RAPID SYSTEM PROTOTYPING OF NETWORKING EMBEDDED SYSTEMS USING SYSTEM-C

Ioannis Papaefstathiou, Dimitrios Mprachos, Christodoulos Yannakou, Dionisios Pnevmatikatos
Department of Electronic and Computer Engineering,
Technical University of Crete
Kounoupidiana, Chania,
GREECE
{ygp, pnevmati}@mhl.tuc.gr

ABSTRACT

Current networking embedded systems are heavily heterogeneous environments involving hardware and software modules with complex communication among them. In order to prototype such systems in the most rapid manner, we need efficient methods for co-developing and, more importantly, co-simulating and co-verifying the hardware and the software modules. A very promising approach for such a development environment is the use of pure C++ for describing the system. In particular the software engineers can use C/C++ while the hardware design can be done in SystemC, a C++ class library capable of accurately modeling the hardware functionality. In this paper, we first present the efficiency of SystemC when designing Digital Signal Processors and networking CPU cores customized for media processing. We compare the design times and implementation characteristics of a 5-stage pipeline CPU core and a complicated DSP module when they are implemented in SystemC as opposed to a traditional hardware description language. Our results demonstrate that a networking and DSP processing unit can be implemented in about 40% less time in SystemC than in VHDL, while the silicon cost is used is no more than 20% higher; the performance of the hardware module is the same in both cases. Moreover, the SystemC hardware modules, communicate much faster with the software testbenches, which are commonly written in C, C++, therefore heavily accelerating the simulation time. As a result, we believe that SystemC is a very promising approach for designing prototypes of embedded systems in the most rapid way. In order to further accelerate the design process of embedded systems in general, We will release the demonstrated CPU and DSP cores as open-source, allowing anyone to use them.

KEYWORDS: SystemC, CPU, DSP, Network Processor

1 Introduction

The explosive growth of the Internet has created an insatiable demand for bandwidth. The emergence of Wavelength **Division** Multiplexing (WDM) has increased the backbone capacity to Terabits per Second, shifting the bottleneck back to the network processing units namely, routers and relevant switching equipment. As a result those units seem to be the bottlenecks of the networks today and more importantly tomorrow. Furthermore, the convergence of voice and data networks as well as the introduction of new Quality-of-Service (QoS)- based services and newly developed protocols require network flexibility that the currently employed hardware units cannot probably provide. Moreover, this widespread use of communication networks in computing systems in the last years and the implementation of complete systems on a chip has triggered the birth of a new generation of embedded systems: the networking embedded systems. Those are embedded systems with interesting wireline and/or wireless interfaces that can be implemented in a single board or chip.)

A promising solution to both problems was provided by the introduction of a new class of Embedded Integrated Circuits called Network Processing Units or NPUs.

NPUs are becoming the silicon core of every network equipment that requires a high degree of flexibility to support evolving network services at extremely high packet rates [1], [2]. Whereas legacy architectures for building networking equipment were based either on general purpose processors (GPPs), which offer high flexibility due to software programmability but poor performance, or application specific integrated circuits (ASICs), which favor speed over flexibility, the NPU approach achieves both flexibility and performance. NPU engines designed mainly for specific router/switch network cards have recently become available commercially. The majority of the commercial NPUs fall mainly into two categories: The ones that use a large number of simple RISC CPUs and those with a small number of high-end special purpose CPUs that are optimized for the processing of network streams. All network processors are system-on-chip (SoC) designs that combine processors, memory and I/O on a single chip. The processing engines in these network processors are

typically RISC cores and DSP engines, which are sometimes augmented by specialized instruction, multithreading, or zero-overhead context switching mechanisms.

In order to be able to produce such an NPU, we need efficient methods for prototyping both the software and the hardware, and more importantly for integrating the two in a very fast yet accurate manner. Such an option is offered by the recently introduced SystemC framework which facilitates HW/SW co-design and verification. The advantages of SystemC as a system level design language are as described in [1], [7], [8], [9], [11]:

1. It enables modeling of systems above the RTL level of abstraction – including systems that might be implemented in hardware, software or a combination of both.
2. Modular nature of SystemC promotes reuse of developed components from one system to another.
3. SystemC offers good design space exploration of functional specification and architectural implementation alternatives.
4. It is based on a well established programming language (C++), thus allowing its users to capitalize on the availability of extensive infrastructure of capture, compilation and debug tools.

