Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2008, Article ID 902653, 10 pages
doi:10.1155/2008/902653

Research Article

SystemC Transaction-Level Modeling of an MPSoC Platform Based on an Open Source ISS by Using Interprocess Communication

Sami Boukhechem and El-Bay Bourennane

UMR CNRS 5158, University of Burgundy, 9 Avenue Alain Savary B.P: 47870, 21078 Dijon Cedex, France

Correspondence should be addressed to Sami Boukhechem, sami.boukhechem@u-bourgogne.fr

Received 29 February 2008; Revised 20 May 2008; Accepted 18 August 2008

Recommended by Michael Hubner

Transaction-level modeling (TLM) is a promising technique to deal with the increasing complexity of modern embedded systems. This model allows a system designer to model a complete application, composed of hardware and software parts, at several levels of abstraction. For this purpose, we use systemC, which is proposed as a standardized modeling language. This paper presents a transaction-level modeling cosimulation methodology for modeling, validating, and verifying our embedded open architecture platform. The proposed platform is an open source multiprocessor system-on-chip (MPSoC) platform, integrated under the synthesis tool for adaptive and reconfigurable system-on-chip (STARSoC) environment. It relies on the integration between an open source instruction set simulators (ISSs), OR1Ksim platform, and the systemC simulation environment which contains other components (wishbone bus, memories,..., etc.). The aim of this work is to provide designers with the possibility of faster and efficient architecture exploration at a higher level of abstractions, starting from an algorithmic description to implementation details.

1. Introduction

It has recently become possible to create complex embedded systems called the multiprocessor system on chip (MPSoC), usually used for embedded applications. These systems contain several microprocessors, memories (shared, private), shared busses, and peripherals integrated in a single die [1]. As a consequence, the system designer is confronted with new challenges and difficulties related to the integration of such complex systems. So before any implementation, it is necessary to validate by simulation the system to be implemented. The chosen TLM simulation framework permits rapid exploration of several solutions containing different descriptions of the system components.

For this purpose, a hardware/software TLM cosimulation is used to validate the behavior for both the hardware and the software components of embedded systems, as well as the interaction between them. Moreover, TLM cosimulation also permits the performance evaluation of the whole system at the earlier stages of the design flow before building a prototype, which is faster than HDL register-transfer level (RTL) simulation [2, 3].

Traditionally, mixed language cosimulators are used [4] for simulation which generates a communication overhead between different simulators, often resulting in a significant degradation in the execution time [5]. It is thus necessary to use the same language for modeling software and hardware, and simulating these models at system level in a unified systems design approach. Many research and commercial products provide cosimulation environments which combine ISS, HDL simulators, or systemC models [6, 7].

This motivated our choice of systemC as the modeling and simulation environment for our MPSoC platform. SystemC has become a standard in system level design; it is one of the leading C/C++ design environments, and is an open source, free simulation environment [8].

In addition, TLM cosimulation becomes easier and more efficient, because the entire system can be simulated

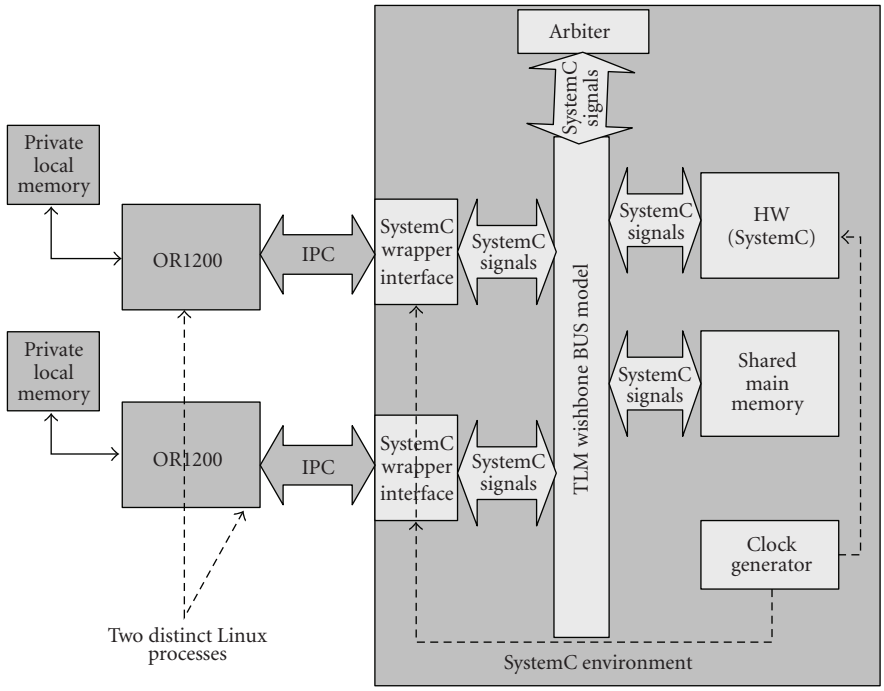

Figure 1: STARSoC platform.

within a single simulation engine (systemC). It permits the simulation at different abstraction levels starting at a very high level of functional description and continuing after refining over time, to synthesizable Register-Transfer Level style, and even combines different levels in one model. The systemC simulation kernel also treats parallel execution and provides functions required to model hardware timing and concurrency [8].

