Usages of SystemC in Chip Design – a Case Study

Itai Yarom, Jer5-0, (itai.Yarom@intel.com)
Gabi Glasser, Jer5-0, (gabi.glasser@intel.com)

ABSTRACT

This paper describes the studies conducted at Intel Jerusalem to implement a SystemC design flow. The team analyzed three aspects of the design flow: (1) Architectural trade-offs on a High-level SystemC model; (2) SystemC-to-gate-level flow on a control block and (3) SystemC-to-gate-level flow on a DSP block.

1. INTRODUCTION

SystemC is an open C++ class library used for hardware system design and validation. The SystemC class libraries add hardware attributes to the C++ language. One of the major advantages of SystemC is that it can be used to describe a system at several levels of abstraction, starting at a very high level of functional description and down to synthesizable Register Transfer Logic (RTL) style. Using SystemC in a design flow arouses questions such as: What is the ROI (Return On Investment) of using SystemC? Is there a complete development environment supporting SystemC? How difficult is it to learn the language? What does an open source language mean for the developer? And will SystemC survive the time test, or will it disappear in a few years from now?

SystemC opens the door for a new method of developing hardware systems. For example, using SystemC as a tool for architects to develop the chip concept, followed by hardware and software developers who can take that concept and refine its parts to become the hardware of the chip, the firmware and software runs over the chip (e.g., drivers). In this paper we will try to answer some of the questions posed above by looking at three scenarios in which we think that SystemC can benefit current design flow. The first scenario involves using a high level SystemC model to evaluate architectural trade-offs. Secondly, we look at the flow from SystemC to gate-level netlist. Finally, we look at a digital signal processing (DSP) design flow using SystemC.

In the past decade we have observed a transition from gate-level design to RTL-level design. SystemC is a candidate for the language that will be used at all levels of system and chip design. Starting to use SystemC in current RTL/Behavioral design can accelerate the transition. Finding advantage for starting to use SystemC today can suggest the future of the language.

In Section 2 we present the motivation for using SystemC for chip design, followed by a short SystemC overview in Section 3. Section 4 describes SystemC usages in a chip design flow, follow by conclusions and recommendations in Section 5. We wrap up with acknowledgements and reference in Sections 6 and 7.

2. MOTIVATION

In this section we will present the current design flow and how design flow can be improved by using SystemC. SystemC can provide many benefits to the design flow, including:

1. Architecture development – Architects can deliver an "executable specification" that includes functional, architecture and timing description. If necessary, some blocks of the model can be refined from higher level of behavioral or RTL style SystemC in order to get more precise results. The SystemC model can then help in deciding which part will be implemented in hardware and which parts will be implemented in software.
2. Pre-silicon samples for development teams and customers – Compiling the SystemC into an executable file is similar to generating a model for customers or for the software development team. A good example is a microprocessor. The creation of a pin-and-cycle accurate model at an early stage of the project, prior to the existence of a full RTL model or any other physical model such as an FPGA of the chip, enables other teams to start developing software to be run on the processor or hardware interfacing it, based on the microprocessor SystemC model.
3. Faster and license-free simulation – SystemC is based on C++ and therefore the design can be compiled into an executable file. Each simulation is basically a run of that execution file. The run of exec file is faster than a run of an HDL model inside a simulation. Furthermore, there is no limit on the number of multiple runs of this exec file, unlike the simulation license scheme.
4. Environment for high-level description to netlist generation – This enables developing designs such as DSP that include complicated algorithms with no need to get into implementation considerations. If those blocks are developed with SystemC, one can then use the SystemC model with some refinement as the final stage before synthesis. According to the commonly used DSP design methodology, a high level model is written in Matlab or C++ and then translated manually into RTL code (Verilog or VHDL) for synthesis. In Section 4.3 we will present in detail a DSP flow using SystemC.

SystemC can fit into the gap of a development tool in the system design level. Furthermore, it provides an environment for taking the system-level design and generating a gate-level netlist. Using in-house tools that take advantage of the open source characteristic of SystemC, or EDA tools that support SystemC, the SystemC "synthesis" can be performed.

