Extending Transaction Level Modeling for Embedded Software Design and Validation

Mouna BAKLOUTI Adel BENZINA
MOSIC & Tunisia Polytechnic School
Tunisia Polytechnic School
Tunisia
baklouti_mouna@yahoo.fr, adel.benzina@isd.rnu.tn

Aimen BOUCHHIMA Frederic PETROT
System Level Synthesis Group
TIMA Laboratory
Grenoble, France
{aimen.bouchhima, Frederic.petrot}@imag.fr

Ahmed Amine JERRAYA
CEA LETI
Grenoble, France
ahmed.jerraya@cea.fr

***Abstract*—In this paper, we propose to extend the Transaction Level Modeling (TLM) approach –initially intended as a higher level abstraction of Register Transfer Level (RTL) hardware (HW) design- to cope with embedded software (SW) design and validation. We aim at introducing new SW TLM concepts which will enable refinement of communication at the SW side. The proposed methodology allows system designers to decide about HW and SW communication architecture jointly, so as to ensure maximum performance efficiency for their designs. As such, multi-processor system-on-chip (MPSoC) heterogeneity would be addressed more efficiently from communication viewpoint.**

I. INTRODUCTION

Nowadays, the trend towards improving productivity and reducing time-to-market makes the traditional Register Transfer Level (RTL) to layout design and verification flow inadequate. In deed, design at the implementation level gives unacceptable realization costs and delays. To address these challenges, Transaction Level Modeling (TLM) has been recently promoted as the next modeling abstraction for hardware (HW) design [8]. TLM uses a component-based approach, in which hardware blocks are modules communicating with so-called transactions, where unnecessary details of communication and computation are hidden. Doing so, TLM enables speeding up simulation and exploring implementation alternatives early in the design flow (such as bus topology design, bus priorities, and direct memory access (DMA) size optimization). Although allowing early embedded software development and tightening hardware and software development in system-on-chip (SoC) design were one announced goal of TLM, no TLM design infrastructure has been defined for software (SW). In classic TLM practices, SW is either kept functional or fully developed and simulated at very low abstraction level on top of an instruction set simulator (ISS).

Fig. 1 depicts a generic multi-processor system-on-chip (MPSoC) architecture including its HW and SW components. Given the increasing importance of embedded software in current MPSoC designs, the joint co-design of the HW/SW

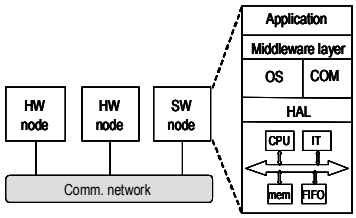

Figure 1. MPSoC generic HW/SW architecture model.

interface is identified as key enabler to tune system performance and master design cost [13].

Fig. 2 shows the different abstraction levels for both HW and SW. The dashed lines joining a HW abstraction level with a SW one define possible integration levels allowing to design and simulate mixed HW/SW systems. The HW/SW interface is defined with respect to each integration level.

In this paper, we advocate a new intermediate integration level called the virtual architecture (VA) level. It associates HW TLM with an equivalent level for SW that we call SW TLM. The SW TLM level corresponds to an abstraction of the classic low level instruction set architecture (ISA) level for SW. At the SW TLM level, SW is described as a set of concurrent and interacting objects managed by an execution environment corresponding to an abstraction of the operating system (OS).

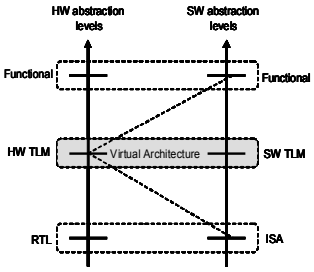

Figure 2. The virtual architecture concept.

1-4244-1278-1/07/$25.00 ©2007 IEEE

The definition of the SW TLM level is largely inspired from recent researches on embedded software modeling. One major contribution of this work is to formalize these efforts, using the transaction level modeling paradigm and to define a global TLM modeling platform for both HW and SW. This allows developing a unified HW/SW interface model to remove discontinuities between HW and SW designs and enabling fast and effective design space exploration.