The ISSs in our platform run as distinct UNIX processes on the host system. They communicate between themselves and with hardware components through their systemC interface wrappers which communicate with the other platform components via abstract systemC communication channels [9, 10]. The instruction set simulators based on C language communicate with their systemC interface wrappers via interprocess communication (IPC) [11] primitives.

In this context, there are many related works, such as [12]. The major contribution of this paper is to develop the cosimulation environment between OR1Ksim [13] and systemC. We used IPC (PIPE) in order to provide MPSoC models for an OpenRISC processor at a higher level of abstraction. These can be used in order to accelerate the validation, performance analysis, and architecture exploration for our project, called STARSoC. It is particularly used for evaluating the hardware-software partitioning and also for comparing the system modeled at a high level of abstraction (TLM) with the register-transfer level during software prototyping.

Our objective in this paper consists in comparing the three abstraction levels which are traditional register-transfer level modeling and transaction-level modeling at instruction accurate level and cycle-accurate level.

In our case study, we have used an open source ISS derived from the OR1Ksim simulation platform which is designed for monoprocessor simulation. For the use in the case of the simulation of multiprocessor systems, we connect two ISSs with systemC communication platform models, by using interprocess communication. Thus, it is very easy to add or to remove a processor from the MPSoC design. The interconnection models and other hardware components can be modeled in systemC or any other hardware description language. The interconnection is based on a standard Wishbone bus [13]. Our communication model uses the shared memory communication mechanism. All processors are simulated at the same abstraction level, for each TLM level [14]. The rest of the paper is organized as follows. We begin in Section 2 with a brief definition of STARSoC design flow and transaction-level modeling. In Section 3, we discuss the simulation platform used in this work. Section 4 describes the bus architecture. Section 5 is devoted to the study of our transaction-level modeling steps of the wishbone bus. Section 6 introduces the communication model adopted in our work. In Section 7, we provide some explanations for the modified instruction set simulator based on OR1Ksim. Section 8 presents some results of experiments we obtained by simulation. We finish with a conclusion in Section 9.

2. STARSoC Design Flow Overview

An overview of the STARSoC design flow [15] is shown in Figure 2. The input description consists of a set of communicating parallel software and hardware processes described in C-code [16]. We specify the number of reconfigurable processors (OpenRISC1200) and the list of peripheral

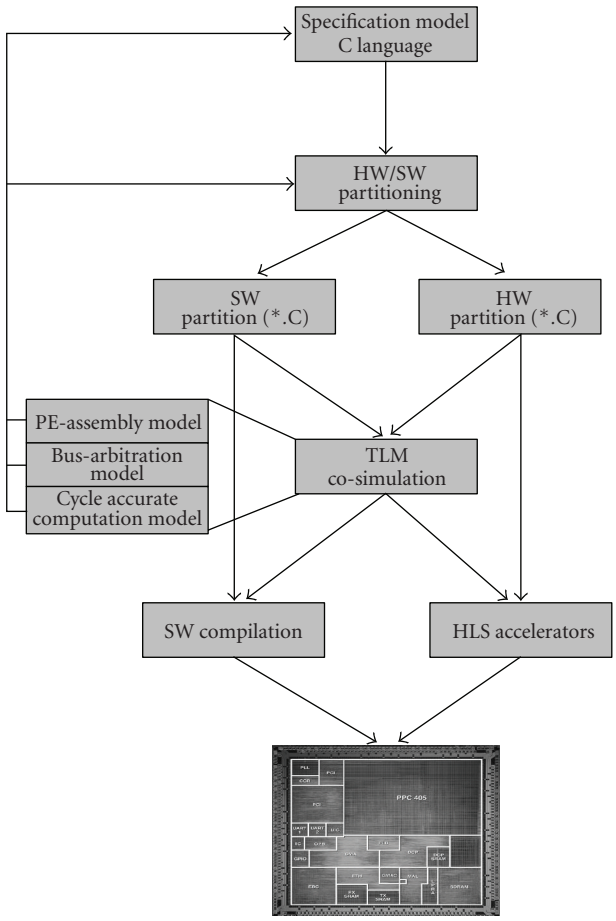

Figure 2: STARSoC design flow.

input/output components connected to each processor. After hardware-software partitioning, the hardware part is synthesized into register-transfer level architecture, and the software part is distributed to the available processors.

The hardware and software partitions are defined by the user. The software part will be reinstrumented to make appropriate calls to the hardware component synthesized via the communication system bus. The bus-based communication architecture will be synthesized to interconnect the software processor(s) and the hardware coprocessor(s). We use the GCC compiler to synthesize the machine code of the software processes and download it into the program memory of each processor in the generated MPSoC platform.

In this work, we focus on TLM abstraction as defined in the following.

2.1. Transaction-Level Modeling

Currently, transaction-level modeling is widely referred to in system-level design literature. It permits a fast simulation and performance evaluation of a complex System on Chips (SoCs) earlier in the design flow, with a more modular and efficient code. This reduces the time-to-market compared with RTL and ensures practical gains for design, because TLM is less detailed. Timing details can be incorporated into these models to allow performance estimation and architecture exploration before the RTL (HDL/systemC) code is generated.