3. WHAT IS SYSTEMC?

In this section we will present the SystemC language and the Synopsys CoCentric System Studio and the CoCentric SystemC Compiler for developing and testing SystemC designs. SystemC is an open source language, first introduced by the OSCI (Open SystemC Initiative) in 1999. The OSCI are major CAD and IP companies standing behind SystemC, including Synopsys, Cadence, and others ...

SystemC is a collection of C++ classes that add hardware elements to the C++ language. Those elements include hierarchical support, modules, ports, signals, logic types (i.e., 0, 1, X and Z), multiple clocks and resets support and parallel execution of events. SystemC libraries can be downloaded free of charge from the OSCI site [6]: http://www.SystemC.org. The SystemC libraries can be modified as part of the open source model. In addition, SystemC simulation is performed using SystemC standard ANSI C++ compilers (e.g. GCC, Intel C/C++ compiler). The SystemC compilation executable can run in Linux, Unix or Windows environments (different compilation is required for each operation system). More details on the SystemC language can be found in [5,6,8,9]

There are several Electronic Design Automation (EDA) companies that provide CAD tools for SystemC. Synopsys developed the CoCentric System Studio and the CoCentric SystemC Compiler for SystemC. This environment uses the open source SystemC and adds to it additional functions, including synthesis of SystemC, generation of Verilog or VHDL code from the SystemC code, and simulation environment that supports dynamic parameters. In Section 5 we will explore the development of design with and without the Synopsys CoCentric System Studio [1,2], and present our impressions. The Synopsys CoCentric SystemC Compiler generates RTL.

4. SystemC usages and Results

In this section we will present the work done at Intel Jerusalem (ICGJ) together with the Jerusalem College of Technology (JCT – Machon Lev). In this work we analyzed several aspects of using SystemC in the design flow: (1) high-level architectural trade-offs on a system level model; (2) SystemC to gate level flow on RTL and behavioral SystemC design of a control block; and (3) from C to gate-level flow on a DSP block. Detailed description of the SystemC usages is described below.

4.1. High-level architectural trade-offs on a system level model

One of the advantages of SystemC is the ability to create a fast and simple chip model. This model can then be used for evaluating the architectural trade-offs of the design, or it can be used for proof of concept. Furthermore, this model can help in deciding which blocks will be implemented in hardware and which will be software-implemented. The hardware blocks can then be refined into a detailed model that can be synthesized. In addition, the software part that includes firmware and software can be developed using the SystemC hardware model. This SystemC hardware model can be used as a prototype for potential customers.

SystemC can be used for exploring architecture trade-offs. The design we used was a design of a network controller block. This consists of hardware elements and firmware (which runs on a processor within the design). In our SystemC design we used a simple model of the processor. However, this model sufficed for performing trade-offs investigation. Figure 1 describes the design functionality, and Figure 2 describes the SystemC design structure as it was implemented in Synopsys CoCentric System Studio environment.

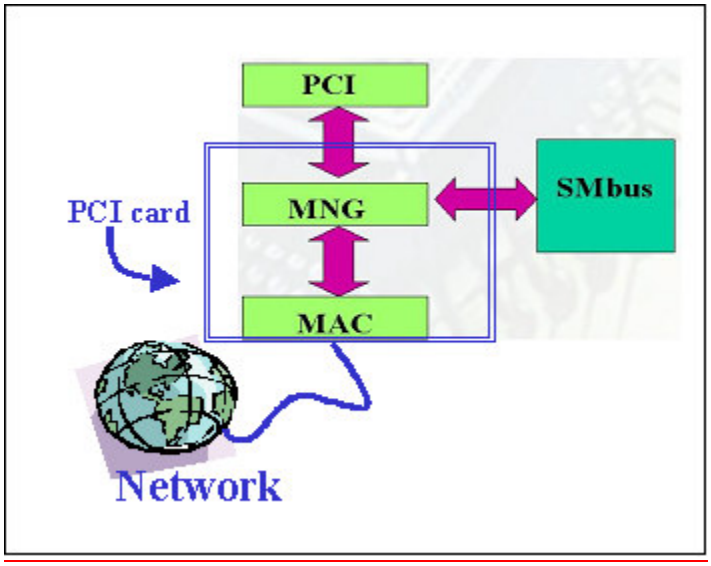

Figure 1: The test case used for the high-level architectural trade-offs was the network controller design. The MNG block is discussed below. This block, located in the computer's network controller, receives management packets from the network and executes the required manageability procedure by interfacing the PCI bus and SMBus of the computer.