The rest of the paper is organized as follows. Section 2 reviews some related works in high level HW/SW modeling. Section 3 describes the VA intermediate abstraction level. Section 4 focuses on SW TLM concepts and the definition of the underlying execution model. As an example we apply our methodology to an MJPEG application in Section 5. Finally, Section 6 concludes this paper with a brief outlook on future works.

II. Related Work

Many recent research efforts have focused on abstracting the classic RTL level used as HW/SW integration platform model. Most of these works addressed either the HW side or the SW side of the problem, but none of them provided flexible and unified HW/SW platform model at higher abstraction level. At the HW side, the transaction level modeling (TLM) has been identified as suitable candidate for HW RTL abstraction [3] [4] [6]. TLM is built as a high level application programming interface (API) that defines how HW components communicate. The literature distinguishes three TLM levels: un-timed (called PV: Programmer View level), timed (called PVT: PV with Timing level) and Cycle Accurate. Recently, many works demonstrate TLM application in real-life designs [9] [10] with the focus on architecture exploration and simulation speedup [14]. In all these works, SW is either considered at a low abstraction level (i.e. the instruction set architecture level) or simply at the functional level (mainly as test-bench functions).

At the SW side, , recent research activities have focused on high level real-time operating system (RTOS) modeling within system level design environments like SystemC or SpecC [2][5]. However, in these works, the interaction between the OS simulation model and architecture HW components (i.e. I/O, interrupt handling) is not clearly explained. In [1], we presented a system level simulation model at the OS level. The structure of the HW/SW interface simulation model acts as an adapter between the application SW and the HW part. This interface is composed of two layers: one SW layer, called OS model layer, which constitutes the OS simulation model and one HW layer, called device functional layer, which emulates a set of device controllers. To enable the correct modeling of I/O operations, we use a model of the I/O interface called device functional model. This entity has to ensure the correct interaction with the OS side while providing the necessary adaptation of the communication interface (protocol, abstraction level, etc). While this work proposes a HW/SW refinement, it assumes a fixed HW interface model.

For HW/SW co-design, we note also the Cadence VCC (Virtual Component Co-design) environment which was initially intended to integrate virtual HW and SW components, explore complex HW/SW tradeoffs and analyze resulting performance early in the development cycle [12]. We think that the non success of VCC is mainly due to the fact that it does not use a well defined abstraction level for both HW and SW making the model difficult to deal with.

III. Virtual Architecture Level

A. Overview

Fig. 3 gives an overview of an MPSoC design modeled at the VA level. Parts of the figure in grey correspond to conventional HW TLM objects. Here, the design example is build around hierarchical bus architecture composed of a system bus and a local CPU bus connected to the system bus via a bridge. Unlike conventional TLM designs however, SW is neither executed on top of an instruction set simulator nor fully abstracted at the functional level. In the VA model, SW is modeled at the OS level as a set of SW TLM objects that co-design and interact with the rest of HW TLM components.

Within a SW TLM description, we mainly identify three conceptual layers:

- The program layer that corresponds to the SW being designed by SW programmers. This may be application tasks or device drivers allowing external communication with hardware.
- The resource management layer corresponding to what we call SW BUS. This entity abstracts the real operating system and allows coordinating and arbitrating the rest of components.
- The virtual resource layer that specifies, from a programmer point of view, what kind of resources available in the SW subsystem (SW node of Fig. 1). In our case, we distinguish two kinds of resources: logical memory and processing unit (PU).

An important object that could be qualified as hybrid HW/SW TLM object is the bus functional model (BFM). A BFM is a special bridge that is supposed to link the SW bus with a particular HW bus (namely the CPU bus). Its main role is to forward external accesses (to hardware registers for instance) from the SW side to the HW side. It is also responsible of connecting interrupts coming from the HW side with the appropriate corresponding handlers in the SW side.

