# A novel way to efficiently simulate complex full systems incorporating hardware accelerators

Tampouratzis Nikolaos
Microprocessor and Hardware Laboratory
Technical University of Crete, Greece
ntampouratzis@isc.tuc.gr

Konstantinos Georgopoulos
Microprocessor and Hardware Laboratory
Technical University of Crete, Greece
kgeorgopoulos@isc.tuc.gr

Yannis Papaefstathiou
Synelixis Solutions
Farmakidou 10, Chalkida, Greece
ygp@synelixis.com

***Abstract*—The breakdown of Dennard scaling coupled with the persistently growing transistor counts severally increased the importance of application-specific hardware acceleration; such an approach offers significant performance and energy benefits compared to general-purpose solutions. In order to thoroughly evaluate such architectures, the designer should perform a quite extensive design space exploration so as to evaluate the trade-offs across the entire system. The design, until recently, has been predominantly done using Register Transfer Level (RTL) languages such as Verilog and VHDL, which, however, lead to a prohibitively long and costly design effort. In order to reduce the design time a wide range of both commercial and academic High-Level Synthesis (HLS) tools have emerged; most of those tools, handle hardware accelerators that are described in synthesisable SystemC. The problem today, however, is that most simulators used for evaluating the complete user applications (i.e. full-system CPU/Mem/Peripheral simulators) lack any type of SystemC accelerator support. Within this context this paper presents a novel simulation environment comprised of a generic SystemC accelerator and probably the most widely known full-system simulator (i.e. GEM5). The proposed system is the only solution supporting the very important feature of global synchronization across the integrated simulation; furthermore it has been evaluated based on two different computationally-intensive use cases and the final results demonstrate that the presented approach is orders of magnitude faster than the existing ones.**

## I. Introduction

It is known that the long hardware design cycles delayed the early development of the software, since the former had to be taped out first. A way to disrupt this dependency had to be found and it materialised in the emergence of *full-system* simulators, which allowed the engineers to describe not only the new software but also the new hardware in the same simulation environment and thus significantly accelerate the final product delivery.

This work is based on the novel interconnection of a full-system simulator with a SystemC cycle-accurate hardware accelerator device. Specifically, we introduce a novel flow that enables us to rapidly prototype synthesisable SystemC hardware accelerators in conjunction with a full-system simulator without worrying about communication and synchronisation issues. SystemC is selected because of its cycle-accurate simulation features, while it is one of the most widely used input languages for the HLS tools. In addition, the official effort for SystemC definition and promotion known as Open SystemC Initiative (OSCI), now known as Accellera [1], provides an open-source proof-of-concept simulator while it has been approved by the IEEE Standards Association [2]. The base full-system simulator utilized is GEM5 [3], since it is a fully featured system which is, also, the most widely-used open-source one. We use the full-system mode of the simulator so as to be able to simulate a complete system comprised of a number of devices and a full Operating System (OS). As a result, the user can verify his/her application in a cycle-accurate manner via whole system simulation, including memory hierarchy, caches, peripherals, etc, with full operating system interaction (e.g. scheduler, drivers etc.), thus making the simulation more realistic/accurate.

Figure 1 presents an abstract view of the GEM5 full-system simulator when coupled with a full operating system. It consists of a central bus onto which several devices can be attached, DRAM memories, caches, CPUs, etc. In order for a new accelerator to be incorporated, the designer must ensure that is connected to the bus as well as that the appropriate OS drivers have been developed, as analytically described in the following sections.

The rest of the paper is organized as follows. Section II provides a brief overview of the similar existing approaches, while Section III describes the design as well as the most significant GEM5 functionality enhancements/modifications that were performed in order to incorporate our novel SystemC accelerator. Subsequently, Section IV unfolds the comparative results of our novel approach and, finally, Section V concludes the paper.