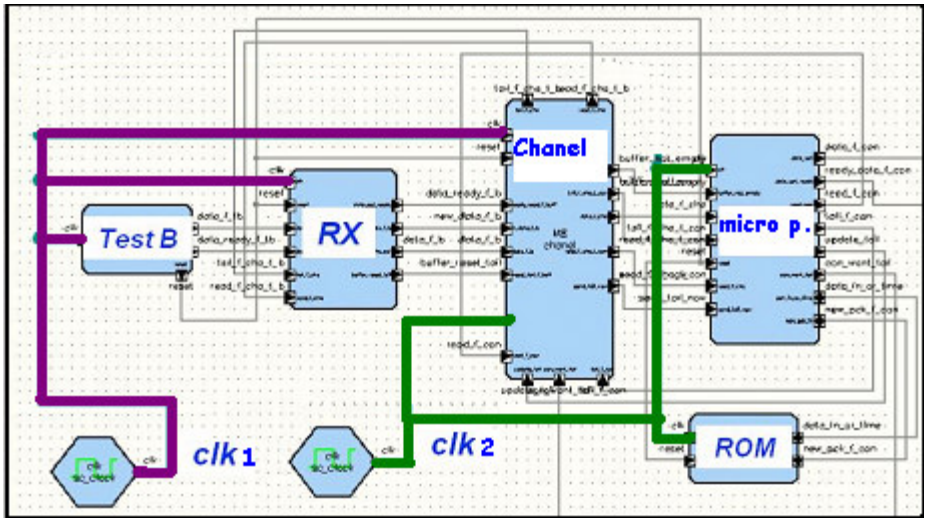

Figure 2: The design of the system level model used for the high-level architectural trade-offs, as can be seen in the CoCentric System Studio.

The MNG block received inputs from the PCI, MAC and the SMBus and performed an action on the inputs. Our goal was to find the best buffer size needed to handle this information, considering the behavior of the elements that push the information (i.e., the PCI, the MAC and the SMBus) and the time needed to perform the action. The information of the load was known, since we had information on the behavior of those elements in previous chips. Otherwise, the behavior could be investigated as part of the SystemC model.

In order to find the optimum buffer size, we dynamically controlled the buffer using the CoCentric System Studio[1]. The optimizer goal is to decrease the number of lost packages, while finding the lower bound of the buffer size. The algorithm is described below:

1. Let APL be the Allowed number of Packages Lost.
2. Let BS be the buffer size. Set BS = init value.
3. Run simulation.
4. Let PL be the number of Package Lost.
5. Let MPL be the Maximum number of Package Lost.
6. Set MPL = PL.
7. if (PL > APL) then
   BS++;
   else BS--;
8. if ((MPL > APL) && (PL <= APL)) then {
   print BS;
   exit;
   }
9. If (MPL < PL) MPL=PL;
10. Rerun simulation (with the new BS) and then go back to step 7.

[1] This can be done without the environment of the CoCentric System Studio, but the CoCentric environment provides built-in support for such tasks. Furthermore, the monitoring of the buffer size is easier to perform as part of the CoCentric environment.

The above algorithm resulted in the best buffer size. Furthermore, this algorithm enables finding the best buffer size when the allowed number lost of packages may be greater than zero. One of the breakthroughs of using this technique for architecture trade-offs is the dynamic of parameter changing while performing reset to the chip at each change. In this case we simulated optimizing one parameter, but the same technique can be used for optimizing multiple parameters. The trade-offs between the parameters can be controlled by known decision-making techniques and algorithms, which can use different weights for each parameter and attempting to optimize the parameters while considering their important (weight). Looking on the optimization for different sets of weight can help the architect learn the behavior of the design and its trade-offs.