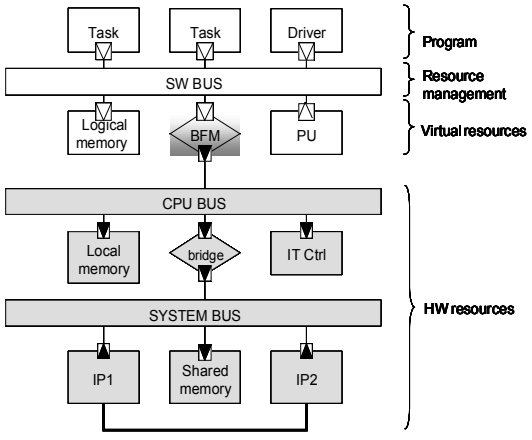

Figure 3. Virtual architecture model.

Note that for all design objects –regardless of the conceptual layer they belong to- the same modeling concepts and notations are being used based on the TLM approach. In the following subsections, the conventional HW TLM methodology is firstly presented. Then the SW TLM basic concepts are described and their application to SW refinement is explained.

B. *Typical Virtual Architecture based design flow*

Fig. 4 depicts a typical SoC design flow featuring the VA abstraction level. Like classic SoC design flows, the proposed flow starts from a (non-executable) specification that undergoes a first partitioning step allowing further HW and SW refinements. In the figure the result of this step is called System Architecture. This corresponds to an executable form of the specification (using SystemC for instance) where annotations are simply introduced to distinguish parts of the application that will be mapped to HW or SW respectively. The ultimate result of the flow is an RTL architecture that can be the entry of conventional synthesis back-end tools.

Unlike classic SoC design flows however, the proposed flow introduces an intermediate design step based on the VA concept. The Virtual Architecture results from the integration of both HW and SW parts that are supposed to be refined up to the TLM level. In SystemC, this integration merely corresponds to the top-level instantiation of the different (HW and SW) TLM modules and channels. This intermediate design step allows to:

- (1) bridge the classic gap between HW and SW design by providing an intermediate level for HW/SW integration allowing gradual HW/SW co-design.
- (2) break the long exploration loop that classically separates the system architecture level from the final RTL level. This enables fast and effective design space exploration, taking benefit from the large simulation speedup of TLM compared to RTL.

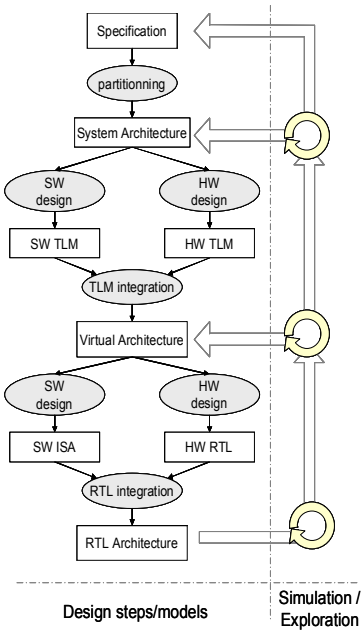

Figure 4. A typical design flow involving the VA level.

C. *HW TLM basic concepts*

In this paper, we consider the SystemC/TLM standard. This standard provides the foundation layer to develop interoperable SystemC TLM IPs (Intellectual Properties). TLM defines small set of generic and reusable TLM interfaces (blocking/non blocking, unidirectional/ bidirectional) through a layered approach: user layer, protocol layer and transport layer. TLM splits in two main abstraction levels, namely PV (Programmer View) and PVT (PV with Timing) as shown in Fig. 5. PV level is designed for embedded SW validation and platform integration. It consists in the vision the programmer has on his system. This view is un-timed. The protocol used at this level is generic (e.g. TAC: Transaction Accurate Communication) and synchronization reflects causal dependency between several computation units and is not based on delay constraints. PVT level corresponds to a timed view of the system. This view is useful for performance estimation. A PVT platform is a PV platform augmented by a specific timed bus model (e.g. STbus, AMBA …).