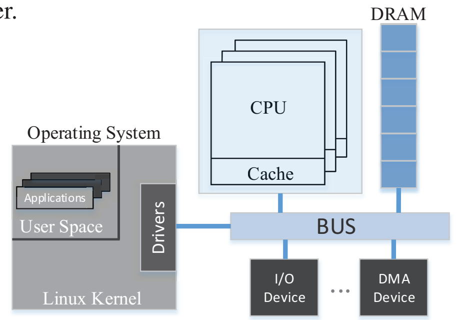

Fig. 1: Full-system GEM5 Architecture

## II. Related Work

When considering the utilization of FPGAs and Graphics Processing Units (GPUs) as application-specific accelerators, the providers of the silicon (i.e. FPGAs, and GPUs) have provided their own design and simulation tools. However, there are very few presented approaches, so far, covering the integration, in a single simulation environment, of application-specific accelerators and a full-system simulators.
The work found in [4] describes an accelerator that can be integrated alongside a multicore processor, connected to its

last-level cache, that performs natural language processing. Furthermore, the authors of [5] present Aladdin, a pre-RTL power performance simulator designed to enable rapid design space exploration for accelerator-centric systems. Both of these works use GEM5's *syscall* emulation mode which means that no operating system can be supported within their simulations. In addition, this mode does not model virtual memory, Translation Lookaside Buffers (TLBs) as well as other OS-related characteristics.

Finally, in [6] the authors present an M5-SystemC synchronisation scheme that simulates the GEM5 simulator by replacing only the GEM5 $simulate$ function with a SystemC $simulate$ function rather than support pure SystemC applications.

## III. Design

This section describes the most significant enhancements/modifications that were performed in order to expand the GEM5 full-system simulator functionality so as to support our novel SystemC accelerator. Those enhancements/modifications have culminated into the proposition of a novel design flow that can act as a guideline for the design of any accelerator. Figure 2 illustrates the integration of our SystemC simulator with the GEM5 simulator, while the following paragraphs describe the components in detail.

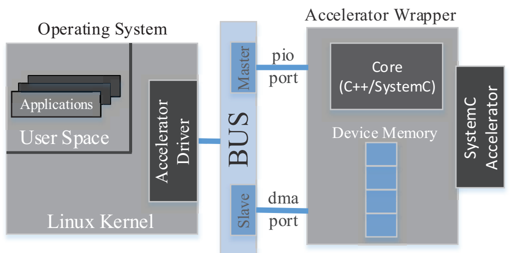

Fig. 2: Integration of SystemC accelerator with GEM5 (full-system mode)

### A. Operating System (OS)

The OS can be represented as a layered structure, as in Figure 2. It contains the User Space with the user applications and all the appropriate libraries, and the Kernel Space (in our case a Linux Kernel), which is strictly reserved for running a privileged operating system kernel and most of the device drivers. In order to incorporate efficiently our accelerator module we have developed a set of device drivers. The accelerator is activated through programmed I/Os that provide the start address and the size of the array used for the descriptors of the accelerator. The CPU can then sleep until an interprocessor interrupt from the accelerator is delivered to indicate task completion; this approach allows for full overlap of the two sub-simulations. In addition, an $ioctl$ function was developed in order to achieve efficient user-kernel space communication; this novel function can mainly perform the following tasks (additional helper commands were developed but cannot be listed in this paper due to the page limit):

- **QUERY_SET_DATA** - Initialise the Direct Memory Access (DMA) copy transaction from Host (Kernel Space) to Accelerator Wrapper.
- **QUERY_GET_DATA** - Initialise the DMA copy transaction from Accelerator Wrapper to Host (Kernel Space).
- **QUERY_CALL_DEVICE** - Call the SystemC accelerator (executing the specified application) .

Finally, an interrupt handler was implemented in order to receive appropriate interrupts from the Accelerator Wrapper, such as the SystemC accelerator finish signal, the *memcpy* finish signal, etc.

### B. Memory Bus

GEM5 provides a Memory Bus to interconnect all architecture components such as CPUs, Caches, RAMs as well as I/O devices using master and slave ports. Our novel Accelerator Wrapper device was attached to the GEM5 system using one master and one slave port. Specifically, one bus master port was connected to the peripheral I/O Accelerator Wrapper port *pio* in order to read and write into the accelerator's wrapper registers; similarly, one bus slave port was connected to the *DMA* Accelerator Wrapper port in order to write/read large amounts of data to/from the accelerator device memory, as shown in Figure 2.

### C. Accelerator Wrapper

Our Accelerator Wrapper device was developed in order to achieve efficient communication and synchronisation of GEM5 with the SystemC accelerator. It inherits all GEM5 DMA device characteristics so that full DMA transactions utilizing the full operating system can be performed. In addition, it contains a large Device Memory to store the data from the OS *memcpy*, simulating the DDR memory found in most of the real systems incorporating PCI-connected FPGA and/or GPU boards.