We have several definitions concerning the exact place of TLM in the simulation level. TLM is not a single abstraction level but involves several abstraction levels (multilevel model). We can refine the models over time to include more information. In most cases, TLM is defined above the RTL [17, 18]. Gai and Gajski [14] clearly define four transaction level abstraction models, where the communication and the computation are explicitly separated. The system is represented as a set of communicating processes. These processes perform computations and communicate with other processes through an abstract channel. The different transaction levels defined by [14] are PE-assembly model, bus-arbitration model, time-accurate communication model, and cycle-accurate computation model. In our work, the use of TLM in STARSoC platform (MPSoC platform) design refers to a set of abstraction levels quoted in [14].

2.2. STARSoC TLM

In Figure 3, all levels belong to the TLM levels except the first level. We provide below a brief description of all these levels.

(1) The first model is a *specification model* which is described by a parallel process (c program) without any architecture details.

(2) The second model is the *PE-assembly model*, implemented by using ISSs (OR1Ksim) which communicates through interprocess communication. We chose an implementation of IPC, called PIPE, for its capacity of data and command transfer.

(3) The third model is the *bus-arbitration model*. In this level, we have added two parameters: address (for memory access) and bus arbiter (for bus access), also by using IPC. In this level, STARSoC is a time approximate computation. In each clock cycle, the ISS performs one instruction. In our work, this level can also be called an instruction accurate execution model.

(4) The fourth model is the *cycle-accurate computation model*. In this model, each ISS is cycle accurate.

The advantage of this model is that it allows designers to exploit the platform at earlier stages of the design flow.

3. Proposed Architecture

The target architecture used in this work is a multiprocessor system-on-chip (MPSoC) platform, whose communications are performed via a shared memory, as shown in Figure 1.

Our reference platform consists in integrating several ISSs wrapped under the systemC wrapper interface, the TLM wishbone BUS model, private memories (associated with each ISSs), and a shared memory used for communication between the ISSs. Because we have a small set of processors

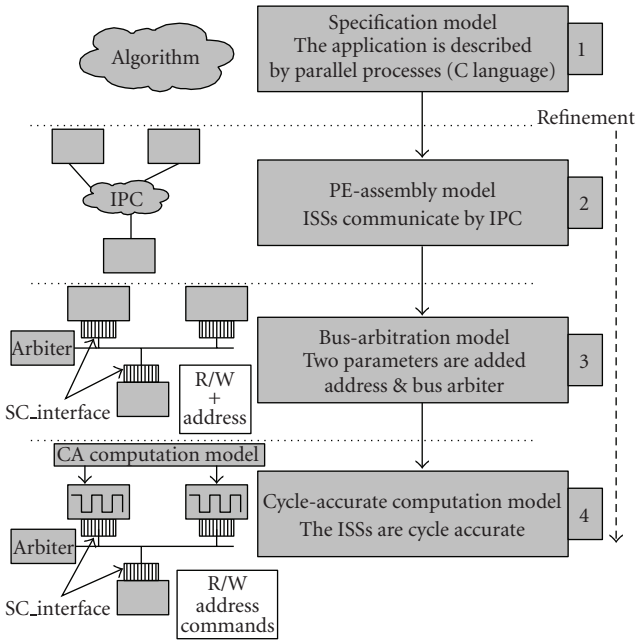

Figure 3: STARSoC platform at different abstraction levels.

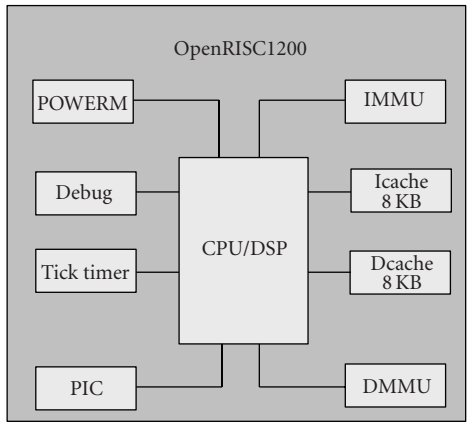

Figure 4: ISS unit architecture.

in this work, we chose a shared bus as the interconnection model instead of network-on-chip (NoC) and shared memory as the communication model instead of message passing.

The ISSs used in our platform are derived from the open source simulator of the OpenRISC processor (OR1200), written in C and called OR1Ksim (shown in Figure 4). Each ISS contains its own cache memory, private local memory, and the minimum set of units required to perform basic functionality (see Figure 4). These ISSs can provide detailed functional information, such as register values, the program execution time, and other timing information, used among them to carry out the comparison between three abstraction levels: instruction accurate level (IA) which corresponds to the third level in Figure 3, cycle-accurate (CA) level which corresponds to fourth level in Figure 3 (owned to TLM), and RTL during simulation.

```
                or1ksim\cpu\or32\insnset.c
INSTRUCTION (1_sb)
  {
    . . .
  if (add_s)
    {
      . . .
      write (fd_sb_data, &PARAM1, strlen (&PARAM1));
      write (fd_sb_add, &add, strlen (&add));
      . . .
    }
    . . .
  }
. . .
INSTRUCTION (1_1bz)
  {
    . . .
  if (add_1)
    {
      . . .
      write (fd_1bz_add, &add, strlen (&add));
      read (fd_1bz_data, &data, strlen (&data))
      . . .
    }
    . . .
  }
```

Algorithm 1: Example of an IPC Call from the ISS.