D. *SW TLM basic concepts*

Fig. 6 outlines the basic SW TLM components. In a SW TLM environment there are modules requiring services "initiators" and modules providing services "targets". Initiators and targets communicate by sending requests and responses back and forth. These modules may correspond to different types of SW components:

- Application components, also called user elements (e.g. task, active object…); these components may be master or slave modules.
- Abstract resources consisting in logical memories and processing units. Logical memories hide the actual physical address mapping of the different storage locations of the hardware architecture (registers, memory banks …). The refinement process is then in charge of mapping these logical locations once the architecture is fixed. Processing units are virtual execution components running SW within a CPU subsystem and corresponding to independent execution threads on the actual hardware architecture (ranging from simple core to symmetrical multi-processing SMP and simultaneous multi-tasking SMT …).

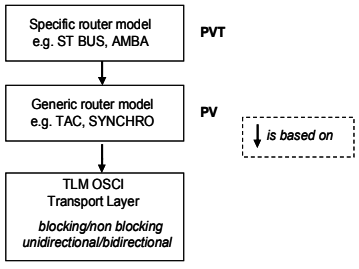

Figure 5. TLM layers.

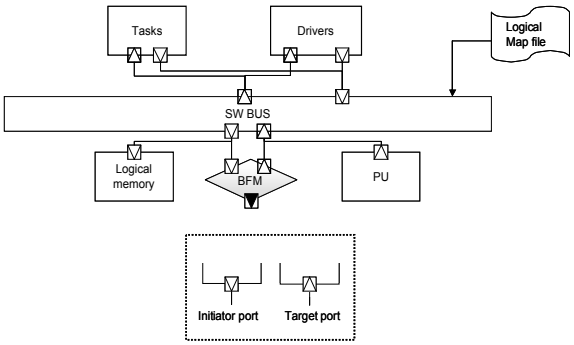

Figure 6. SW TLM components.

- Driver modules implementing external communication with other subsystems or nodes. TLM drivers address one or more logical memory components that hold information about the memory mapping. They need also RTOS model services to access the corresponding HW devices.
- SW bus which is at the heart of the whole SW node model. It could be defined as a logical path that serves multiple SW tasks or multiple SW computation and communicating units upon an OS model. Its main role is to:
  - ensure task scheduling and time sharing;
  - intercept logical transactions and process them;
  - issue these logical transactions to the BFM.

The SW bus is responsible of two important mechanisms: routing and arbitration. The first corresponds to routing services within SW components; while the second dynamically resolves concurrent service requests.

IV. IMPLEMENTATION

A. SW TLM hierarchy

Like HW TLM, SW TLM splits in three layers as shown by Fig. 7. The Service layer is built as a set of interfaces that define how models communicate. In fact, the interface protocol defines the semantic of transferring a service between two different modules. SW TLM interfaces (synchronous/ asynchronous) specify communication services and are based on the TLM OSCI transport layer. During a synchronous transaction, the function does not return until the transaction either completes successfully or fails.

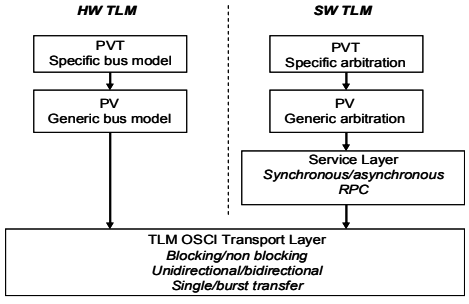

Figure 7. SW TLM layers.

During an asynchronous transaction, the function returns after the transaction has begun, passing a transaction identifier for reference. All interfaces inherit from *sc_interface*. When we move a service from initiator to target we call this a "required service" and when we move a service from target to initiator we call this a "provided service".

