gem5+RTL: A Framework to Enable RTL Models Inside a Full-System Simulator

Guillem López-Paradís
guillem.lopez@bsc.es
Barcelona Supercomputing Center
Universitat Politècnica de Catalunya
Barcelona, Spain

Adrià Armejach
adria.armejach@bsc.es
Barcelona Supercomputing Center
Universitat Politècnica de Catalunya
Barcelona, Spain

Miquel Moretó
miquel.moreto@bsc.es
Barcelona Supercomputing Center
Universitat Politècnica de Catalunya
Barcelona, Spain

ABSTRACT

In recent years there has been a surge of interest in designing custom accelerators for power-efficient high-performance computing. However, available tools to simulate low-level RTL designs often neglect the target system in which the design will operate. This hinders proper testing and debugging of functionalities, and does not allow co-designing the accelerator to obtain a balanced and efficient architecture.

In this paper, we introduce gem5+RTL, a flexible framework that enables simulation of RTL models inside a full-system software simulator. We present the framework's functionality that allows easy integration of RTL models on a simulated system-on-chip (SoC) that is able to boot Linux and run complex multi-threaded and multi-programmed workloads. We demonstrate the framework with two relevant use cases that integrate a multi-core SoC with a Performance Monitoring Unit (PMU) and the NVIDIA Deep Learning Accelerator (NVDLA), showcasing how the framework enables testing RTL model features and how it can enable co-design taking into account the entire SoC.

KEYWORDS

gem5, RTL, System-On-Chip (SoC), Accelerators, Heterogeneous computing, Simulation, Verilator, GHDL

1 INTRODUCTION

Existing Systems-on-Chip (SoCs) are becoming increasingly complex, incorporating a growing number of hardware blocks to their designs. These blocks can range from domain-specific accelerators to smaller designs that provide additional monitoring or security capabilities. Most of these blocks are developed independently to be later integrated into a number of different SoCs with different characteristics. Therefore, these hardware blocks are often tested and designed in isolation, without a comprehensive view of the target platform they are going to be integrated into.

Current tools to perform functional testing and design space exploration analysis of these hardware blocks are typically simulation-based. The main drawback is that these simulation frameworks are restricted to *testbenches* that feed the interfaces of the block in isolation. As a consequence, they do not model all the potential interactions and restrictions that may arise when new hardware is integrated into a complex SoC with a complete software stack.

Thus, there is a clear need for tools that enable testing these hardware blocks both in terms of the implemented functionality and the expected performance they will provide on an existing SoC. These tools should be able to deliver a comprehensive hardware/software ecosystem where all the main components of the SoC are present with a working software stack. In addition, it would be desirable that Register-Transfer Level (RTL) designs already implemented in Hardware Description Languages (HDLs) such as Verilog or VHDL could be integrated into the simulated system with minimal effort, namely without requiring extra porting implementation work.

In this paper, we introduce the gem5+RTL framework, a flexible infrastructure that enables easy integration of existing RTL models with the popular full-system gem5 simulator [25]. gem5+RTL enables to perform functional testing and design space exploration studies of existing RTL models on a full-system environment that models an entire SoC able to boot Linux and run complex multi-threaded workloads. This paper makes the following contributions:

- We provide a framework that enables easy integration of existing RTL hardware blocks within an SoC for full-system simulations. The framework provides an *RTLObject* class with the necessary functionality to communicate with any of the SoC components via standard gem5 *timing ports* and *packets*. This class has a clean application interface with the actual RTL model, which is converted to a *C/C++* shared library by leveraging the Verilator [34] and GHDL [17] toolflows. As a result, we support the most used HDL's in industry and academia: VHDL and Verilog/System Verilog.
- We demonstrate how to use the framework to integrate existing RTL models into a full-system simulated SoC. For this purpose, we employ an in-house Performance Monitoring Unit (PMU) design, and the NVIDIA Deep Learning Accelerator (NVDLA) accelerator. For the latter, we are able to integrate up to four NVDLA instances on the same SoC with eight cores and a multi-level cache hierarchy.
- We evaluate the main functionalities of the PMU and validate the results with the statistics obtained from the gem5 simulation, which enables early testing on a system with multiple hardware components connected to the PMU. In addition, we perform a design space exploration that integrates multiple NVDLA accelerator instances on an SoC that features different memory technologies. We find out that each NVDLA instance has to support at least 64 in-flight memory requests to perform well on the evaluated workloads; and that as the number of NVDLA instances increases, certain memory technology configurations become a bad design choice as they fail to deliver sufficient memory bandwidth.

The rest of the document is organized as follows. Section 2 provides context for our work and discusses current approaches for simulating RTL modules. Section 3 describes our framework to enable RTL models in gem5, while Section 4 presents two relevant

The final publication is available at ACM via http://dx.doi.org/10.1145/3472456.3472461

use cases based on a PMU and the NVDLA. Section 5 explains the experimental methodology, and Section 6 presents our evaluation of the proposed use cases. Finally, Section 7 describes the related work, and Section 8 summarizes our contributions and main findings.

2 MOTIVATION AND BACKGROUND

There is a growing interest in designing low-level hardware blocks based on Hardware Description Languages (HDLs) such as Verilog and VHDL. This interest comes from multiple sources, for example: (i) from domain-specific areas that demand low-power high-performance solutions; (ii) from modifications to existing hardware designs that can be acquired via licenses such as Arm-based Intellectual Property (IP) blocks; or (iii) from the myriad of proposals based on the RISC-V ISA coming from both academia and industry. In addition, given the area and power constraints in today's designs, the motivation to implement small hardware blocks in HDLs to accurately measure their area and power costs is appealing in order to understand if certain design choices are appropriate.

Therefore, there is a clear need to have tools to quickly test the functionality of these hardware blocks on a realistic target system that includes all the agents that may impact their behavior in the target System-on-Chip (SoC) design.

2.1 Existing Solutions based on Software Simulators

Among the computer architecture community, gem5 [25] is one of the most well-known full-system simulators. gem5 supports multiple ISA's such as Armv8, x86_64, and recently RISC-V; as well as in-order and out-of-order core models and multi-level cache hierarchies. It can emulate peripheral devices and boot an unmodified Linux kernel to run multi-threaded and multi-process workloads. Full-system simulators like gem5 are flexible and easy to extend, but performance estimations are not cycle-accurate as they are based on high-level models. Moreover, obtaining accurate area and power estimations is even more challenging, as tools based on event counts like McPAT [23] are the best option. In addition, gem5 is open-source and has a large community with contributions from academia and industry, and is a good starting point for early design exploration before moving onto the HDL domain.

There have been proposals to extend gem5 to use it as a pre-HDL tool in order to enable co-design of accelerators while taking into account the dynamic interactions within an SoC design. This enables to design efficient and balanced accelerator microarchitectures. In this category, gem5-Aladdin [33] and PARADE [12] propose frameworks that combine the full-system features of gem5 and accelerator models generated from C code. The former obtains these C models from a pre-RTL tool called Aladdin [32], which takes high-level language descriptions and data dependence graph representations of the accelerator as inputs, and the latter uses High-Level Synthesis (HLS) tools. These approaches go one step beyond gem5 models, but once the design of the target hardware block is polished, an HDL implementation of the design is needed.

2.2 Existing Solutions based on FPGAs

Despite good progress on fast and general FPGA-based simulators [10, 22, 29, 35], these frameworks require substantial investment in specialized FPGA boards [37], are difficult to use for architectural exploration and extremely tedious to modify; as they involve complex FPGA toolchains with long compilation times. Moreover, these frameworks usually focus on specific subsystems, since simulating entire SoCs can be prohibitive. Their use is typically restricted to the last stages of the design cycle.