The TLM wishbone bus model is used to connect the ISSs with the rest of the system. These ISSs are executed simultaneously and share the memory address space used for inter processor communication.

The platform is entirely implemented in systemC language, except for ISSs which are wrapped under systemC. All these components are connected by the TLM wishbone BUS model, also implemented in systemC. The TLM bus model is based on the basic Wishbone communication protocol functionality at a high-abstraction level. It executes the Wishbone bus transaction without timing accuracy or pin accuracy. Besides the private memories which are associated with each ISS, our platform includes one main shared memory used for communication.

In order to wrap ISS under systemC, a systemC wrapper interface is added to the ISS C model. This process involves defining a systemC module and adding input/output ports that correspond to the ISS input/output arguments which we have implemented in the load/store functions from the OR1k CPU directory (see Algorithm 1). This can be done by using interprocess communication as shown in Figure 5. The systemC wrapper interface is made sensitive to a positive clock edge. At every positive clock edge, the systemC wrapper interface calls the corresponding C function inside the ISS via IPC, such as PIPE (used in our example) and Sockets [19]. SystemC wrappers and ISSs C models are both run as distinct Linux processes.

The communication between the components (software-software/software-hardware) can be effected using this

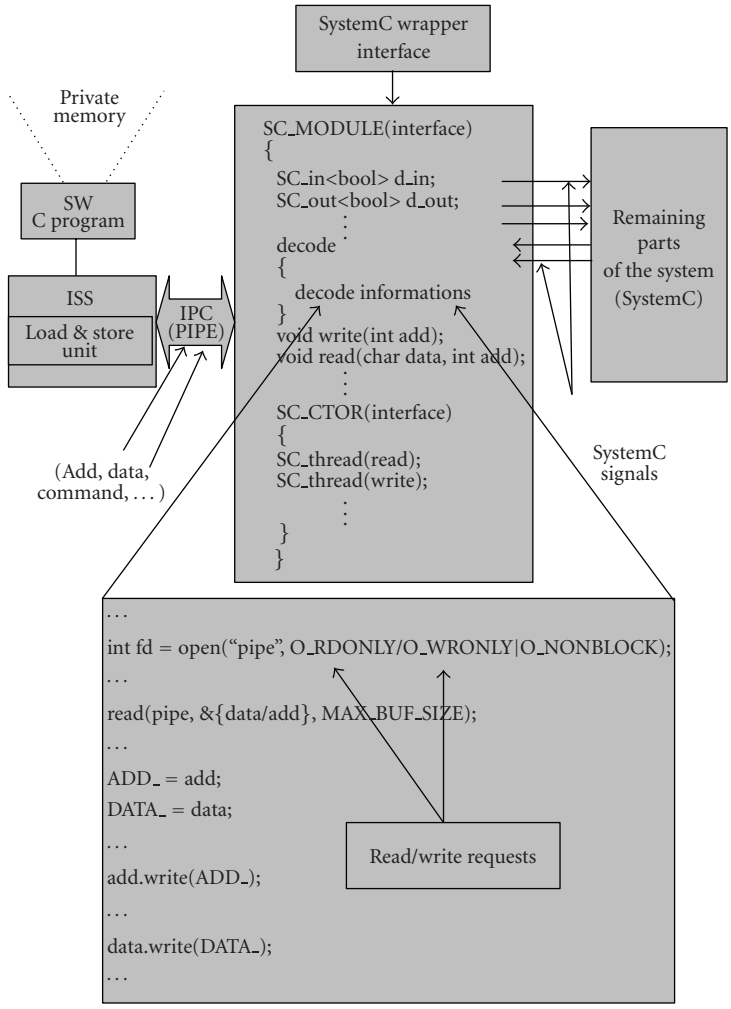

Figure 5: SystemC ISS wrapper interface.

shared memory, through systemC signals. In this case, the multiprocessing environment synchronization must be ensured by using the semaphore. Our simulation environment is configurable and allows specification of several parameters, such as the type of interconnect (Wishbone,..., etc.), the number of processing elements, and memory parameters.

In our example, we have used two ISSs, the first one writes data on the shared memory, the second ISS reads the data written by the first one (see Figure 6), taking into account the synchronization between them.

At the beginning, we wrapped each ISS under SystemC. At every clock cycle, the ISSs execute one instruction, in order to perform instruction accurate simulations. The pipeline effects are not considered at all. In this case, we have a systemC instruction accurate simulation. We then refine the simulation to a cycle-accurate simulation; we fully simulate the processor pipeline and we take into account the number of cycles necessary for each instruction (processor pipeline stage, memory access,..., etc.). We have in this case a full systemC cycle-accurate computation simulation.

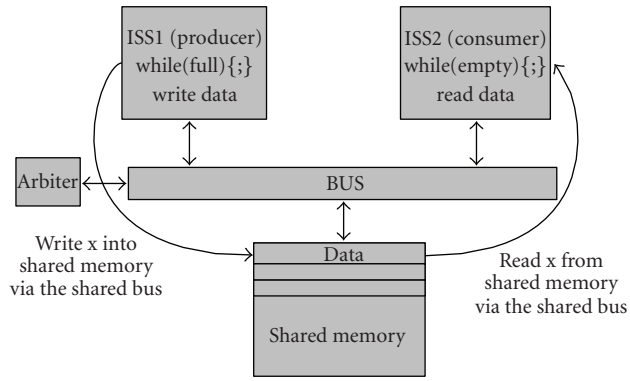

Figure 6: Our communication model using shared memory.