SW TLM PV is built on top of SW TLM and is based on the Service layer. At PV level, we have no real notion of timing and transaction's arbitration is generic. PVT is a modeling style where a PV and an arbitration specification coexist. PVT adds timing information on each data treatment or transfer. For the SW bus, the timing must account for the number of transfers as well as arbitration between multiple software components.

B. The global execution model

To enable time accurate simulation, SW execution time has to be modeled. This is achieved by performing static annotations within the original SW code. This kind of code instrumentation is well covered in the literature [7]. Given a processor type, the time needed by a SW basic block to execute is estimated. The SW application code is then instrumented accordingly by annotating each basic block using its corresponding delay (Fig. 8).

These time annotations would correspond to SystemC "wait" statements. However, the statically estimated delays do not take into account the possible occurrence of hardware interrupts, nor do they take into consideration the effect of stalls associated with concurrent accesses to the same address space entity. To solve this problem, we use a special annotating function "crunch" which implements an appropriate algorithm based on the dynamic sensitivity of the SystemC "wait(delay,event)" function:

```
//crunch(d)
wait(d,interrupt) ;
if (HW_interrupt)
   { update d;
     execute ISR ;
     goto start;
   }
else return;
```

From the viewpoint of OS model, HW interrupts (interrupts from the HW part to the SW part) are modelled as an *sc_event* called "interrupt" (associated with an identifier).

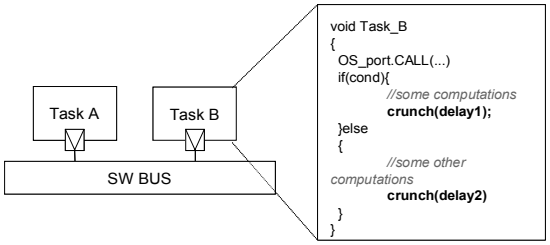

Figure 8. Modeling SW time.

To handle asynchronous interrupts, we rely on the "wait(delay, interrupt)" function inside "crunch()". This function allows waiting for the "interrupt" event with a timeout equal to "delay". In other words, the function returns either when the specific event has occurred within the waiting delay or at the end of the timeout. During the execution of crunch() function, when an interrupt occurs at the target processor, the corresponding interrupt service routine (ISR) should be invoked (implemented within the OS model). The ISR can even call the OS scheduler and another (higher priority) task may be activated causing the currently running task be preempted.

Fig. 9 illustrates the run-time behavior of the crunch() function, where two SW tasks concurrently access the SW bus. The "crunch" function may be called everywhere in the SW code, either in the context of SW tasks or in that of the SW bus. In the latter case, the use of the "crunch()" function allows to evaluate the overhead related to the OS (time slots occupied by the SW bus in Fig. 9).

V. EXPERIMENT: MJPEG CASE STUDY

Based on our SW TLM methodology, we performed a HW/SW co-simulation of an MJPEG application. Fig. 10 shows the task graph of the MJPEG application. It consists of concurrent threads that communicate with each other via FIFOs: TG (Traffic Generator), DEMUX (Demuxer), VLD (Variable Length Decoding), IQ (Inverse Quantization), ZZ (ZigZag scan), IDCT (Inverse Discrete Cosine Transform), LIBU (Line Builder) and RAMDAC (Random Access Memory Digital to Analog Converter). The bold lines represent the decompression flow, and the dashed lines represent the parameters of configuration which were global variables of the initial configuration.

The Virtual Architecture model of the application is shown in Fig. 11. In our model, we have two HW IP blocks: TG and RAMDAC, two SW modules mapped on two ARM7 processors (each SW module executing three tasks), memory, FIFO controller and Interrupt SW. These components are connected via the SYSTEM bus. Fig. 11 shows also the CPU subsystem1's architecture using SW TLM. Here we need only FIFO drivers for communication. The FIFO driver is a slave/master component. In this case, the FIFO driver provides a basic API to the application consisting in read and write services.