To overcome the problem of making big investments in FPGA equipment, Firesim [21] proposes to take advantage of the EC2 F1 instances of Amazon Web Services (AWS) that have Xilinx FPGA's [31]. Firesim is an open-source full-system hardware simulator accelerated through FPGA's that can simulate from small hardware modules to complex multi-node systems. Besides, Firesim relies on a domain-specific language called Chisel [7] to describe the hardware design. Chisel is compiled to automatically generate HDL code in Verilog; however, this code is not readable by developers and any change to the HDL requires dominating Chisel, which hinders the use of Firesim.

2.3 Existing Solutions based on HDL Simulators

In order to bring a hardware design into reality, whether the synthesis target is an FPGA or an ASIC, an RTL model is needed due to the level of precision it expresses. It is the standard in industry and academia, providing good power and area estimations. The two main HDLs used today are VHDL and Verilog. One of the main disadvantages of RTL models is the long development time needed, added to the verification process required. Additionally, the methodology to work with RTL is tedious, usually requires closed-source tools with expensive licenses, and simulations are very slow.

To overcome the limitations of HDL simulator tools, a number of open-source simulation platforms have been proposed. Two of the most well-known within the community are Verilator [34] and GHDL [17]. GHDL directly translates VHDL into machine code, resulting in a good simulation speed. Until recently, co-simulation with GHDL was not possible and to our knowledge, it has not been interfaced with any simulator like gem5 until now. Verilator translates Verilog and System Verilog into C++, which eases integration with other simulation tools. It is actively used in the industry, even outperforming some commercial simulators [2, 34].

PAAS [24] also uses gem5 and Verilator with the focus of simulating FPGA-based accelerators, where Verilator and gem5 run independently and communicate through Inter Process Communication (IPC). This infrastructure is specially designed for FPGA-based accelerators and is less flexible for other use-cases that require frequent inter-connection with gem5-simulated blocks, e.g. adding a new cache in RTL connected to the cores of gem5 would be very difficult to simulate and incur a large communication overhead between the two simulators. Therefore, we propose gem5+RTL, an easier and cleaner interface with Verilator to integrate RTL models within the SoC, leading to better usability.

Finally, Verilator has also been integrated with Multi2Sim [15, 36], again targeting FPGA-based systems that integrate an FPGA within a multi-core system via bus-based architectures. However,

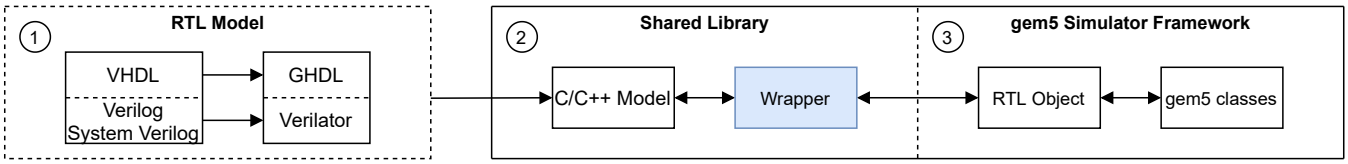


Figure 1: The gem5+RTL framework has three main blocks: 1) RTL models; 2) a shared library that includes the C++ model generated with Verilator, and 3) gem5 extensions to communicate with the shared library.

the latter proposal cannot support an OS and is restricted to only bus-based architectures. gem5+RTL overcomes the mentioned limitations, offering a simple and clean interface between gem5 and Verilator, and for the first time, between GHDL and gem5.

2.4 On-going Challenges to Simulate Hardware Designs

Multiple approaches exist to model and simulate hardware designs. From high-level language representations, to pre-HDL models that are based on a combination of high-level descriptions, and finally, the use of HDLs. The latter provide much higher levels of accuracy in terms of performance, area, and power estimations, and are starting to become widely available. However, simulating in HDL environments is tedious, and often the simulation is restricted to the actual design, missing all the interactions that exist within a SoC. This fact limits the possibilities when testing the features and capabilities of the models, and hinders co-designing with the SoC to achieve a balanced and efficient system architecture.

By combining the flexibility of a full-system software simulator like gem5 and the accurate C/C++ models provided by Verilator and GHDL, our framework enables a standard way of inserting any hardware design written in RTL (including Verilog, SystemVerilog or VHDL) into a simulated SoC with a full software stack running on top of it. The following section describes the framework in detail, which makes use of the existing gem5 *ports* and *packets* infrastructure to seamlessly allow the connection of hardware models with any SoC component simulated within gem5. This enables testing and debugging of small hardware blocks functionalities, as well as doing design space exploration performance studies that take into account a full SoC hardware/software infrastructure.

3 GEM5+RTL FRAMEWORK

Next, we introduce gem5+RTL, a flexible framework that enables simulation of RTL models inside a full-system software simulator like gem5. First, we provide a general overview of the framework (Section 3.1). Then, we describe in detail the three main components in gem5+RTL (Sections 3.2-3.4). Finally, we discuss some relevant features in the framework and provide some connectivity illustrative examples (Section 3.5).

3.1 General Overview

Figure 1 shows an overview of the gem5+RTL framework. This framework has three main building blocks:

(a) *RTL Models*: The first component consists in using Verilator or GHDL to convert an RTL model written in VHDL, Verilog or SystemVerilog into a C/C++ model. Such RTL models do not require any modification to be integrated into the SoC.

(b) *Shared Library*: Once the C/C++ model of the RTL design is ready, a wrapper to interact with it is required. This wrapper and the C/C++ RTL model are then combined into a shared library that is integrated with gem5.

(c) *Gem5 Simulator Framework*: On the gem5 side, we provide a generic framework to ease the integration of a wide range of potential hardware blocks, which may require different needs in terms of connectivity. To achieve this, we have defined a generic *RTLObject* class. This object comes bundled with all the functionality needed to interact with a simulated gem5 SoC.

The following sections describe these components in detail.

3.2 RTL Models

The main objective of gem5+RTL is to integrate unmodified RTL models into a full-system simulator. To do so, we make use of Verilator to convert an RTL model written in Verilog or SystemVerilog into a C++ model, and GHDL to convert from VHDL to a C model.

3.2.1 Verilog/System Verilog. Verilator requires to specify the top module, the intermediate compiler to be used, the language of the output model (C++ or SystemC), and the RTL files' locations. In addition, it can receive extra parameters to decrease simulation time via multi-threading, or enable features like checkpointing.

Verilator also provides usability features for the generated C++ models. For example, the possibility to output trace waveforms from the C++ RTL model, both in FST and VCD format, which is needed for debugging purposes. Since these traces can become enormous, they can be enabled and disabled from the command line or dynamically within the simulation.

3.2.2 VHDL. GHDL requires more effort in the build process and it is more difficult to replicate Verilator's behavior. It requires the top module to be a file similar to a testbench that will interact with the actual hardware block intended to be simulated. We offer a simple top file to easily adapt it to the needs of different hardware blocks. GHDL also offers the feature of generating waveforms, but these cannot be dynamically enabled/disabled at runtime.

3.3 gem5+RTL Shared Library

Once the C/C++ model of the RTL design is ready, a wrapper to interact with the generated model is required. This wrapper is similar to a *testbench*, in the sense that it has the functionality to communicate with the C++ model interfaces.