Subsequently, a core containing mixed C++ and SystemC code was implemented for the connection of the GEM5 C++ functions and the accelerator's SystemC threads, as shown in Figure 3. Here we must note that the GEM5 and SystemC simulators run concurrently on different threads. Specifically, when the GEM5 OS requests a SystemC accelerator call, a new thread emerges using the *pthread* library and starts the SystemC simulation by calling function $sc\_start$. Consequently, at the end of the SystemC accelerator, the new thread is killed; if GEM5 requests another SystemC call, a new thread will be created, etc.

The Accelerator Wrapper consists of, in total, eight C++ and SystemC-thread modules as described below:

1) **Dynamic Memory Allocator C++ Module** - The Buddy dynamic memory allocation algorithm scheme [7] is implemented in the Accelerator Wrapper to allocate and free Device Memory segments through the GEM5 operating system, similar to the *cudaMalloc* of NVIDIA GPUs [8].
2) **DMA Write/Read C++ Modules** - Two Direct Memory Access engines were developed so that the GEM5 dma_device can efficiently transfer high data volumes from the Linux driver to the Accelerator Wrapper and vise versa, similar to the *cudaMemcpy* of NVIDIA GPUs [8]. The user can define, through one parameter, the delay of the DMA data transfer in order to achieve a realistic latency.
3) **Synchronisation Event C++ Module** - A GEM5 synchronisation event function is implemented and it is triggered at every SystemC accelerator device

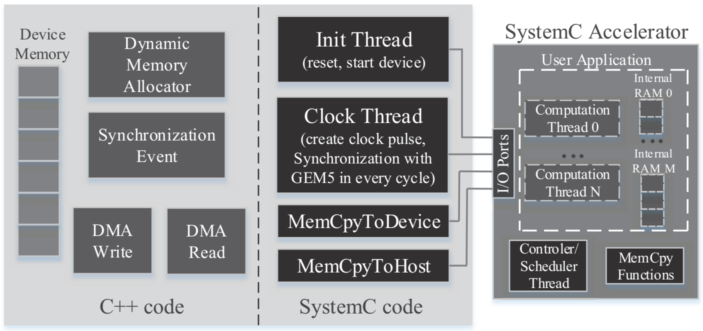

Fig. 3: Accelerator Wrapper Device

cycle. This function checks whether the SystemC accelerator has reached the next cycle. Finally, in case GEM5 is faster than the SystemC accelerator[1], it reschedules the synchronisation event function in order to wait for the SystemC accelerator.

4) **Init SystemC Thread** - The initialisation thread is implemented in SystemC so as to generate the *reset* and *start* signals both of which are essential for SystemC's module execution. Furthermore, the thread calls the *sc_pause*[2] command when the SystemC acceleration task is finished.

5) **Clock SystemC Thread** - The clock thread is implemented in SystemC in order to generate the actual *clock* signal of the accelerator when called by GEM5's OS; the DeviceClock GEM5 option is added in order to define the clock frequency (e.g. −−DeviceClock=500MHz). Moreover, at every SystemC cycle, the full synchronisation with GEM5 is achieved by checking whether the GEM5 has completed its tasks within this time frame; whenever required, the SystemC accelerator waits for the GEM5, using SystemC function *wait* with $SC\_FS$ time granularity.

6) **MemCpy ToDevice/ToHost SystemC Threads** - The Two *MemCpy* SystemC threads developed pass the data from the Wrapper Device Memory to the corresponding synthesisable I/O ports, which depend upon the data type, such as int, double, etc., so that they eventually arrive at the SystemC accelerator. The user can define through one parameter the amount of data to be read/written in one SystemC cycle.

