II
BANDWIDTH BENCHMARK RESULTS

Configuration	Result
QEMU & FPGA memory NON DMI	67.1 kB/s (read/write perf.)
QEMU & FPGA memory DMI	3.796 MB/s (Read perf.) 16.6 MB/s(Write perf.)
QEMU & SystemC memory (b_transport)	0.097 MB/s(Read perf.) 0.097 MB/s(Write perf.)
SystemC memory (DMI)	276.2 MB/s(Read perf.) 276.2 MB/s(Write perf.)

TABLE III
LINUX BOOT BENCHMARK RESULTS

Configuration	Result
RocketCore VP	19.0 s (Linux boot time)
BOOM core VP	19.14 s (Linux boot time)

To test the framework in a more complex context a network benchmark was conducted. To make this possible a new platform was built that has the Rocket core on the FPGA and a Synopsys DWMAC model in SystemC. The platform is shown in Fig. 7.

The other components like UART and memory are used to interact with the platform and boot a Linux kernel that has device drivers for the DWMAC. After setting up the TAP interface that is used by the DWMAC model the simulation can be started and networking can take place between the host and the Linux running on the FPGA. iPerf3 was used to measure the network throughput in both transaction directions. The results are shown in Table IV. The throughput from the simulation host to the platform amounted to 29.6 kB/s while the other direction was measured at 9.69 kB/s.

TABLE IV
IPERF3 BENCHMARK RESULTS

Direction	Result
Host to Platform	29.6 kB/s
Platform to Host	9.69 kB/s

V. Conclusion

In this work the SytHIL framework was presented. The framework enables the integration of QEMU models, SystemC models and RTL hardware descriptions into one unified VP. Therefore, SytHIL simplifies the combined testing and verification of both the target software and the target hardware RTL descriptions. To instantiate the RTL descriptions FPGAs are used, as they are faster than RTL simulators. To exhibit the capabilities of our framework relevant use case, such as Linux boot and data transfer via network, were demonstrated and benchmarked. All measurements were carried out using AWS FPGA cloud servers as simulation hosts, as they provide an inexpensive, standardized platform that is both powerful and easily scalable.

Overall, our results show good performance, but care must be taken when designing the VP to ensure that the distribution of components between the various supports is optimal and reflects the use case. One aspect of this is moving data between the FPGA and the host which incurs an overhead. We have shown a significant improvement on data throughput, adopting the SystemC DMI mechanism, which provides more flexibility and performance.

Eventhough, the SytHIL framework is already in use industrially we plan to extend it. In future work we will improve the performance of the SytHIL framework by enabling DMI access from the FPGA to the SystemC domain. In addition, we will address timing synchronization between the three simulation domains, as this is currently not handled by SytHIL in an efficient fashion.

References

[1] S. Hassoun, M. Kudlugi, D. Pryor, and C. Selvidge, "A transaction-based unified architecture for simulation and emulation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, pp. 278–287, 2 2005.

[2] X. Zhang, F. Hui, Q. Wang, and X. Shen, "Integrated iss and fpga soc hw/sw co-verification environment design," vol. 2, 2008, pp. 1071–1075.

[3] M. B. Ayed and M. Abid, "A fast hardware/software co-verification method using a real hardware acceleration," 2012.

[4] C. Q. Li, H. C. Huang, C. Y. Xiang, A. W. Ruan, and W. Tang, "A novel methodology for hardware acceleration and emulation," in *2011 International Symposium on Integrated Circuits*, 2011, pp. 597–600.

[5] Y. B. Liao, P. Li, A. W. Ruan, Y. W. Wang, W. C. Li, and W. Li, "Hierarchy communication channel in transaction-level hardware/software co-emulation system." Institute of Electrical and Electronics Engineers Inc., 2008, pp. 94–99.