The gem5+RTL framework requires this wrapper to implement two additional simple functions: *tick* and *reset*. Both are employed by the generic gem5 infrastructure, the former to signal when a clock tick takes place and the latter to reset the state of the modeled hardware. The wrapper and the C++ RTL model are then combined into a shared library. This significantly helps with gem5 integration,

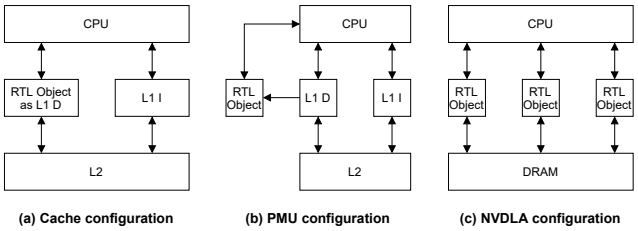


Figure 2: Different connectivity options within the SoC of a simulated gem5+RTL system.

and draws a clear separation line between what needs to be provided for each RTL model and the generic framework that we provide on the gem5 side. By having an independent shared library, gem5 can be compiled independently from Verilator. As a result, it does not require recompilation if the shared library changes, when, for example, trying new features of a new version of Verilator.

3.4 Changes to Gem5 to Support RTL Models

On the gem5 side, we provide a generic framework to ease the integration of a wide range of potential hardware blocks, which may require different needs in terms of connectivity. To achieve this, we have defined a generic *RTLObject* class. This object comes bundled with all the functionality needed to interact with a simulated gem5 SoC, including a number of predefined timing ports that enable seamless connectivity with the main hardware blocks of the SoC. Minor adjustment might be needed depending on the complexity of the hardware module to integrate. The following sections have additional details on the provided functionality.

The RTLObject class is meant to act as an abstraction layer between the RTL model defined in the shared library, and the gem5 simulated SoC. Next, we describe the main features of this class:

- A set of timing ports to seamlessly connect with other gem5 objects. The class currently supports a total of four ports, two towards the CPU side and two towards the memory side of the SoC hierarchy. This includes all the necessary functionality to send and receive packets using gem5's timing infrastructure. Adding more ports is trivial, but most hardware blocks will find the provided ones sufficient.
- Functionality to connect to a TLB object for address translation. This can be an existing object in the SoC or one specifically added to be used by the integrated RTL model.
- A tick event function that is called at the same frequency as the simulated SoC core objects. However, a parameter can be used to change the frequency with respect to the core in order to adapt to that of the integrated hardware block.
- Like all gem5 objects, it has an associated python class file used to instantiate and connect all the simulated objects within the SoC via the mentioned timing ports.

The shared library wrapper must implement two functions, *tick* and *reset*. The function *tick* from the shared library is invoked from the RTLObject side inside its own tick event. The data necessary to perform a *tick* on the RTL model side is passed as a parameter of the function via a void pointer to a predefined data structure. Similarly, the data produced on the RTL model side that is needed on the gem5 side is returned on another data structure at the end of the *tick* function. Therefore, the gem5 RTLObject and the shared library need to define these data structures and to have the necessary code to populate and consume their fields.

3.5 gem5+RTL Connectivity Examples

Figure 2 shows three examples illustrating different connectivity options with the SoC of a simulated gem5+RTL system. We enable a myriad of connectivity scenarios starting from small hardware modules such as a PMU or a cache, up to bigger IP's like an accelerator or even a small core connected to the memory hierarchy.

The gem5+RTL components have been designed to provide such flexibility in the inclusion of RTL models in the full system simulator. In this paper, we demonstrate the utility of the proposed framework by integrating two relevant use cases to a modeled SoC: a PMU model (see Figure 2 (b)) and the NVIDIA Deep Learning Accelerator (NVDLA) (see Figure 2 (c)). The next section describes the implementation of such case studies in detail.

4 GEM5+RTL USE CASES

Verification is an essential part of the process of developing a hardware design and probably the hardest and more time consuming. Different methods can be employed, but simulation-based techniques are the most popular. These methods are not as accurate as FPGA-emulated systems, but it is a convenient first step to do functional verification of a design. One of the problems when verifying a hardware block is that the validation environment is usually constrained to the block itself. Therefore, interactions that might only surface at the system level might not be stressed. For this reason, we use gem5 to model full system setups and then insert RTL models to be able to check these interactions and test functionalities. In this context, our first use case is a Performance Monitor Unit (PMU) for which we test its main functionalities and validate its results.

Our second use case focuses on using our framework to perform design-space exploration studies on the integration of a hardware accelerator into the architecture of an existing SoC. Numerous hardware accelerators are designed in isolation and can be integrated in existing SoCs. However, their requirements in terms of, for example, memory bandwidth can determine how the integration occurs or the memory technology to be employed. To this end, we use our framework to perform a design space exploration analysis to connect multiple NVDLA accelerators into a SoC. Our tool enables to determine the right memory hierarchy configuration and to evaluate the trade-offs between different memory technologies.

Finally, GHDL has been tested with a bitonic sorting accelerator written in VHDL. We have used this example to develop the support for this tool in gem5, which is very similar to the one for Verilator.

4.1 Debugging RTL Models on a Full-System Environment

We integrate an in-house PMU Verilog model with the simulated gem5 processor core. In the past, this synthesizable PMU has been integrated in the Cobham-Gaisler LEON3 [11] family of processors, which is a representative platform in space domain. It provides a configurable number of counters to monitor events, programmable event thresholds to generate interrupts, and the possibility to have multiple cores connected to the PMU. It is interfaced through the

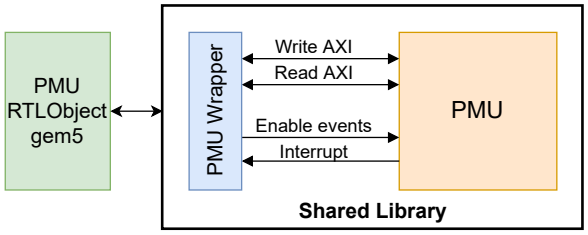


Figure 3: Internal wrapper and gem5 connections for the PMU hardware block.

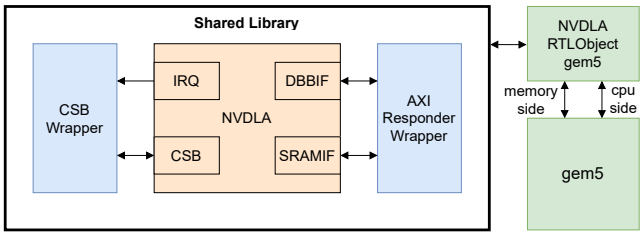


Figure 4: Internal wrapper and gem5 connections for the NVDLA hardware block.

Arm Advanced eXtensible Interface (AXI) protocol [5] for reading and writing the counters, and its configuration. The events are specified by a one-bit signal, rising up the signal on any cycle adds one to the event counter.

Figure 3 shows the necessary connections between the wrapper and the *C++* RTL model inside the shared library. The wrapper communicates with the *RTLObject* on the gem5 side via *structs* that are exchanged every tick. The *tick* function inside the wrapper receives an input struct with the necessary information to be sent to the PMU model, that is, the AXI read/write input and the event_enable[0-19] bits. Upon completion of the *tick* function, an output struct is filled so that the *RTLObject* receives the information returned on the AXI read/write channels and the interrupt signal.

To use the PMU, it first needs to be configured by enabling the events to be monitored. Additionally, thresholds for specific events can be programmed by specifying a mask and a threshold value. This is done by writing to configuration registers via AXI commands. When a threshold is reached, the PMU generates an interrupt and resets the threshold counter.