4.2. SystemC-to-gate-level flow on RTL and behavioral SystemC design of a control block

In section 4.1 we created a behavioral SystemC. In this section we address the question of how we can take the behavioral SystemC and use in synthesis. We present a flow from SystemC to gate-level netlist. There are two flows available: (1) A flow that uses SystemC behavioral style [3] model as an input to the SystemC synthesis, and (2) a flow that requires SystemC code refinement to RTL level [4] description and then performance of SystemC RTL code synthesis. We performed both of the flows on the design presented in Section 4.2. A behavioral and RTL SystemC models were used, and the synthesis results were compared to a Verilog model of the same block. The synthesis was done using Synopsys tools, including CoCentric System Studio and CoCentric SystemC Compiler. The CoCentric System Studio provided a simulation environment for debugging the design, and the CoCentric SystemC Compiler provides synthesis environment that generated Verilog code out of the SystemC design. Once the Verilog code is generated it can be used in any digital design flow and is not limited for Synopsys synthesis tool only. We used Synopsys Design Compiler for the Verilog synthesis.

MNG block synthesis results comparison			
	Manual Verilog	SystemC RTL	SystemC BEH
Number of ports	42	42	42
Number of nets	186	177	625
Number of cells	163	153	120
Number of references	27	30	62
Combinational area	1401	1276	11556
Noncombinational area	2203	2243	11257
Net Interconnect area	18337	16989	151568
Total cell area	3605	3519	22813
Total area (sqr micron)	21942	20509	174381

Table 1: Synthesis results comparison[2]

We can see that the SystemC RTL style synthesis results, presented in table 1, are similar to the results in the Verilog RTL code synthesis manual. The similarity is in the area and timing parameters. This is contrary to the behavioral synthesis results (as can be seen in table 1), when the area was eight times bigger. Therefore, we recommend performing the synthesis on the RTL SystemC design, using the CoCentric SystemC Compiler.

4.3. From C/ SystemC to gate-level flow on a DSP block

The current design flow for DSP designs uses C models, which can interface with Matlab. After testing the C model and verifying that it performs the needed functionality, then the RTL code needs to be generated. The C model owner and the RTL model owner translate it manually. Verification that both codes have the same functionality is done using simulation.

[2] The results do not include the microprocessor logic.

This flow is not straightforward and it has some limitations. First, there is a gap between the architecture C model and the RTL model. In the translation process, some sacrifice will occur in order to enable the RTL model creation. For example, the C model checks the general algorithm. It is untimed (declaration of timed and untimed can be found at [5]) model. For example, the filter delay model is hard to model in the C model, but will be modeled in the Verilog model. Second, it is not easy to verify that the two models are identical. Furthermore, sometimes they are different, as in special working modes that weren't model in the C code. Verification should to be done by simulation techniques, which are less effective as compared to formal verification techniques, such as equivalence checking.

To solve these issues, SystemC provides the same development environment for both the untimed behavioral DSP and timed RTL models. The DSP designer can work on the DSP in the C or SystemC environment[3]. When the model is ready, the RTL designer can refine the untimed C or SystemC behavioral model to timed SystemC RTL, or can use the untimed behavioral synthesis tool on the behavioral DSP model[4]. Verifying that the results are equal cannot be done today using formal verification techniques, but this is an open issue for the EDA vendors, who must provide support for SystemC in their formal tools.

We took a DSP design of a filter (described in Figure 3), which was designed originally in C and then translated manually to Verilog. We took the C model and translated it into SystemC models, one behavioral and one RTL. The behavioral and RTL SystemC were synthesized using the Synopsys CoCentric SystemC Compiler. Using this flow a synthesizable and readable Verilog file can be generated.

The first flow (seen in Figure 4), using the RTL SystemC, model worked smoothly on this example. The generated Verilog from SystemC passed equivalence checking (using a formal verification tool) with the manually written Verilog of the DSP filter. The translation of the C model to the SystemC model was easy straightforward. After gaining a basic understanding of the design, it took only several hours to perform the transfer. The entire task of the above of C to gate level required only two days in our first shot, and it can take even less with increasing experience with SystemC and the DSP design.