Moreover, as it is widely supported, one way to heavily increase the design productivity, something that seems inevitable in order to exploit the constantly increasing system and silicon complexities, is the use of open-source IP cores [12]. The open-source approach seems to have several advantages: the core is very cheap (normally free), therefore a very large number of them can be used in a single SoC, the user can have source code access and there is a group of developers that provide know-how, maintain and improve the core.

In this paper we describe the design process of both a 5-stage pipeline CPU core and an advanced DSP unit for media processing, when SystemC is used. Moreover, we compare the design times of these cores with those when using a traditional hardware description language (VHDL). We concentrate on the processing units (and not on the other hardware modules), since the vast majority of the software in a SoC is executed on such processing units; by designing them using SystemC we have the extremely important advantage that the whole simulation/verification process of the hardware and the software is run in the same environment: a C++ simulator. In particular, the contributions of our paper are:

1. We demonstrated that SystemC can trigger an acceleration in the design process of a processing unit by at least 30%
2. Using the state-of-the-art design tools the cores designed with SystemC are not more expensive, in terms of silicon, that if they were designed with VHDL.
3. The co-simulation of software and hardware is heavily accelerated (by at least 300%) by using SystemC modules since everything is executed in a C development platform.
4. Both processing units are available as open-cores so as to allow their free integration in any embedded system.

2 Related Work

Even though it is widely supported that SystemC heavily accelerates the design, and especially the verification process of today's complex hardware systems, the published work comparing this relatively new design and verification approach to the traditional HDL-based ones is very limited.

The advantages of SystemC when used in hardware-software co-simulation are demonstrated from a theoretical perspective in [3][1]. However, this work does not examine what the speedup achieved, in terms of both design and simulation time is, when the SystemC methodology is employed, whereas it is not demonstrate any information regarding the silicon cost of such an approach.

The actual advantages of embedded software development using SystemC are thoroughly presented by Sirpatil *et. al.* [4]. Moreover, the advantages of hardware design using this methodology are demonstrated in detail in [5]. None of this work however presents any quantitive results regarding the efficiency of SystemC in either software or hardware development.

Also Steinert *et.al.* [6] compares the simulation speed and the synthesis results triggered when an FIR filter was designed using both SystemC and VHDL. Their results demonstrate that the SystemC design can be simulated three times faster than the equivalent VHDL one, while the synthesis results triggered, in terms of both silicon area and clock speed achieved, by both designs were very similar. Our work differs from their approach mainly because we have measured and demonstrated not only the synthesis and simulation results but also the design time needed when SystemC and VHDL are used as the design technologies. Moreover, one of the most important features of SystemC is that, since it is a C++ class, the whole simulation/verification process of the hardware and the software is run in the same environment: a C++ simulator. In order to take advantage of this feature, the hardware system that executes the software should be described in SystemC; therefore, it is more important to have the CPUs of a SoC designed using SystemC, that any other hardware sub-systems such as the FIR filter. Furthermore, in this paper we are using two different case studies covering both simple and complex CPU-based systems.

The actual design process of CPUs when SystemC is used is described in [6] and [12]. However, both papers demonstrate only the silicon area and performance achieved when only SystemC was employed for the design and implementation of those CPUs. We also present such information but more importantly we compare those results with the ones achieved when a traditional HDL is used, while we also focus on the very important nowadays acceleration in the simulation time triggered by the SystemC simulation framework.

3 Case Studies

In order to demonstrate the efficiency of SystemC design methodology in accelerating the design and simulation/verification process of today's' complex embedded systems we have used two different but similar reference scenarios.

3.1 5-stage pipeline simple CPU

The first reference design is a 5-stage Pipelined RISC CPU. In particular, it is an implementation of the DLX architecture presented in [13]. The DLX pipeline has 5 stages: IF (Instruction Fetch, ID (Instruction Decode), EX (Execution), MEM (Memory) and WB (Write Back). The first one is responsible of getting the instructions out of the program memory, the second stage is responsible of selecting the operand registers, decoding the command and reading out the registers the third for executing the actual command, while in the forth stage the possible data accesses are performed and in the fifth the results are written back to the Register File.