For evaluation purposes, we have connected a PMU to the commit instruction event of the core and to the L1D cache miss event. In the case of L1D misses, these can only happen once at any given cycle; however, the gem5 out-of-order core model we employ can commit up to four instructions per cycle. To properly configure the PMU, we have connected four event signals, making it possible to properly count the number of committed instructions. Finally, we have also connected the clock as a PMU event. This allows us to generate periodic interrupts by setting a threshold for this event, which is used to print the mentioned events (committed instructions and L1D misses). This use case enables to test the PMU functionalities on a comprehensive hardware/software SoC environment.

4.2 Design-Space Exploration of the SoC Integration

NVDLA is an open-source accelerator created by NVIDIA that targets frequent inference operations such as convolutions, activations, pooling, and normalization. It is a flexible and scalable hardware design that is aimed at being the standard in the community. Even though it targets embedded systems, multiple NVDLA accelerators can coexist to target systems with different requirements.

NVDLA is programmed using Verilog and different hardware configurations are available as a result of its flexibility, but only two are verified: nv_large and nv_small. The open-source release includes support for UVM verification and includes traces of real applications such as the GoogleNet and AlexNet Convolutional Neural Network (CNN) workloads.

NVDLA needs the following interfaces to connect to a SoC [26]:

- *Configuration Space Bus (CSB)* is a low bandwidth interface for the processor to configure the NVDLA accelerator.
- *External Interrupt (IRQ)* is a 1-bit signal that reports the completion of operations and errors.
- *Data BackBone (DBBIF)* is a high-bandwidth AXI4-compliant interface that is meant to be connected to a high-latency memory such as the main memory of a SoC.
- *SRAM Connection (SRAMIF)* is a secondary interface to connect with a small SRAM memory. The purpose of this small memory is to increase performance in systems that require high-performance.

Next, Figure 4 shows how the NVDLA hardware block is integrated into the gem5+RTL framework. The NVDLA release comes bundled with Verilator support, and includes wrapper classes developed by NVIDIA. These classes provide AXI and CSB interfaces that, with minor modifications, can be used as part of the shared library wrapper that communicates with the *C++* RTL model.

Originally, the AXI wrapper models an ideal external main memory, which is now replaced by sending the requests to the *RTLObject* via the output struct. Then, the *RTLObject* makes use of standard gem5 ports to send these requests to the main memory of the simulated gem5 SoC. Similarly to the PMU, there is an input struct passed as parameter to the wrapper *tick* function, with the data necessary to feed the interfaces to the NVDLA model, and an output struct with the data the RTLObject needs to communicate with the rest of the simulated SoC. We have decided to connect both memory interfaces (DBBIF and SRAMIF) to the main memory subsystem of the simulated SoC. A better solution, and a possible extension of this work, could hook a proper SRAM such as an scratchpad memory to the *SRAMIF* interface or explore other fast memories.

Additionally, in a real system, an NVDLA accelerator should be connected to the IOMMU for programmability and security reasons. We have decided to bypass the IOMMU, as gem5 support for this hardware unit is still in early stages and there is little documentation. In addition, using an IOMMU requires a compliant device driver.

5 EXPERIMENTAL METHODOLOGY

This section describes the configuration of the simulation infrastructure, the benchmarks employed, and the experiments performed to evaluate the two use cases with gem5+RTL.

[1] Obtained from a synthesis for the Xilinx FPGA KC705.

Table 1: Parameters for gem5+RTL full-system simulations.

Processor size	8 cores
Cores	3-wide issue/retire, 92-entry instruction queue, 192-entry ROB, 48 LDQ + 48 STQ, 2GHz
Private Caches	L1I: 64KB, 4-way, 2 cycle, 8MSHRs LID: 64KB, 4-way, 2 cycle, 24MSHRs L2: 256KB, 8-way, 9 cycle, 24MSHR, stride prefetcher
Last-Level Cache	16MB, 16-way, 64B lines, 8 banks, 32MSHRs per bank Data bank access latency of 20 cycles.
NoC	Coherent crossbar, 128-bit wide, 2 cycles
Main Memory	**DDR4-2400:** 2 ranks per channel, 16 banks per rank 8KB row-buffer, 128-entry write, 64-entry read buffers per channel, 18.75GB/s peak bandwidth per channel **GDDR5:** quad-channel, 16 banks/channel, 2KB row-buffer 128-entry write and 64-entry read buffers per channel 112GB/s peak bandwidth **HBM:** 8 channels, 16 banks/channel, 2KB row-buffer 128-entry write, 64-entry read buffers per channel 128GB/s peak bandwidth
PMU	Configured with 20 32-bit counters, 1GHz, 5k LUTs[1]
NVDLA	nv_full config: 2048 8-bit MACs, 512 KiB buffer, 1GHz, 2M LUTs[1]

5.1 gem5+RTL Configuration

The gem5+RTL framework has been configured to model an Arm-based SoC full-system environment that models the application, the operating system, and the architecture in detail. The simulated system runs Ubuntu 16.04 with Linux kernel version 4.15. We simulate a contemporary SoC with eight out-of-order cores and a detailed multi-level memory hierarchy. The simulator is extended with the required RTLObject class to incorporate different RTL models. Table 1 details the architectural parameters. Note that, for the NVDLA use case, we will evaluate three different memory technologies.

5.2 Evaluated Benchmarks

5.2.1 PMU Case Study. We have created a simple benchmark composed of three sorting algorithms that exhibit different computational patterns: QuickSort [20], SelectionSort [18], and BubbleSort[6]. We execute each of these algorithms one after the other.

Between each of them, we insert a 1-millisecond `sleep` call to easily identify the end of each application phase when reading the performance counters. We configure the PMU to generate interrupts every 10,000 cycles. When these interrupts occur, the monitored events are dumped. We monitor events that allow us to extract instructions per cycle (IPC) and misses per kilo instruction (MPKI) metrics, which we then compare with the ones gem5 outputs in its statistics over the same interval.

We also evaluate the simulation time overhead when adding the evaluated PMU model to gem5. The reported execution times are the average over three simulations. We run our sorting benchmark on the standalone gem5, using gem5 and the PMU model, and gem5+PMU with waveform tracing enabled.

5.2.2 NVDLA Case Study. We program the accelerators by running a simple user-level application on the simulated SoC host cores. This application loads an NVDLA trace into main memory, containing instructions and data, and then signals the accelerator to start execution and waits until the accelerator finishes. We evaluate two application traces provided by NVIDIA: (i) a small memory-intensive convolution, *sanity3*; and (ii) the second convolution of the GoogleNet CNN pipeline, which has more computations and uses 3×3 filters.

5.3 Performed Experiments

5.3.1 PMU Case Study. We have evaluated the functional behavior of the PMU with the sorting benchmark mentioned above. We gather the PMU counter values every 10,000 cycles and plot IPC and MPKI through time compared to gem5 statistics. We also evaluate the simulation time overhead when adding the evaluated PMU RTL model to gem5. The reported execution times are the average over 3 simulations. We run our sorting benchmark on the standalone gem5 simulated SoC (*gem5*), using gem5 and the PMU RTL model (*gem5+PMU*), and *gem5+PMU* with waveform tracing enabled (gem5+PMU+waveform).