4. Bus Architecture

Most SoC designs are based on hardware blocks connected together with bus signals which are classified as groups of data, address, and control links. Several companies provide

the following SoC bus architectures so that designers can easily integrate the IP blocks into a single silicon chip: Core Connect, AMBA, CoreFrame, Wishbone, and Silicon Backplane Network. Our architecture platform is designed around the Wishbone bus, whose architecture was developed by Silicore Corporation, Minn, USA [20]. In 2002, Silicore placed the bus specification into the public domain via OpenCores [13] which is an organization that promotes the development of open IP cores. The Wishbone bus architecture is very simple since it defines only one bus protocol. However, the Wishbone bus architecture supports various features depending on the desired bus operations: multiple masters, single cycle read/write, block transfer cycles that systematically perform a set of single read cycles and/or a set of single write cycles.

However, in the case of TLM simulation, the wishbone bus protocol needs to be redefined as transaction-level ports of TLM. To the best of our knowledge, there has been no TLM implementation for the wishbone bus to date, we have only descriptions at signal level. It is, therefore, necessary to map signals into TLM transaction level ports (not pin accurate and not cycle accurate as only the computation models are cycle accurate). These models (TLM) should meet a few requirements including the following:

(i) speed,

(ii) flexibility,

(iii) the ability to model and evaluate several arbitration schemes,

(iv) clarity and ease in integrating other component models efficiently.

5. Transaction-Level Modeling of the Wishbone Bus System

We implemented a transaction-level model of the wishbone bus. This model accurately respects the wishbone bus protocol.

Since the bus is modeled as an abstract channel without including any specific details of the bus protocol, it enables faster communication simulation models.

We present all the steps in our methodology to develop a wishbone bus TLM architecture, by describing our transaction-level modeling steps. We first used function calls (performed by calling IPC functions) to model the wishbone signals and we then used systemC signals to implement the wishbone protocol.

In the first step, we modelize the behavior of each transaction-level port. For example, in the RTL handshaking protocol, a master can immediately get an *ACK_I* (bus grant signal) from the bus, after sending an STB_O (bus request signal) if the bus is ready (free). This step is implemented as the port's transaction of a master call *Check_bus_Grant()* and receives *true* as a return value. The arbiter selects a request from this master after applying an arbitration strategy, decodes the destination address, and sends the request to the slave destination. The arbiter calls read() and write() functions implemented in the slave. The slave receives the request from the arbiter, performs any required computation, the read/write operation, and optionally waits for a fixed number of cycles before sending a response back to the arbiter. The arbiter ensures eventual completion of the transaction. After that, the master (ISS) sends ADDR (address) and DATA (read/write data). The transaction is a single *word/bytes* read/write transfer and receives ACK.

In the second step, we implemented wishbone signals by systemC signals. The translation into a systemC signal is done by an *SC_Interface* module associated to each ISSs.

The TLM wishbone bus model that we created is shown in Figure 7, where the implementation of communication is less detailed than register-transfer level.

The simulation speeds were measured at both TL model and RTL. The TL model is about 300 times faster than the RT level model.

6. Communication Model

An important aspect in the design of multiprocessor systems is the exchange of data between SoC modules. Several communication methodologies are possible, such as shared memory and message passing. However, shared memory is the most common type of interprocessor communication paradigm, for multiprocessor system-on-chip (MPSoC) platforms, where a small set of processors share a common address space.

In this section, we present the model of communication developed and implemented in our platform architecture, which is based on the shared memory approach to perform data exchange between ISSs. We thus have two different types of memories.

A private memory space exists for each ISS which cannot be seen by any other ISS in the system except for the owner. We also have memory that is shared by all ISSs and used for communication.

An example is illustrated in Figure 6 to demonstrate our communication model, established between two processors via the shared memory which is used for this purpose. The platform architecture consists of two ISSs connected by a TLM Wishbone bus model. Besides private memories, a shared memory is used by the ISSs for data exchange. The program running on ISS1 (producer) deposits data needed by the program running on ISS2, into shared memory and waits until this data is read by the program on ISS2 before depositing other data and vice versa. But in this case, we have a well-known problem which is synchronization; it is a very critical issue in platforms based on shared memory communication. This problem arises due to the fact that the bus (in the case of a shared bus) as well as the memory is shared between different ISSs. On the one hand, these ISSs exchange data through this shared memory, on the other hand, these data are sent and received via the TLM shared bus. This situation generates shared resource access conflict if we have several simultaneous requests.