3.2 Open-RISC DSP

The second reference design is a Digital Signal Processor (DSP) based on the widely used open-RISC architecture described in [14]. The overall architecture of the conventional open-RISC is shown in Figure 1. It is a 5-stage pipeline Harvard architecture which in addition to the standard arithmetic operations it also supports a group of Multiply-and-Accumulate (MAC) functions.

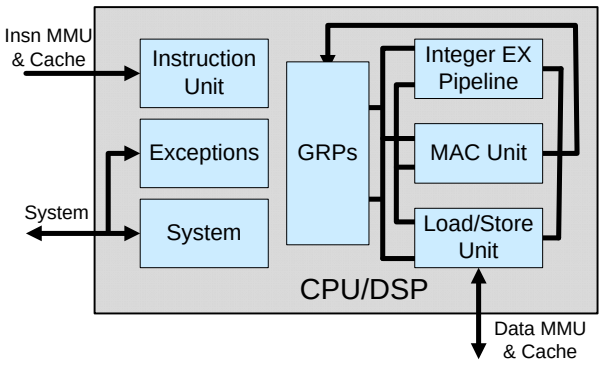

Figure 1 OpenRISC CPU

In addition to the standard commands of the open-RISC core, several DSP commands are supported in the implementation presented in this paper with the architecture shown in Figure 2. The main reason for supporting such commands in the reference design was to evaluate the efficiency of the System-C framework not only when designing RISC-CPUs but also when implementing complicated processing units. A list of those commands is shown in Table 1. Note that the open-RISC core can also execute all the ordinary RISC commands such as additions, multiplications, conditional and unconditional branches, compares, loads, stores etc.

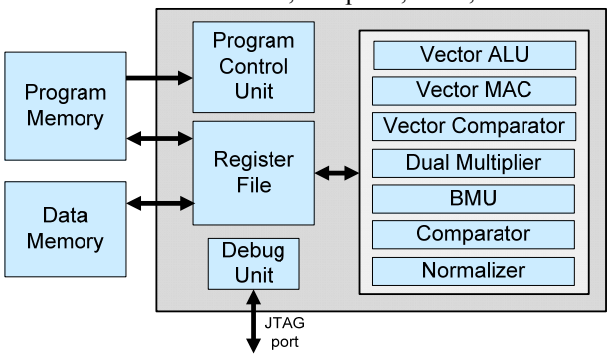

Figure 2 OpenRISC DSP extension

4 Experimental Results

The described reference designs have been implemented using both a traditional hardware description language (VHDL) and SystemC, by a group of students having almost negligible previous experience in both VHDL and SystemC, but having a solid background in Computer Architecture. Those students have Grade Point Average (GPAs) that differed by at most 5%, while they have been working at the exact same environment.

Table 1 : DSP Special Commands

Instruction	Operation	Purpose
SMLAxy{cond}	16 x 16 + 32 → 32	Signed MAC
SMLAWy{cond}	32 x 16 + 32 → 32	Signed MAC wide
SMLALxy{cond}	16 x 16 + 64 → 64	Signed MAC long
SMULxy{cond}	16 x 16 → 32	Signed multiply
SMULWy{cond}	16 x 32 → 32	Signed multiply long
QADD Rd, Rm, Rs	SAT(Rm + Rd)	Saturating add
QDADD Rd, Rm, Rs	SAT(Rm + SAT(Rs x 2))	Saturating add double
QSUB Rd, Rm, Rs	SAT(Rm - Rd)	Saturating subtract
QDSUB Rd, Rm, Rs	SAT(Rm - SAT(Rs x 2))	Saturating subtract double
CLZ{cond} Rd, Rm	COUNTZ(Rm)	Count leading zero
DSPRCa Rd	LD Register High	Load Low