5.3.2 NVDLA Case Study. We then perform a design space exploration study with a focus on the memory hierarchy. To this end, we run an experimental campaign modifying the next parameters:

- *Maximum inflight requests*: The maximum number of permitted in-flight memory requests from an NVDLA.
- *Main memory technologies*: We have chosen DDR4, GDDR5, and HBM, being the most relevant nowadays.
- *Memory channels*: Number of memory channels for DDR4. We use a fixed quad-channel configuration for GDDR5, and for HBM a fixed eight-channel per stack.
- *Number of NVDLA instances*: We have evaluated SoC systems that integrate 1, 2 and 4 NVDLA accelerators. When multiple NVDLAs are present in the system, each of them executes a separate instance of the same workload.

The goal is to determine the right amount of memory bandwidth necessary to feed a particular number of NVDLA accelerators and to see the trade-offs each of the memory technologies offer.

Finally, we also evaluate the simulation time overheads for this use case. In this case we measure the execution time when running Verilator simulations with the provided NVDLA wrapper (*verilator*), gem5 with NVDLA using a perfect memory model (*gem5+NVDLA+perfect-memory*), and gem5 with NVDLA and the DDR4 memory (*gem5+NVDLA+DDR4*).

6 EVALUATION

In this section we describe the evaluation performed using the gem5+RTL framework. First we evaluate the PMU functionalities, and then perform a design-space exploration study using the NVDLA.

6.1 PMU Functional Evaluation

As described in Section 5, we evaluate the PMU functionality by registering a number of hardware events: (i) committed instructions, (ii) L1D cache misses, and (iii) cycles. Every 10,000 cycles, we dump and compare the values registered in the PMU with the ones from gem5, shown in Figure 5.

Figure 5 shows IPC (y-axis) across time (x-axis) for a benchmark that executes three sorting kernels. The solid line represents the measurements done by the PMU, while the dotted line represents those recoded by gem5 statistics. As can be seen in the figure, both measurements are reporting the same IPC values. However, looking at the data, we observe minor differences because of two reasons:

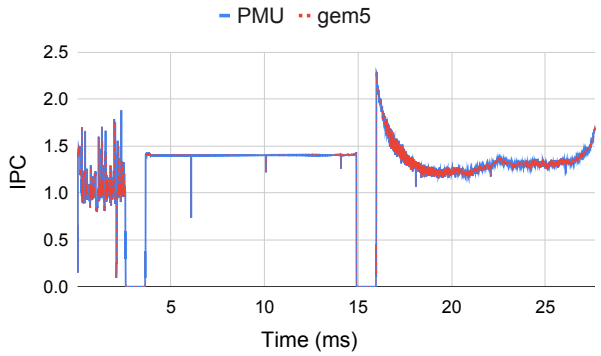


Figure 5: IPC measurements over time (ms) for the PMU and gem5 statistics on three sorting kernels separated by 1ms sleep, easily observable by a region of IPC to 0.

(i) due to the 1-cycle delay of the PMU to record the events, and (ii) because after an interrupt a reset takes place, which means some events are lost during the few cycles the reset takes.

While these discrepancies are negligible and no visible differences are observed in the figure, gem5+RTL enabled us to study these interactions and determine the exact number of events lost due to the reset process. This is useful information for the PMU designers, who may take some action if the reset process hinders the target resolution of the PMU unit.

Looking at the actual IPC curves we can easily distinguish the three sorting algorithms in the plot, as there is a 1*ms* sleep call between them, visible with an IPC of 0. QuickSort [20] is clearly faster compared to SelectionSort [18] and BubbleSort[6], taking a fraction of the time to sort 10× more elements. Additionally, we have performed the same experiment on a Fujitsu A64FX [28] node using perf, and we have observed the same curves for the three kernels, but with different IPC values as our gem5 configuration is not configured or meant to mimic the A64FX architecture.

Finally, Table 2 shows the simulation time overhead of gem5+RTL normalized to a gem5 execution without the attached PMU RTL model over three different array sizes (3, 30 and 60 thousand elements). We show results for a simulation that includes the PMU RTL model (*gem5+PMU*), and one that in addition has the waveform tracing support enabled (*gem5+PMU+waveform*).

The overheads when adding the PMU model are of up to 1.24× without waveform support. gem5+RTL is able to integrate the generated cycle-accurate *C++* RTL model and simulate it with manageable overhead. This result is expected as the PMU RTL model is a relatively small hardware block with limited interaction with the whole SoC design. However, the overhead significantly increases to up to 7.27× when waveforms are enabled. Therefore, the use of waveforms needs to be minimized to avoid hefty simulation times.

6.2 NVDLA Design Space Exploration

Next, we evaluate the different trade-offs when integrating the NVDLA accelerator into an exiting SoC. To do this, we use two different workloads: *Sanity3* and *GoogleNet*. As described in Section 5, the parameters of the study include: (i) the number of allowed in-flight memory requests for each NVDLA accelerator, (ii) the number of NVDLA instances integrated in the SoC, and (iii) the main

Table 2: Simulation time overhead when using gem5 and the PMU RTL model (*gem5+PMU*) and with waveform tracing enabled (gem5+PMU+waveform) normalized to a gem5 execution without PMU. Simulations with three different array sizes (3, 30 and 60 thousand elements).

	Size		
Configs	3k	30k	60k
gem5+PMU	1.09	1.18	1.24
gem5+PMU+waveform	3.16	6.44	7.27

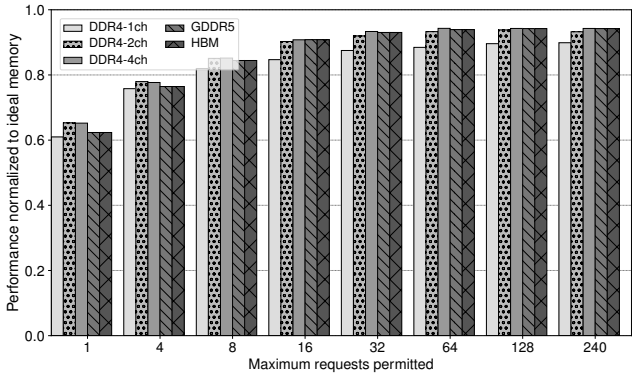
memory technology employed in the SoC. The number of allowed in-flight requests ranges from 1 to 240, the number of NVDLA instances from 1 to 4, and the different main memory configurations are: DDR4 with 1, 2, and 4 channels, a four-channel GDDR5, and an HBM stack. These memory configurations offer different memory bandwidth capabilities, as described in Table 1.

Figures 6 and 7 show the performance results normalized to an ideal 1-cycle main memory with 1 (a), 2 (b), and 4 (c) NVDLA accelerators for *GoogleNet* and *Sanity3*, respectively. Values close to one represent scenarios where memory contention is not affecting the performance of the NVDLA accelerators.

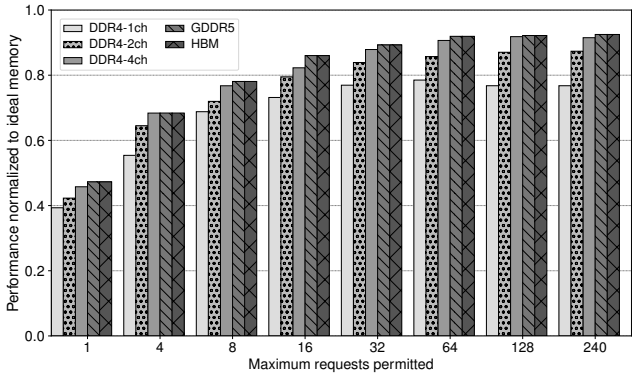
As can be seen in *GoogleNet*, when employing one NVDLA accelerator (Figure 6 (a)) all memory technologies perform similarly, providing sufficient bandwidth to feed the accelerator. The only exception is DDR4-1ch, which falls a bit behind with 16 or more maximum in-flight requests. In *Sanity3* this observation holds as the performance drops significantly with DDR4-1ch (see Figure 7 (a)). Even the DDR4-2ch and DDR4-4ch setups fail to deliver comparable performance with respect to the GDDR5 and HBM configurations for 16 and 32 maximum in-flight requests. Nonetheless, if a maximum of 240 in-flight requests are allowed, the DDR4-2ch configuration offers competitive performance, and a system with such a memory configuration should be enough to host one NVDLA.