[9] Y.-I. Kim, K.-Y. Aim, H. Shim, W. Yang, Y.-S. Kwon, A. Ki, and C.-M. Kyung, "Automatic generation of software/hardware co-emulation interface for transaction-level communication," in *2005 IEEE VLSI-TSA International Symposium on VLSI Design, Automation and Test, 2005. (VLSI-TSA-DAT).*, 2005, pp. 196–199.

[6] M. G. Seok, T. G. Kim, and D. Park, "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. E100A, pp. 1374–1383, 7 2017.

[7] D. S. Brahme, S. Cox, J. Gallo, M. Glasser, W. Grundmann, C. N. Ip, W. Paulsen, J. L. Pierce, J. Rose, D. Shea, and K. Whiting, "The transaction-based verification methodology," Cadence Berkeley Labs, Tech. Rep., 2000.

[8] F. Borlenghi, D. Auras, E. M. Witte, T. Kempf, G. Ascheid, R. Leupers, and H. Meyr, "An fpga-accelerated testbed for hardware component development in mimo wireless communication systems," in *2012 International Conference on Embedded Computer Systems (SAMOS)*, 2012, pp. 278–285.

[10] S. Lee, M.-K. Jung, I.-C. Park, and C.-M. Kyung, "isave: a behavioral emulator for in-system algorithm verification," in *Proceedings of Second IEEE Asia Pacific Conference on ASICs. AP-ASIC 2000 (Cat. No.00EX434)*, 2000, pp. 303–306.

[11] H. Shim, S.-H. Lee, Y.-S. Woo, M.-K. Chung, J.-G. Lee, and C.-M. Kyung, "Cycle-accurate verification of ahb-based rtl ip with transaction-level system environment," in *2006 International Symposium on VLSI Design, Automation and Test*. IEEE, 2006, pp. 1–4.

[12] Y. Nakamura, "Software verification for system on a chip using a c/c++ simulator and fpga emulator," in *2006 International Symposium on VLSI Design, Automation and Test*, 2006, pp. 1–4.

[13] N. Kim, H. Choi, S. Lee, S. Lee, I.-C. Park, and C.-M. Kyung, "Virtual chip: making functional models work on real target systems," in *Proceedings 1998 Design and Automation Conference. 35th DAC. (Cat. No.98CH36175)*, 1998, pp. 170–173.

[14] J. Shen, T. Suh, H.-H. S. Lee, S.-L. Lu, and J. Shen, "Initial observations of hardware/software co-simulation using fpga in architecture research," 2006. [Online]. Available: https://www.researchgate.net/publication/228578868

[15] P. Schumacher, M. Mattavelli, A. Chirila-Rus, and R. Turney, "A virtual socket framework for rapid emulation of video and multimedia designs," in *2005 IEEE International Conference on Multimedia and Expo*, 2005, pp. 872–875.

[16] D. Kim, A. Izraelevitz, C. Celio, H. Kim, B. Zimmer, Y. Lee, J. Bachrach, and K. Asanovicc, "Strober: Fast and accurate sample-based energy simulation for arbitrary rtl," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 128–139.

[17] F. Bellard, "Qemu, a fast and portable dynamic translator." in *USENIX annual technical conference, FREENIX Track*, vol. 41. Califor-nia, USA, 2005, p. 46.

[18] "Standard Co-Emulation Modeling Interface (SCE-MI) Reference Manual Version 2.4," 11 2016.

[19] G. Delbergue, M. Burton, F. Konrad, B. Le Gal, and C. Jego, "QBox: an industrial solution for virtual platform simulation using QEMU and SystemC TLM-2.0," in *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, TOULOUSE, France, Jan. 2016. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01292317

[20] E. Iglesias, "libsystemctlm-soc," https://github.com/Xilinx/libsystemctlm-soc, 2017.

[21] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz *et al.*, "The rocket chip generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, 2016.

[22] C. Celio, D. A. Patterson, and K. Asanovic, "The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-167*, 2015.