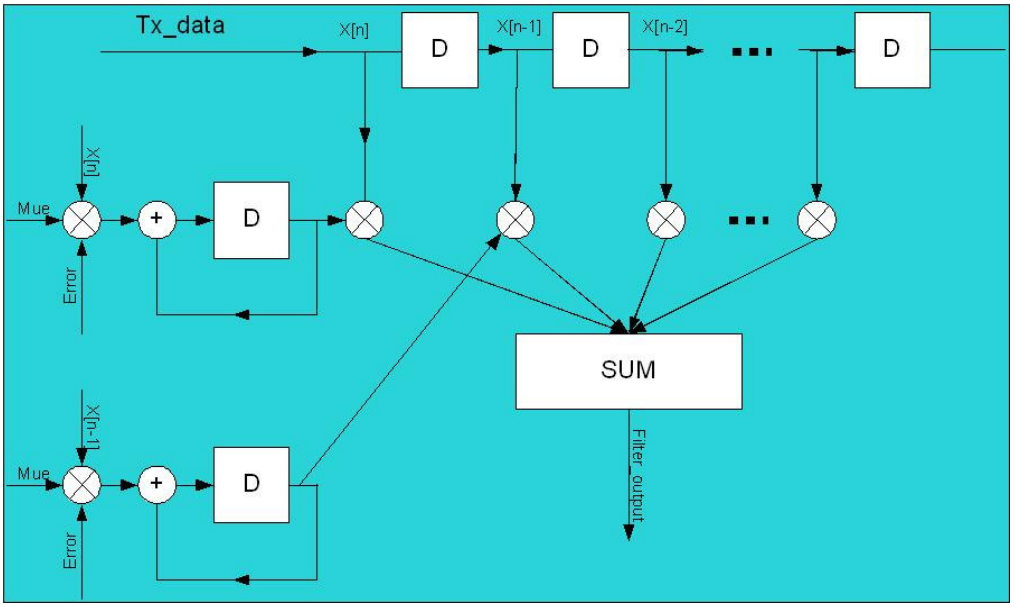

Figure 3: The DSP buffer design

[3] SystemC is an extension of C++, which is a particular extension of C. We recommend using SystemC instead of C or C++ for hardware models.

[4] In order to use the behavioral synthesis tool, the behavioral design should be at a certain behavioral level, for example one that includes clocks and ports. Otherwise, the behavioral DSP model must be refined to include those details.

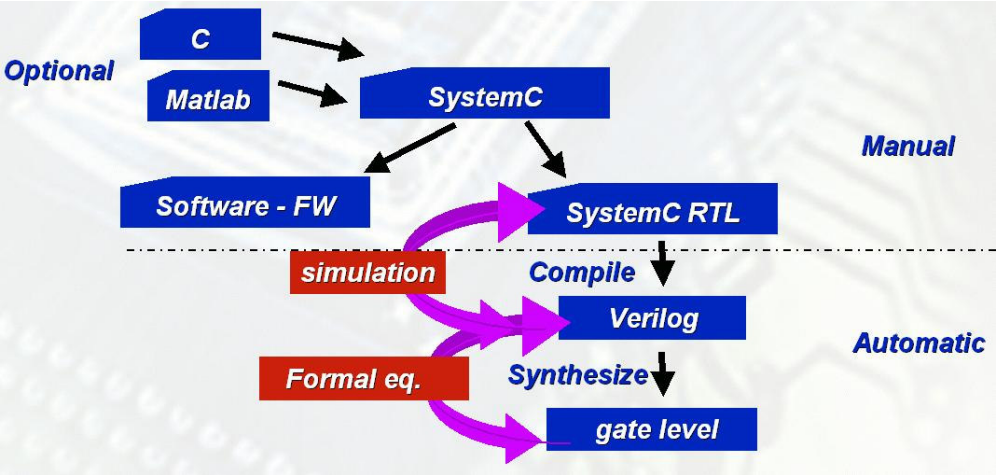

Figure 4: The DSP recommended flow

Using RTL SystemC model is more efficient, as compared C and RTL models. On the one hand, initial translations of the C model to Verilog take a similar amount of time as do the translation of the C model to SystemC. On the other hand maintenance of the C and the RTL models will require adding the changes to both of the models. This will increase the time and effort compared to maintenance of the SystemC model. Once a SystemC model had been reached, then this model is used for both check and debug of the algorithm and RTL synthesis. In the SystemC model the changes are performed on the SystemC RTL, and those changes move with no effort to the Verilog and the netlist using synthesis.