Thus, we need to use a simpler arbitration scheme such as round robin (RR) and the semaphore in order to ensure synchronization. In the example shown in Figure 6, two processors are involved. We used a round-robin arbitration

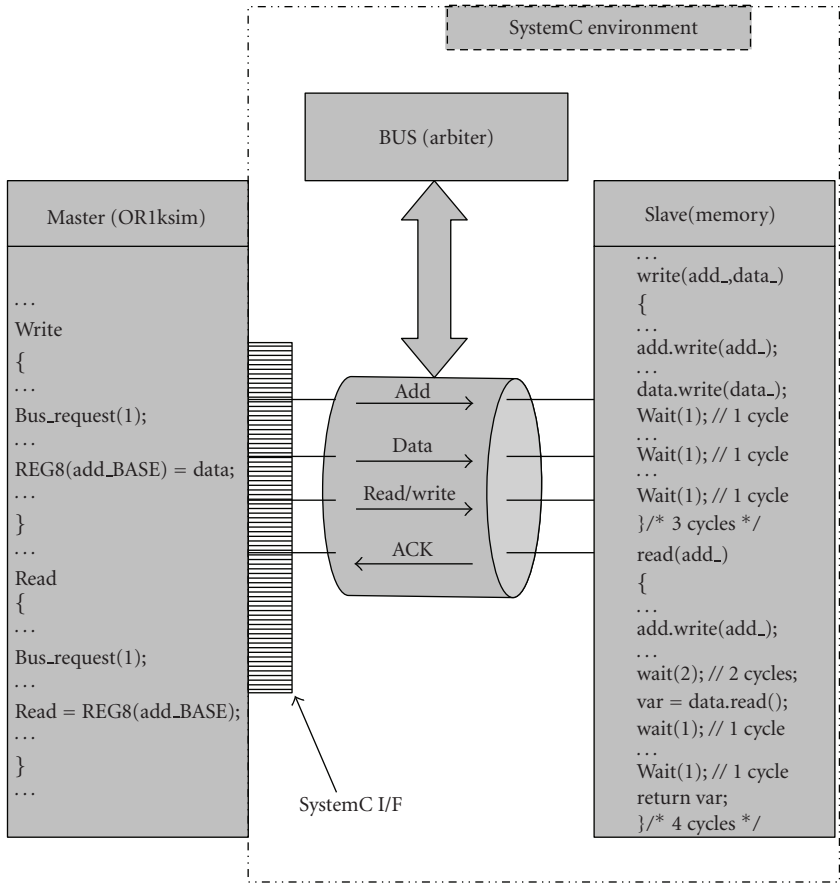

Figure 7: TLM wishbone BUS.

policy for the bus access, and we used the semaphore for shared memory access.

7. Processing Elements

The ISS used in our framework is based on OpenRISC architecture emulator, written in C and called OR1Ksim, licensed under the GNU LGPL license. The goals are to emulate 32-bit OpenRISC CPUs with a high level of abstraction capable of running real operating systems, such as NetBSD [21], RTEMS [22], eCos [23], and uClinux [24] and intended for embedded, portable, and network applications.

OpenRISC 1200 is an open source IP-core freely available from the OpenCores website [13]. This soft core is a MIPS-based 32-bit scalar RISC with Harvard microarchitecture, a 5-stage integer pipeline, virtual memory support (MMU), and basic DSP capabilities. The overview of the OR1200 architecture can be seen in Figure 4.

For this core, we have two descriptions at different abstraction levels. The first is a free open-source description written in a synthesizable Verilog code, with a low level of abstraction (RTL abstraction) verified by several functional tests and implemented into FPGAs and ASICs.

The second description is written in C code (OR1Ksim simulator) and provides several features [25]:

Table 1: Simulation time results at different abstraction levels.

	Number of iterations		
	10	100	1000
Abstraction levels	Simulations times		
Insruction accurate (IA)	0.97 (*S*)	1.5 (*S*)	8.4 (*S*)
Cycle accurate (CA)	1.38 (*S*)	2.3 (*S*)	12.32 (*S*)
RTL	3.4 (*S*)	12.56 (*S*)	62.24 (*S*)

(i) free, open source code,

(ii) high level and fast architecture simulation that allows code analysis at an earlier stage and system performance evaluation,

(iii) supports all major models of OpenCores peripheral and system controller cores,

(iv) easy addition of new peripheral models,

(v) remote debugging through a network socket with the GNU Debugger (GDB),

(vi) support for different environments (memory configurations and sizes, OR1K processor model, configuration of peripheral devices).

The tools used for compilation and debugging are the standard GNU toolchain, including the GCC compiler which