		Part of MAC register
DSPRCb Rd	LD Register Low	Load High part of MAC Register
ROL	Rotate Left	Left Rotation
ROR	Rotate Right	Right Rotation
SUBSHIFT	rD = rA-SHIFT	Integer Subtraction with shift
ADDSHIFT	rD: rA+SHIFT	Integer Addition with shift
ABS	rD = |rA|	Absolute value
MASA	rD = (rA-rB) * (rA[32:16])	Multiply High Part
RND	rD = rA+2^15	Round
CMPL	rD= (rA)'	Complimen t
SQUR	rD = (rA[31:16] * rA[31:16])	Square
EXP	rD = (number of leading bits)-8	Exponent

The reason for picking such students was to be able to check whether SystemC is indeed an easier language for someone that has just entered the hardware design world that traditional Hardware Description Languages. Moreover, by comparing the implementation results when a certain FPGA is targeted, the efficiency (or inefficiency) of the SystemC approach is clearly demonstrated. Another approach for investigating the efficiency of SystemC when designing Processing Units for Embedded System was to have the same person developing both the VHDL and the SystemC model. We strongly believe that this approach would not lead to a fair comparison since much of the time when designing such a core is spent in the basic understanding of the organization of the various circuits; this is the same no matter whether SystemC or VHDL is used, therefore the actual design times would heavily depend on the order of which the two cores were designed (i.e. the design time for the core developed second would have probably been shorter irrespective of the language used).

4.1 DLX case

Two identical copies of the DLX were designed using both VHDL and SystemC. The time periods needed for designing the main blocks of the DLX, in each of the cases, are listed in the following table. It should be noted that the two students designed the two cores, were working exactly 5 hours each day, had absolutely no previous experience in either VHDL or SystemC.

This table clearly demonstrates that there is a speedup of about 1.5 when designing the DLX processing core in SystemC while this speedup is further increased while the Learning time for each of the language is taken into account. It should be stressed though that both student were very familiar with the C programming language.

Table 2 : Design Times of DLX blocks

Block-Name	**SYSTEM C**	**VHDL**
ifstage	3 day	6 days
decstage	9 days	15 days
alustage	3 day	4,5 day
memstage & wbstage	3 day	3 day
top level	3 day	4,5 day
data hazards & forwarding	12 days	15 days
Sub-Total	**33 days**	**48 days**
Basic Learning	6 Days	16 Days
Total	**39 Days**	**64 Days**

Moving to the implementation details of the two designs when targeting the same FPGA device (Spartan3-400, Speed Grade-3) we end up with the results demonstrated in the table 3.

As the results in this table present, although the SystemC design is implemented using more hardware resources, the overhead is far less in all cases than the design time speedup gained when using SystemC instead of VHDL, while the clock frequency achieved in both cases is virtually the same.

Table 3 : Implementation Characteristics of DLX

Logic Utilization	**SystemC**	**VHDL**	**SystemC Overhead**
Number of Slices:	810	679	19%
Number of Slice Flip Flops:	422	333	26%
Number of 4 input LUTs:	1480	1239	19%
Maximum Clock Frequency (MHz)	61.2	60.1	1%

4.2 Open-RISC case

The open-RISC DSP core has also been designed using SystemC. Although this was a much more complicated design than the DLX, the time needed for its development, when SystemC was used, was not significantly higher. Figures 3 and 4 give an overview of the design time for the various subsystems of the open-RISC DSP core.

Based on the report of the designers of the initial Open-RISC CPU core [14], it took four very experience engineers more than 6 man-months to design it using Verilog (another Hardware Description Language). However, as Figure 3 clearly demonstrates, it took an undergraduate student only 70 days (or about 3 man-months) to design the same unit using SystemC! Moreover, all those DSP commands that were added to the standard open-RISC core were developed in less than one man-month as Figure 4 shows.

Moving to the implementation of the core in an FPGA, the initial design in Verilog takes 3000 slices (as listed in [14], while the one designed using SystemC just 3276 as the report of the tools show; an overhead of less than 10%. Moreover, the complete open-RISC DSP (supporting all the extensions) can be placed in less than 3600 slices!

Moreover, the initial core is running at 33MHz, while in the same FPGA device the SystemC designed one run at 36MHz!