When integrating two NVDLA accelerators (Figures 6 and 7 (b)), allowing at least 64 in-flight memory requests is a mandatory requirement to avoid significant performance degradation. The *GoogleNet* benchmark requires at least DDR4-4ch to attain the same performance as the high-bandwidth memory configurations. In the case of *Sanity3*, even with DDR4-4ch there is a noticeable performance degradation with respect to GDDR5 and HBM. In this case, SoC designers need to decide whether the performance increase provided by GDDR5 and HBM memory technologies is worth the cost with respect to DDR4-4ch. gem5+RTL helps SoC designers take informed design decisions by evaluating an existing hardware RTL model on a full-system environment that would be very hard to replicate in existing simulation-based testing environments. To put things into perspective, the Jetson AGX Xavier Developer Kit includes 2 NVDLA accelerators and has 137GB/s of peak memory bandwidth [27]. The DDR4-4ch, GRRD5, and HBM configurations have 74.96GB/s, 112GB/s, and 128GB/s respectively.

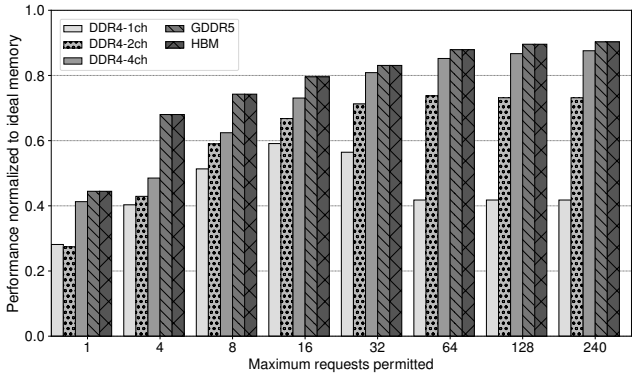
Finally, with four NVDLA accelerators (Figures 6 and 7 (c)), the number of maximum requests need to be 240 to avoid performance degradation on the *Sanity3* benchmark. In this scenario, the use of high-bandwidth memory is recommended as DDR4 fails to deliver



(a) Performance of the system with a single NVDLA accelerator.

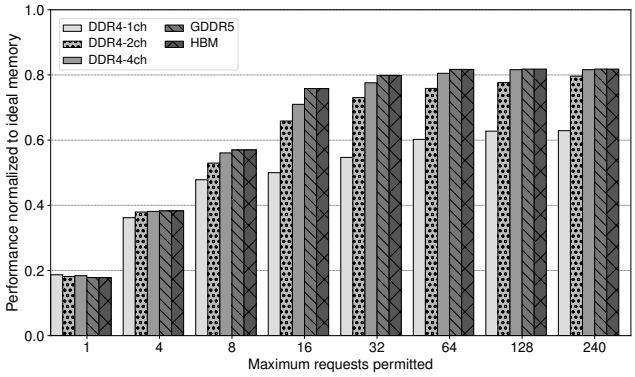


(b) Performance of the system with two NVDLA accelerators.

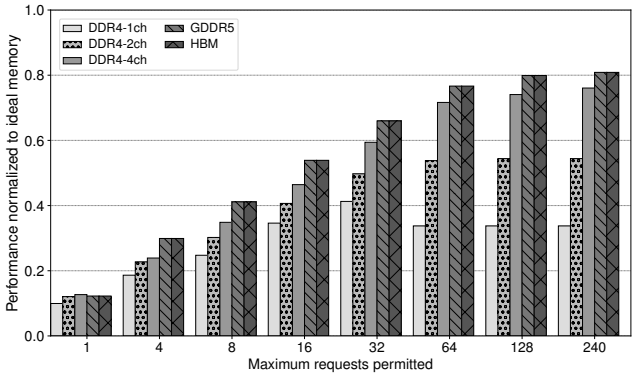


(c) Performance of the system with four NVDLA accelerators.

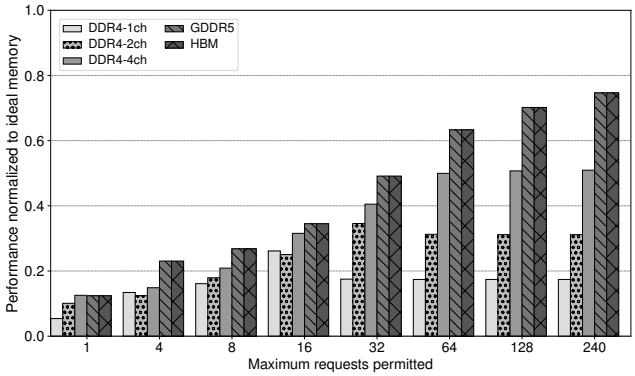
Figure 6: Design-space exploration using the GoogleNet benchmark. Normalized to an ideal 1-cycle main memory.



(a) Performance of the system with a single NVDLA accelerator.



(b) Performance of the system with two NVDLA accelerators.



(c) Performance of the system with four NVDLA accelerators.

Figure 7: Design-space exploration using the Sanity3 benchmark. Normalized to an ideal 1-cycle main memory.

competitive performance. Even the GDDR5 and HBM technologies see a performance drop with respect to the 2 NVDLA accelerators from 0.81 to 0.74. Therefore, architecting an SoC with four NVDLA accelerators requires higher bandwidth technologies such as HBM2 in order to fully utilize the available hardware.

We observe performance degradation when increasing the number of memory requests that the NVDLA can send with the DDR4-1ch memory configuration. This behavior can be seen in Figures 6 (c), 7 (b), and 7 (c). We have confirmed that this is due to severe memory contention on the memory controller side. This is another example of the insights we can gain when using the gem5+RTL framework.

Next, Table 3 shows the simulation time overhead of gem5+RTL normalized to a standalone Verilator simulation employing the wrapper that NVIDIA provides [1], namely *nvdla.cpp*[2]. This is compared to two gem5 simulated systems that integrate a single NVDLA accelerator, one simulating an ideal memory like the one used as a baseline in the design-space exploration evaluation

[2]We have modified this C++ class to have less debug printing and to disable waveform tracing, significantly improving simulation time.

Table 3: Simulation time overhead of gem5+RTL normalized to a standalone Verilator simulation with a single NVDLA accelerator. *gem5+NVDLA+perfect* has an ideal memory, while *gem5+NVDLA+DDR4* uses the DDR4-4ch configuration.

	Sanity3	GoogleNet
gem5+NVDLA+perfect-memory	2.67	1.49
gem5+NVDLA+DDR4	3.12	1.54

(*gem5+NVDLA+perfect-memory*), and another one using the DDR4-4ch memory configuration (*gem5+NVDLA+DDR4*).

Compared to the standalone Verilator execution, gem5+RTL with DDR4-4ch takes up to 3.12× more time with *Sanity3*. It has a higher overhead because the actual benchmark runtime is shorter, which makes the portion of time devoted to loading the trace into memory more prevalent. The standalone Verilator execution does not have this step as it reads the trace directly. Therefore, the simulation time overhead with larger benchmarks like *GoogleNet* is actually reduced to just 1.54×. These simulation time overheads are expected as a full-system SoC with a DDR4 main memory model is now simulated.