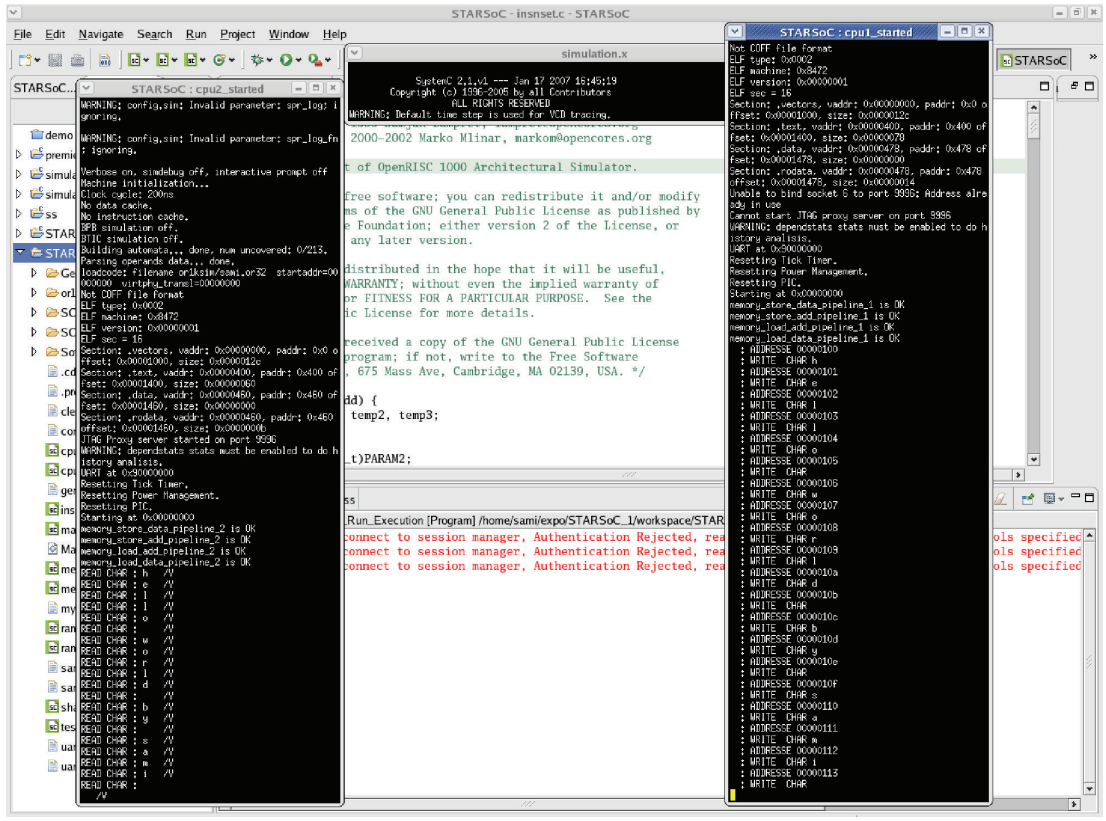

Figure 8: STARSoC IDE.

works well. The debugger is available through GDB, and it is easy with DDD which provides a more user friendly environment.

8. Results

In this section, we first describe the system architecture example used in our experiments, and we then present results that we carried out to validate the architecture generation process of STARSoC and evaluate its performances. We also compare performances among the different abstraction levels.

In our experimental example, we used a shared-memory MPSoC platform generated by a STARSoC generator tool in which multiple OpenRISC processors can be integrated and interconnected by a TLM wishbone bus model.

We performed experiments using the systemC design environment which we applied to a system consisting of two ISSs (wrapped under systemC) accessing a shared memory through a TLM bus. The architecture of the system is shown in Figure 6. All system components were specified in systemC, except for the two ISSs.

The bus arbitration mechanism is managed by the module labeled bus arbiter which is implemented by the round robin arbitration policy. In each cycle, one of the masters has the highest priority to access the shared resource. If the token-holding master does not need the bus at this cycle, the master with the next highest priority who sends a request can be granted the resource.

The interface between the bus and the ISS interface consists of four signals: a read/write signal *rw*, an address *addr*, the data *data*, and an acknowledge signal *ACK*. Similar signals exist between the bus and the shared memory thanks to which the access is synchronized by a semaphore.

ISS was built by the GNU cross-compiler (GCC version 3.4) and cross-debugger (GDB version 5.0) with the OpenRISC as a target.

The application executed by the two processors is stored in their local memory and consists of a producer-consumer (sends characters from one processor to the other) application executed in parallel and synchronized by the same clock signal. The application is loaded on the private memory of the producer core (writer) which then writes those characters into the shared memory at a given address. The consumer core (reader) reads those data from the same address. If the consumer is faster than the producer, the memory will be empty and the consumer waits until the producer writes data. Conversely, if the producer is faster, the memory will be full, and the producer waits until the consumer reads the data.

The same environment was employed for simulation on different abstraction levels. The simulation results were obtained by executing the platform on a Pentium IV at 3.0 GHz with a RAM memory size of 786 MB, based on Linux Fedora Core 3. Table 1 shows three columns 10, 100, and 1000, corresponding to the fixed number of iterations (reads or writes to the memory) of the algorithm which is performed by each ISS. The simulation times shown in

Table 1 correspond to the execution times of the programs at the different simulation abstraction levels.

Three models were generated and simulated: an instruction accurate model (bus arbitration model), a cycle-accurate system simulation model, and a synthesizable RTL model (using Verilog and simulated with ModelSim). Generally, the first two models are used to validate the software and the system architecture (they include the ISSs, bus model at transaction level, and functional models of other components, all of which use systemC models).

All model descriptions were automatically generated from the STARSoC generator tool shown in Figure 8. They have also been validated by simulation using the same testbenches.

After running the testbenches, we obtained some experimental results.

(i) The instruction accurate model is about 7 times faster than the RTL model.

(ii) It runs twice as fast as a full cycle-accurate system simulation.

(iii) Our cycle-accurate model runs five times as fast as an equivalent RTL simulation.

The simulation time analysis was used to compare the efficiency obtained from the TLM description in systemC and from the underlying RTL platform. The results reported above demonstrate the advantage of using a higher level of abstraction than RTL when carrying out architectural exploration.