Overall, using SystemC the design time was cut in half while the silicon overhead was less than 10% and the speed of the implemented silicon was virtually the same!

4.3 Overall Remarks

From all the results presented in the last sub-section it is clear that the productivity, of at least the un-experienced hardware designers, is heavily increase if SystemC is used. It should be stressed that in both cases the exact same designing/debugging and implementation tools were used. By carefully analyzing the design process of the two devices, we ended-up with the following reasons for this increase:

- The designer is much more familiar with the actual language format and keywords since SystemC is based in C++, whereas he is not in the case of an HDL, and moreover he has to spend much time getting familiar with the HDL specific building blocks like processes, interfaces etc.
- SystemC provides many different, yet easy, ways of describing the functionality of the modules since the basic language blocks are indeed numerous. This result in significantly fewer lines of code; in average in the DLX case the SystemC code was about 40% less than the corresponding VHDL code, whereas in the open-RISC DSP, the code reduction was about 70%!
- Debugging in SystemC is much easier, because, in the majority of the cases, the error messages referred to the particular lines where the errors existed. On the other hand, when VHDL was used, errors were not clearly marked during compilation (even though it was the same simulation environment and probably one of the most widely used such tools, Modelsim).

The main disadvantage of SystemC comes from the fact that since it is effectively a subset of C++, the designer is tempted to use complex data structures, pointers etc, that cannot be turned into real hardware.

5 Conclusions

SystemC seems to be a very promising approach for designing a networking Embedded System comprising of both Software and Hardware. This paper clearly demonstrates the productivity of a relatively un-experienced hardware designer can be significantly higher if he is using SystemC than in the case a traditional hardware description language is employed when designing the core of a network processor or a Digital Signal Processor. This productivity difference grows with the complexity of the designed hardware core; the more complicated it is the higher the productivity difference between SystemC and traditional HDLs. Moreover, the hardware modules designed in SystemC require, in the worst case, 20% of additional hardware resources, whereas they have the exact same performance as the ones designed using an HDL. The advantages of SystemC are further augmented due to the fact that the co-simulation of software and hardware, performed in an embedded system, is heavily accelerated (by at least 300%) by using hardware modules designed in SystemC since everything is executed in a C++ development environment.

Acknowledgements

Part of the work presented in this paper has been carried out within the ICT AWISSENET project (Grant agreement no 211998), financed by the European Community.

References

[1] W. Miuller W. Rosenstiel, J. Ruf, "*SystemC Methodologies and Applications*", Kluwer Academic Publishers, ISBN 1-4020-7479-4 May 2003

[2] T. Rissa, A. Donlin, W. Luk, "Evaluation of SystemC Modelling of Reconfigurable Embedded Systems", DATE 2005, March 2005, Germany

[3] L. Benini et.al, ***"****Legacy SystemC co-simulation of multi-processor systems-on-chip****",*** IEEE ICCD 2002, Sept 2002, Freiburg, Germany

[4] B. Sirpatil, J. Armstrong, J. Baker, *"Using SystemC to Implement Embedded Software"*, 11th International HDL Conference and Exhibition (HDLCon 2002), March 2002, California, USA

[5] I. Yarom, G. Glasser, I. Davidovitch, D. Mamet, *"Three Different Usages of SystemC in Chip Design"*,SNUG 2003, March, California USA

[6] M. Steinert, S. Buch and D. Slogsnat, *"Using SystemC for Hardware design; Comparison of results with VHDL, Cossap and Cocentric"*, SNUG Europe 2002.

[7] F. Bruschi, F. Ferrandi, "*Synthesis of complex control structures from behavioral SystemC model*", DATE 2003, March 2003, Germany

[8] D. Lettnin, A. Braun, M. Bodgan, J. Gerlach, Wolfgang Rosenstiel,*"Synthesis of Embedded SystemC Design: A Case Study of Digital Neural Networks"*, DATE 2004, Feb 2004, France.