The other flow of behavioral synthesis was not efficient enough. The result of the behavioral synthesis was not logical equivalent, and the netlist result unsatisfactory. The results of the behavioral synthesis are inferior in area and timing aspect to the RTL synthesis results. We should note that Synopsys are working to improve those results, but in the meantime we recommend on the SystemC RTL synthesis flow.

5. SUMMARY

As a result from the evaluation described above we believe that the SystemC language is ready for use at all stages of system and chip design. As we have shown, the SystemC can improve current chip design flow. We have focused on three areas, described in detail in subsections 4.1, 4.2 and 4.3.

According to Swan [8], one of the SystemC goals is to enable system chip modeling. As have shown here (in subsection 4.1), one of the SystemC strength is enabling the investigation of architectural elements trade-offs. When designing the chip, the architects can set the parameters that they want to optimize. They can then define weights to those parameters in order to determine the parameters' importance. Using the optimized algorithm that we have shown, the architects will find optimal values for those parameters. By performing optimization using different weights, the architects can better understand the design trade-offs.

Following the architectural trade-offs, we have shown, that the SystemC model can be synthesized to gate-level netlist. In Section 4.2 two synthesis flows were compared, and while the behavioral synthesis flow is not yet mature, the SystemC RTL synthesis flow is working extremely well. Our results showed that the netlist generated from SystemC and that generated from Verilog (written manually out of the specification) were similar. Therefore, moving from designing chips in Verilog or VHDL to SystemC will not affect the netlist quality.

Finally, we presented a case in which using a SystemC can increase the flow productivity significantly. DSP designs are done today in most cases using a C and Matlab models. SystemC is based on C++, and therefore can easily replace the C models while keeping the connection to the Matlab environment. Instead of manually re-writing the C model in RTL, the designer can refine the SystemC model to RTL level abstraction and synthesized it. Furthermore, using the SystemC environment for the DSP enables to perform maintenance changes only in one place (the SystemC model) instead in two places (the C and the RTL models).

In summary, we have presented advantages of using SystemC in today's design development flows. Using SystemC requires an investment of learning the language. Since SystemC is based on C/C++, this is a relatively simple task. On the other hand, the SystemC's advantages, some of which have been presented here, enable a good ROI for this transition in cases where SystemC can bring added value (more information can be found in Section 4). Regarding the Synopsys CoCentric System Studio, it adds cost to the open source no costs SystemC environment. Nevertheless, the Synopsys CoCentric System Studio and the Synopsys CoCentric SystemC Compiler enable good development and synthesis environments, which can further facilitate the transition to SystemC. Therefore, we believe that the CoCentric System Studio and the CoCentric SystemC Compiler can bring added value to the SystemC.

6. ACKNOWLEDGMENTS

We would like to thank Maty Vaisman, Rafi Cohen and the Synopsys team for their support in this SystemC evaluation effort, by providing license and support for the CoCentric System Studio and the CoCentric SystemC Compiler, and offering material and courses on the SystemC language, which were extremely helpful. We would like to thank Professor Natan Avivi and the Jerusalem College of Technology (JCT) for working with us on this project. Without them this project would not be as successful as it was.

7. REFERENCES

[1] M. Amirfathi, U. Holtmann, L. Ramachandran, H. Toma and Y. Zhang, CoCentric System Studio -Synthesizable SystemC RTL Code Generation, Synopsys, July 2001.

[2] CoCentric System Studio - Release Note, Synopsys, 2002.

[3] Describing Synthesizable Behavioral SystemC, Synopsys, August 2001.

[4] Describing Synthesizable RTL SystemC, Synopsys, May 2001.

[5] T. Grötker, S. Liao, G. Martim and S. Swan, System Design with SystemC, Kluwer Academic Publition, 2002.

[6] Open SystemC Initiative (OSCI) web site: http://www.SystemC.org.

[7] SystemC version 2.0 User's Guide, Open SystemC Initiative (OSCI), 2001.

[8] S. Swan, An Introduction to System Level Modeling in SystemC 2.0, Open SystemC Initiative (OSCI), May 2001.

[9] S. Swan, D. Vermeersch, D. Dumlugöl, P. Hardee, T. Hasegawa, A. Rose, M. Coppola, M. Janssen, T. Grötker, A. Ghosh and K. Kranen, Functional Specification for SystemC 2.0 Open SystemC Initiative (OSCI), October 2001.