9. Conclusion

The availability of a fast high level simulation makes architecture exploration possible at different abstraction levels.

In this paper, we have presented and validated our methodology for cosimulation at a high level of abstraction (TLM) within a single simulation environment based on systemC language.

Our environment is based on the use of open source ISSs C models (OR1Ksim), wrapped under systemC language by using UNIX interprocess communication.

Comparing three different abstraction levels, namely, instruction accurate level, cycle-accurate level, and RTL level (VHDL model), we have analyzed the STARSoC generated multiprocessor SoC platform.

The experimental results show that the use of systemC as a modeling language for the design of abstraction levels may significantly reduce the design validation time, enabling the development of very fast models. In addition, the simulation results at higher levels of abstraction show that there are no significant communication overheads between the ISS C model and its systemC wrapper, due to the fact that we have a small number of used cores. This model would be very useful for functional HW/SW cosimulation of large SoCs based on OpenRISC.

This motivates our choice for systemC as a system design language, dedicated to architecture exploration in our STARSoC project which is the main contribution of this work. This gives the designer the possibility of exploring the STARSoC platform at several levels, which represents a notable advantage for STARSoC design flow.

In future work, we plan to add an embedded operating system like eCos and to integrate heterogeneous IPs cores in our platform.

References

[1] P. G. Paulin, C. Pilkington, M. Langevin, et al., "Parallel programming models for a multiprocessor SoC platform applied to networking and multimedia," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 7, pp. 667–680, 2006.

[2] A. A. Jerraya, A. Bouchhima, and F. Pétrot, "Programming models and HW-SW interfaces abstraction for multiprocessor SoC," in *Proceedings of the 43rd Annual Conference on Design Automation (DAC '06)*, pp. 280–285, San Francisco, Calif, USA, July 2006.

[3] K. Hines and G. Borriello, "Dynamic communication models in embedded system co-simulation," in *Proceedings of the 34th Design Automation Conference (DAC '97)*, pp. 395–400, Anaheim, Calif, USA, June 1997.

[4] I. Petkov, P. Amblard, M. Hristov, and A. Jerraya, "Systematic design flow for fast hardware/software prototype generation from bus functional model for MPSoC," in *Proceedings of the 16th IEEE International Workshop on Rapid System Prototyping (RSP '05)*, pp. 218–224, Montreal, Canada, June 2005.

[5] J. Jung, S. Yoo, and K. Choi, "Performance improvement of multi-processor systems cosimulation based on SW analysis," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '01)*, pp. 749–753, Munich, Germany, March 2001.

[6] Coware.Inc., N2C3, http://www.coware.com/#cowareN2C.html.

[7] Seamless, CVE, 2005, http://www.mentor.com/seamless.

[8] Open SystemC Initiative, SystemC Version 2.0, Users Guide, 2001, http://www.systemc.org/.

[9] L. Benini, D. Bertozzi, D. Bruni, N. Drago, F. Fummi, and M. Poncino, "Legacy SystemC co-simulation of multi-processor systems-on-chip," in *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD '02)*, pp. 494–499, Freiburg, Germany, September 2002.

[10] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli, and M. Olivieri, "MPARM: exploring the multi-processor SoC design space with systemC," *The Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, vol. 41, no. 2, pp. 169–182, 2005.

[11] W. R. Stevens, *UNIX Network Programming, Volume 2: Interprocess Communications*, Prentice-Hall, Upper Saddle River, NJ, USA, 2nd edition, 1998.

[12] N. Saint-Jean, G. Sassatelli, P. Benoit, L. Torres, and M. Robert, "HS-scale: a hardware-software scalable MP-SOC architecture for embedded systems," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI '07)*, pp. 21–28, Porto Alegre, Brazil, March 2007.

[13] OpenCores, http://www.opencores.org/projects/or1k.

[14] L. Gai and D. Gajski, "Transaction level modeling: an overview," in *Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '03)*, pp. 19–24, Newport Beach, Calif, USA, October 2003.

[15] A. Samahi and E.-B. Bourennane, "Automated integration and communication synthesis of reconfigurable MPSoC platform," in *Proceedings of the 2nd NASA/ESA Conference on Adaptive Hardware and Systems (AHS '07)*, pp. 379–385, Edinburgh, UK, August 2007.

[16] M. B. Gokhale, J. M. Stone, J. Arnold, and M. Kalinowski, "Stream-oriented FPGA computing in the Streams-C high level language," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '00)*, pp. 49–56, Napa Valley, Calif, USA, April 2000.

[17] Coware, http://www.coware.com/.

[18] Cadence, http://www.cadence.com/.

[19] W. R. Stevens, B. Fenner, and A. M. Rudoff, *Unix Network Programming, Volume 1: The Sockets Networking API*, Addison Wesley, Reading, Mass, USA, 3rd edition, 2003.

[20] Silicore, http://www.pldworld.com/_hdl/2/_ip/-silicore.net/wishbone.htm.

[21] NetBSD, http://www.netbsd.org/.

[22] RTEMS, http://www.rtems.com/RTEMS.

[23] eCos, http://ecos.sourceware.org/.

[24] uClinux, http://www.uclinux.org/.

[25] OpenCores, http://pkgsrc.se/emulators/or1ksim.