### D. SystemC Accelerator

A reference SystemC accelerator has been developed in order to evaluate the Accelerator Wrapper and the Linux Kernel Drivers; this is also a helpful guideline for designers/users that expect to develop their own SystemC accelerators. Subsequently, the Accelera open-source libraries have been incorporated with the GEM5 SCons construction tool [9] in order to allow for the compilation and execution of complete system applications. It must be noted, however, that the user can write synthesisable SystemC before importing this to some kind of HLS tool, such as Xilinx' Vivado HLS or Cadence' CtoS. This is plausible due to the fact that only synthesisable ports are used for connecting the SystemC accelerator to the Accelerator Wrapper, Figure 3. Furthermore, the SystemC accelerator consists of a main SystemC thread and two SystemC functions described below:

- **Controller/Scheduler SystemC Thread** - This is the main SystemC thread which is called by GEM5's OS while the user has the ability to create as many individual cores as required so as to best serve his/her application, as represented in Figure 3 by the dashed-line threads. These threads are scheduled by the Controller/Scheduler SystemC Thread.
- **MemCpy ToDevice/ToHost SystemC functions** - Two SystemC *memcpy* functions were implemented in order to allow for the efficient communication with the Accelerator Wrapper MemCpy SystemC Threads so as to transfer data from the Wrapper Device Memory to the accelerator's memories.

## IV. Evaluation

The efficiency of our novel approach has been evaluated using two SystemC-described accelerators; the proposed method has been compared with the conventional manual approach, both illustrated in Figure 4. In general, the integrated simulation utilizing a full-system simulator, such as GEM5, and a SystemC simulator is not a trivial process mainly due to the data exchange needed between the two stand-alone simulators.

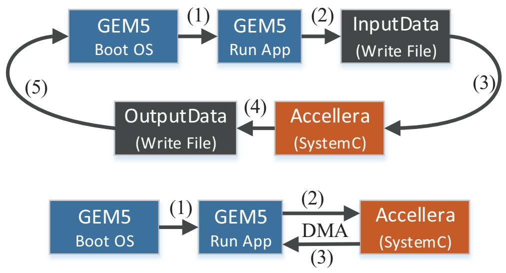

Fig. 4: (i) Standard (top) & (ii) Our novel method integrating GEM5 Full System Simulator with Accellera Simulator

Based on the existing conventional flow, the application executed in GEM5 writes the data that should be fed to the SystemC simulator in files, denoted as Steps 1 & 2 of Figure 4. Then, the full-system simulator has to terminate or stall its execution so that the SystemC simulator can read the data from those files without any synchronization problem, i.e. Step 3. Subsequently, in the conventional method, the SystemC simulator returns its results to the GEM5 application using output

[1]This can happen when GEM5 has not any processing work to do and SystemC accelerator has to manipulate a lot of threads.

[2]sc_pause command is selected instead of sc_stop because the latest destroys all SystemC structure and as a result, the same SystemC module can not be called twice.

files, denoted as Steps 4 & 5. Another important remark is that normally in every SystemC simulator call, the full-system must boot the OS from scratch, while, and more importantly, no notion of synchronisation exists in such a file-based exchange. In contrary, the proposed method resolves efficiently all those issues since the full-system simulator boots the OS only once and, since it communicates with the SystemC simulator through programmed I/Os and DMA engines, full global synchronisation is supported. Moreover, the proposed approach is orders of magnitude faster that the conventional one as demonstrated in Figure 5.

Two self-implemented use cases are used to evaluate our system based on Mutual Information (MI) [10] & Transfer Entropy (TE) [11] algorithms. The processing platform used in the comparative analysis is an Intel i7-3632QM at 2.2GHz with 8GB of RAM. Furthermore, the MI and TE results are based on a typical data size of 40K samples while the number of iterations range from 100 to 500 for MI and from 20 to 100 for TE, as shown in Table I; these ranges have been selected due to the fact that they constitute typical values in order to get high accuracy results. In addition, the results of Figure 5 focus only on the high-end value of TE algorithm (i.e. 100 iterations) since this is the worst case for our system.

Table I illustrates the overall simulation time when calling 10 times the SystemC accelerator (i.e. our MI and TE code executed on top of the Accellera Simulator). It becomes apparent that the proposed method is one order of magnitude faster than the existing conventional file-based method in all cases and that is because the GEM5 simulator boots the OS only once. Furthermore, the overall simulation time does not radically change in this case since the simulation of our very small SystemC-based accelerators is orders of magnitude faster than the actual time needed by the GEM5 simulator. Moving to more complicated accelerators (such as for example a TE module handling 100 iterations) we clearly see the efficiency of our approach since the complex code takes slightly more time than a code half or even 5 times smaller (i.e. cores handling 50 or 20 TE iterations respectively) due to our novel synchronization approach.