[9] S. Chtourou, O. Hammami, "*SystemC Space Exploration of Behavioral Synthesis Options on Area, Performance and Power consumption*", IEEE ICM 2005, Dec 2005, Pakistan

[10] L. Charest, E.M. Aboulhamid C. Pilkington, P. Paulin, "*SystemC Performance Evaluation using a Pipelined DLX Multiprocessor*", DATE 2002. March 2002, France

[11] W. Klingauf. Systematic "*Transaction Level Modeling of Embedded Systems with SystemC*", DATE 2005, March 2005, Germany.

[12] M. Bolado et.al. "*Platform based on Open-Source Cores for Industrial Applications*", DATE 2004, Feb 2004, France

[13] D. Patterson and J. Hennesy, "*Computer Organization and Design*", Elsevier publishers, 3rd Edition, ISBN 1-55860-604-1, 2004.

[14] OpenRISC 1200, http://www.opencores.org/projects.cgi/web/or1k/

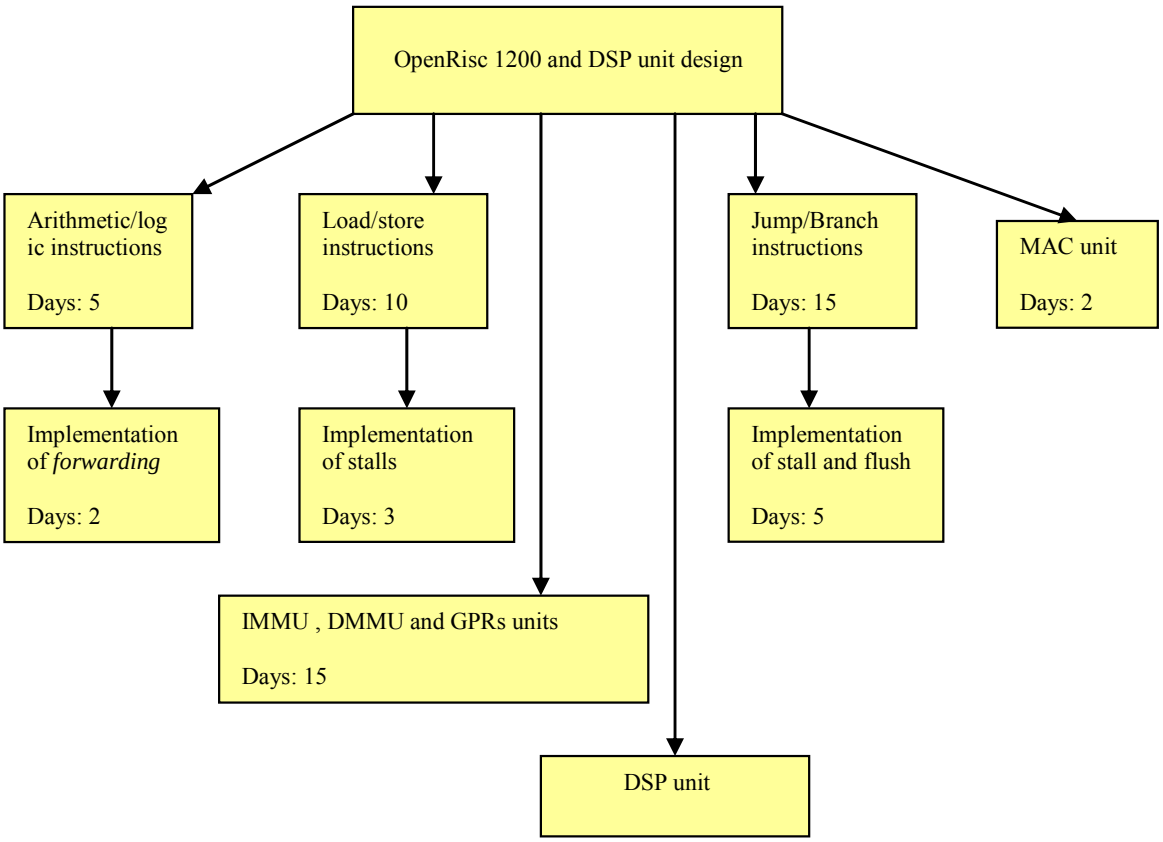

Figure 3: OpenRISC CPU Design Times

Figure 4: OpenRISC DSP unit Design Times