Thus, gem5+RTL enables design-space exploration studies that can provide valuable insight when integrating hardware blocks into an SoC with reduced simulation time overheads.

7 RELATED WORK

Evaluation of RTL models can be done using different tools and at different abstraction levels. From flexible software models in C++ that do not offer cycle accuracy but can obtain reasonable performance estimations, to HDL simulators that model the behavior of a given RTL model, very useful to discover functional bugs, and up to full SoC emulation in a FPGA that can give a good understanding of the overall performance of the system. Each of these solutions can be used depending on the step of the hardware design cycle, and sometimes are done in parallel if the group is big enough.

All the mentioned approaches are useful offering advantages and disadvantages. Software models can be of big interest to easily obtain performance numbers with big flexibility but since they are not modeling RTL, the performance numbers need to be validated and taken cautiously. RTL models are inevitably needed to create the final ASIC or FPGA solution, with slow simulation times but very useful to find functional bugs. Finally, FPGA emulation is time consuming but offers good accuracy compared to others solutions, and is a mandatory step in any design that targets an ASIC or FPGA.

7.1 Software Architectural Simulation

Software simulators offer great flexibility to model from small to full-systems OS-capable SoCs. Written in high-level languages like C++, usually achieve simulation speeds of a few kilo instructions per second (KIPS). Numerous simulators have been proposed that provide different levels of accuracy and simulation speed.

COTSon [4] is a full-system simulator decoupling functional and timing simulation. Functional simulation relies on just-in-time compilation of the simulated program. COTSon features several levels of detail and supports sampling. In addition to performance, ESESC [3] also includes models for power consumption and thermal behaviour. ESESC is the first simulator applying time-based sampling to simulation of multi-threaded applications.

Simulators base on dynamic binary translation are notorious for their higher simulation speeds, at the expense of full-system support, and are usually restricted to the ISA they execute on. *Sniper* [9] belongs to this category and features a purely analytic CPU model. Instead of modelling micro-architectural structures within the CPU, it employs the mechanistic *Interval Simulation* model [16]. Zsim [30] also uses dynamic binary translation, but with a novel parallelization technique called bound-weave that enables simulations of hundreds of cores. In addition, it provides lightweight user-level virtualization to support certain complex workloads that are usually restricted to full-system simulators.

For large-scale multi-node simulations of thousands of cores, the level of abstraction needs to be raised to make simulations feasible. In this context, MUSA [19] presents an end-to-end methodology that combines different levels of abstraction. MUSA is able to model the communication network, microarchitectural details, and system software interactions. Simulations are based on multiple levels of tracing, which difficults the integration of additional hardware.

Finally, the full-system simulator *gem5* [8] features core models with different levels of detail, ranging from a functional model to a detailed superscalar OoO core. Besides others, gem5 supports the x86 and ARM architectures, which are the most prevalent architectures today. We chose gem5 to implement our framework as it has all the necessary key features to interface hardware blocks into a simulated SoC that can run a full software stack. In addition, it has a big and growing community from both industry and academia.

7.2 HDL Simulators

Software simulation of RTL models is a necessary step for verification purposes. It is usually slow, but offers high levels of accuracy that are instrumental to detect functional bugs. Several HDL simulators exist some with commercial and others with free-to-use licenses. To the best of our knowledge, *GHDL* has not been interfaced with any simulator such as gem5.

Verilator is an HDL simulator that is being used in industry and academia environments. In academia, proposal like PAAS [24] and HetersoSim [15] use Verilator focusing on the interactions of FPGAs with the SoC. We differentiate from these simulators by targeting a wider range of RTL Models, and by focusing on systems that are not FPGA centric. In addition, gem5+RTL offers a tightly coupled connection between the SoC and the RTL model, while PAAS relies on inter-process communication. Table 4 compares these relevant proposals with gem5+RTL in terms of features to highlight the main differences.

In industry, there are existing commercial HDL simulators from Cadence, Synopsis, and Mento Graphics that can also be interconnected with other full-system simulators. We chose Verilator because it is open-source and offers a good level of performance when compared to these commercial solutions. In fact, companies like Tesla have recently decided to use Verilator internally [2].

Table 4: Related work comparison.

Features	HeteroSim	PAAS	gem5+RTL
Verilog/SystemVerilog	✓	✓	✓
VHDL	✗	✗	✓
OS Support	✗	✓	✓
System software Support	✗	✓	✓
Not FPGA-centric	✗	✗	✓

7.3 FPGA-accelerated solutions

FPGA-accelerated solutions are of great interest in the last stages of the design cycle for hardware components. However, these environments are less flexible than software solutions and require the entire design to be in RTL. For entire SoC simulations this can be a challenging platform.

Our framework does not target an FPGA-accelerated environment such as Firesim [21]. FireSim has also been integrated with the NVDLA accelerator [14], which requires editing the RTL code of the SoC. gem5+RTL is less accurate at the entire SoC level, but in exchange it offers more flexibility with a generic full-system simulator that is easier to modify and work with. FireSim is also fast in term of simulation time; however, we have seen that Verilator offers great support to speed-up simulations [13], and our simulation times have been manageable for all our use cases.

On the commercial side, there are existing tools that offers FPGA emulation with the capability to emulate entire SoC, such as Cadence Palladium, Mentor Veloce, and Synopsys Zebu. These platforms are too expensive for regular groups in academia and are typically only available on commercial companies.

8 CONCLUSIONS

In this paper, we have introduced a flexible framework that enables the simulation of RTL models inside a full-system software simulator, gem5. The proposed framework enables the integration of VHDL, Verilog and SystemVerilog models anywhere on the simulated SoC design, which can boot Linux and run complex workloads. As a result, we can evaluate two relevant use cases that interface a PMU and the NVDLA accelerator into a multicore SoC. gem5+RTL enables to test an RTL's model features and perform early-stage design space exploration studies of the entire SoC design.

We have used gem5+RTL to test the PMU functionalities and study potential interactions that arise from interfacing with an SoC. We have been able to observe and quantify small event count discrepancies, that the reset functionality of the PMU introduces with respect to the measurements performed by gem5. In addition, we have evaluated the integration of up to four NVDLA accelerator instances, finding that certain memory technologies might not deliver sufficient bandwidth to feed the accelerators. gem5+RTL helps SoC designers take informed design decisions by evaluating RTL models on a full-system environment that would be very hard to replicate in existing simulation-based testing environments.

The gem5+RTL framework is open-source and can be downloaded at `https://gitlab.bsc.es/glopez/gem5-rtl`

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive feedback. This research was supported by the European Union Regional Development Fund within the framework of the ERDF Operational Program of Catalonia 2014-2020 with a grant of 50% of total cost eligible under the DRAC project [001-P- 001723], by the Spanish goverment (grant RTI2018-095094- B-C21 CONSENT), by the Spanish Ministry of Science and Innovation (contracts PID2019-107255GB-C21/AEI/10.13039/501100011033) and by the Catalan Government (contracts 2017-SGR-1414, 2017-SGR-705). This work has also been supported by the European Community's Horizon 2020 Framework Programme under the Mont-Blanc 2020 and EPI projects (grant agreements n. 779877 and n. 826647); and by the Arm-BSC Center of Excellence. G. López-Paradís has been partially supported by the Agency for Management of University and Research Grants (AGAUR) of the Government of Catalonia under Ajuts per a la contractació de personal investigador novell fellowship No. 2021FI B00994. A. Armejach has been partially supported by the Spanish Ministry of Economy, Industry and Competitiveness under Juan de la Cierva postdoctoral fellowship number IJCI-2017-33945. M. Moretó has been partially supported by the Spanish Ministry of Economy, Industry and Competitiveness under Ramón y Cajal fellowship No. RYC-2016-21104.