Finally, Figure 5 compares the overall simulation times between the conventional and the proposed method in TE algorithm (similar results are obtained for MI) using a different number of SystemC accelerator calls when the complexity of the accelerators remain constant (i.e for the same number of iterations). For the case of the first call, our method needs slightly more time due to synchronisation issues, in the order of a second, however, as the number of calls to the accelerator increases, it gradually surpasses and eventually dominates over the conventional method.

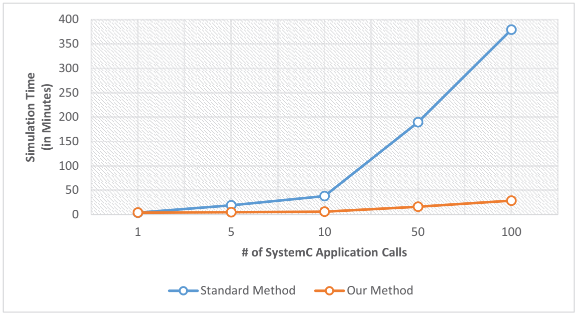

Fig. 5: Simulation time of TE using different # of Accelerator calls

| Mutual Information | | | Ttransfer Entropy | | |
|---|---|---|---|---|---|
| # of Iterations (Complexity) | Standard Method | Our Method | # of Iterations (Complexity) | Standard Method | Our Method |
| 100 | 37.3 min | 4.5 min | 20 | 37.5 min | 4.7 min |
| 200 | 37.5 min | 4.7 min | 50 | 37.8 min | 5.1 min |
| 500 | 37.9 min | 5.3 min | 100 | 38.4 min | 6 min |

TABLE I: Simulation time of (i) standard and (ii) our method using two use cases (10 # of Accelerator calls are used)

## V. Conclusions

This paper proposes a novel approach for extending a full-system simulator with the ability to efficiently and accurately simulate hardware accelerator(s) designed in SystemC. Such an integrated approach can severally reduce the time needed for design space exploration while also accelerate the overall verification process.

As a reference, we have efficiently integrated the most widely used full-system simulator (i.e. GEM5), which supports full hardware systems running complete OSs, with the most widely used SystemC simulation toolset; the novelty comes from the fact that in the proposed environment the designers can simulate their complete software applications running on top of a full operating system and utilizing one or more hardware accelerators. In addition, using our approach a wide range of heterogeneous systems can be modeled such as [12].

The presented approach supports, *for the first time*, global synchronization along the two simulation domains. Moreover, as demonstrated by our two use cases, the overall simulation is accelerated by up to an order of magnitude, when compared with the existing approach, even though the latter has no notion of synchronization.

In order to further increase the impact of this work, the complete source code (Linux drivers, Accelerator wrapper, etc.) will be freely distributed to the community.

## References

[1] "Accelera SystemC wiki website," https://en.wikipedia.org/wiki/Accellera.

[2] *IEEE 1666 Standard SystemC Language Reference*, IEEE.

[3] N. Binkert, B. Beckmann, and Blac, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[4] P. Tandon, J. Chang, and Dreslinski, "Hardware acceleration for similarity measurement in natural language processing," in *Low Power Electronics and Design (ISLPED), 2013 IEEE International Symposium on*, Sept 2013, pp. 409–414.

[5] Y. S. Shao, B. Reagen, G. Y. Wei, and D. Brooks, "Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, June 2014, pp. 97–108.

[6] M. Yu and J. S. et al., "A fast timing-accurate mpsoc hw/sw co-simulation platform based on a novel synchronization scheme," in *Lecture Notes in Engineering and Computer Science*, 2010.

[7] D. E. Knuth, *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1997.

[8] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, 1st ed. Addison-Wesley Professional, 2010.

[9] "SCons: A software construction tool official website," http://scons.org/.

[10] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, New York, NY, USA: Wiley-Interscience, 1991.

[11] T. Schreiber, "Measuring information transfer," *Phys. Rev. Lett.*, vol. 85, pp. 461–464, Jul 2000.

[12] O. T. Papaefstathiou I, Perissakis S, "Pro3: A hybrid npu architecture," *IEEE Micro*, vol. 24, no. undefined, pp. 20–33, 2004.