REFERENCES

[1] 2018. NVDLA Github webpage. (2018). https://github.com/nvdla/hw
[2] Anandtech. 2019. Hot Chips 31: Tesla Solution for Full Self Driving Car. (2019). https://www.anandtech.com/show/14766/hot-chips-31-live-blogs-tesla-solution-for-full-self-driving
[3] Ehsan K. Ardestani and Jose Renau. 2013. ESESC: A fast multicore simulator using Time-Based Sampling. In *High Performance Computer Architecture, 2013 IEEE 19th International Symposium on*. 448–459.
[4] Eduardo Argollo, Ayose Falcón, Paolo Faraboschi, Matteo Monchiero, and Daniel Ortega. 2009. COTSon: infrastructure for full system simulation. *ACM SIGOPS Operating Systems Review* 43, 1 (2009), 52–61.
[5] ARM. 2011. AMBA AXI and ACE Protocol Specification. (2011). https://static.docs.arm.com/ihi0022/g/IHI0022G_amba_axi_protocol_spec.pdf
[6] Owen Astrachan. 2003. Bubble Sort: An Archaeological Algorithmic Analysis. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education* (Reno, Navada, USA).
[7] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović. 2012. Chisel: Constructing hardware in a Scala embedded language. In *DAC Design Automation Conference 2012*.
[8] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (Aug. 2011), 1–7.
[9] Trevor E Carlson, Wim Heirman, and Lieven Eeckhout. 2011. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *High Performance Computing, Networking, Storage and Analysis, 2011 International Conference for*. 1–12.
[10] D. Chiou, D. Sunwoo, J. Kim, N. A. Patil, W. Reinhart, D. E. Johnson, J. Keefe, and H. Angepat. 2007. FPGA-Accelerated Simulation Technologies (FAST): Fast, Full-System, Cycle-Accurate Simulators. In *MICRO*.
[11] Cobham-Gaisler. 2006. LEON3FT Fault-tolerant processor. (2006). https://www.gaisler.com/index.php/products/processors/leon3ft
[12] J. Cong, Z. Fang, M. Gill, and G. Reinman. 2015. PARADE: A cycle-accurate full-system simulation Platform for Accelerator-Rich Architectural Design and Exploration. In *ICCAD*.
[13] Embecosm. 2019. High Performance SoC Modeling with Verilator. (2019). https://www.embecosm.com/appnotes/ean6/embecosm-or1k-verilator-tutorial-ean6-issue-1.html
[14] Farzad Farshchi, Qijing Huang, and Heechul Yun. 2019. Integrating nvidia deep learning accelerator (nvdla) with risc-v soc on firesim. *arXiv preprint arXiv:1903.06495* (2019).
[15] L. Feng, H. Liang, S. Sinha, and W. Zhang. 2017. HeteroSim: A Heterogeneous CPU-FPGA Simulator. *IEEE Comput. Archit. Lett.* (Jan 2017).

[16] Davy Genbrugge, Stijn Eyerman, and Lieven Eeckhout. 2010. Interval simulation: Raising the level of abstraction in architectural simulation. In *High Performance Computer Architecture, 2010 IEEE 16th International Symposium on*. 1–12.

[17] Tristan Gingold. 2007. Ghdl. (2007). http://ghdl.free.fr/

[18] Martin A. Goetz. 1963. Internal and Tape Sorting Using the Replacement-Selection Technique. (1963), 6.

[19] T. Grass, C. Allande, A. Armejach, A. Rico, E. Ayguade, J. Labarta, M. Valero, M. Casas, and M. Moreto. 2016. MUSA: A Multi-level Simulation Approach for Next-Generation HPC Machines. In *SC '16*. 526–537.

[20] C. A. R. Hoare. 1962. Quicksort. *Comput. J.* 5, 1 (01 1962), 10–16.

[21] S. Karandikar, H. Mao, D. Kim, D. Biancolin, A. Amid, and D. Lee et al. 2018. FireSim: FPGA-Accelerated Cycle-Exact Scale-Out System Simulation in the Public Cloud. In *45th ISCA*. 29–42.

[22] A. Khan, M. Vijayaraghavan, S. Boyd-Wickizer, and Arvind. 2012. Fast and cycle-accurate modeling of a multicore processor. In *ISPASS*.

[23] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. 2009. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In *MICRO '09*.

[24] T. Liang, L. Feng, S. Sinha, and W. Zhang. 2007. PAAS: A system level simulator for heterogeneous computing architectures. In *FPL*.

[25] Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, et al. 2020. The gem5 Simulator: Version 20.0+. *CoRR* abs/2007.03152 (2020).

[26] NVIDIA. 2018. Integrator's Manual. (2018). http://nvdla.org/hw/v1/integration_guide.html

[27] NVIDIA. 2019. Jetson AGX Xavier and the new era of autonomous machines. (2019). http://info.nvidia.com/rs/156-OFN-742/images/Jetson_AGX_Xavier_New_Era_Autonomous_Machines.pdf

[28] Tetsuya Odajima, Yuetsu Kodama, Miwako Tsuji, Motohiko Matsuda, Yutaka Maruyama, and Mitsuhisa Sato. 2020. Preliminary Performance Evaluation of the Fujitsu A64FX Using HPC Applications. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. 523–530.

[29] M. Pellauer, M. Adler, M. Kinsy, A. Parashar, and J. Emer. 2011. HAsim: FPGA-based high-detail multicore simulation using time-division multiplexing. In *17th HPCA*. 406–417.

[30] Daniel Sanchez and Christos Kozyrakis. 2013. ZSim: fast and accurate microarchitectural simulation of thousand-core systems. In *ACM SIGARCH Computer Architecture News*, Vol. 41. 475–486.

[31] Amazon Web Services. 2016. Amazon EC2 F1 instances. (2016). https://aws.amazon.com/ec2/instance-types/f1/

[32] Yakun Sophia Shao, Brandon Reagen, Gu-Yeon Wei, and David Brooks. 2014. Aladdin: A Pre-RTL, Power-Performance Accelerator Simulator Enabling Large Design Space Exploration of Customized Architectures. In *41st ISCA*.

[33] Y. S. Shao, S. L. Xi, V. Srinivasan, G. Wei, and D. Brooks. 2012. Co-designing accelerators and SoC interfaces using gem5-Aladdin. In *MICRO '16*.

[34] W. Snyder. 2012. Verilator the fast free verilog simulator. (2012). http://www.veripool.org

[35] Z. Tan, A. Waterman, R. Avizienis, Y. Lee, H. Cook, D. Patterson, and K. Asanovic. 2010. RAMP gold: An FPGA-based architecture simulator for multiprocessors. In *DAC*. 463–468.

[36] Rafael Ubal, Byunghyun Jang, Perhaad Mistry, Dana Schaa, and David Kaeli. 2012. Multi2Sim: A Simulation Framework for CPU-GPU Computing. In *PACT*.

[37] John Wawrzynek, David Patterson, Mark Oskin, Shih-Lien Lu, Christoforos Kozyrakis, James C Hoe, Derek Chiou, and Krste Asanovic. 2007. RAMP: Research accelerator for multiple processors. *IEEE Micro* 27, 2 (2007), 46–57.