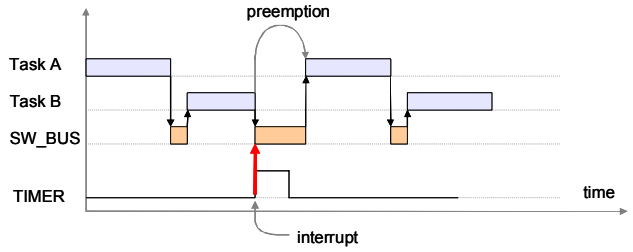

Figure 9. Example of "crunch" run-time behavior.

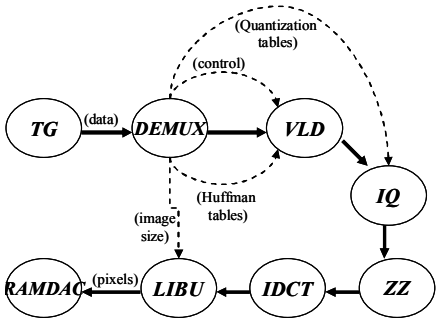

Figure 10. MJPEG application task graph.

A part of driver implementation showing how the write service is implemented using SW TLM concepts is presented in the code below:

```
void w_fifo_drv<Log_ADDRESS,DATA>::write
(const DATA& data);
{
sat_status status;
…
status= initiator_port.CALL(INIT,
mutex_write);

status=initiator_port.CALL
(READ,FULL_REG,full);
…
status=initiator_port.CALL(MUTEX::LOCK,
mutex_write);
```

INIT is a service provided by the SW bus to instantiate pre-defined types as MUTEX, SIGNAL, PIPE, SHM, etc. The read access is implemented in a similar manner.

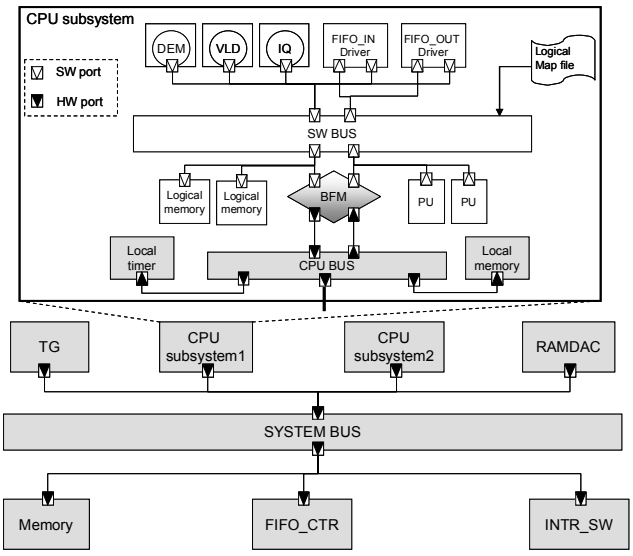

Figure 11. MJPEG VA design.

At the top level, we instantiate the various SW components and we define their binding as follows:

```
{   …
  //Instantiation
sw_router<service_t,int,int> * SW_ROUTER;
fifo_drv<int,int> * FIFO_DRIVER ;
sw_hw_bridge<int> * BFM;
 …
  //Binding
SW_ROUTER->initiator_port(FIFO_DRIVER-
>target_port);
SW_ROUTER->initiator_port(BFM-
>target_port);
FIFO_DRIVER->initiator_port(SW_ROUTER
->target_port);
   …
```

Table I summarizes some simulation results carried out at three abstraction levels (functional level, VA and RTL). These values correspond to the simulation of one sequence of 25 frames. We used ARM7TDMI processors running at 40 Mhz. We note that the Virtual Architectural level is timed (SW is time annotated and HW described at the TLM PVT level). The functional level is un-timed and the RTL level is cycle accurate.

The second column corresponds to the execution time which is the "SystemC" time consumed by the different CPU's in order to process the 1 second video sequence. The simulation time corresponds to the amount of time spent by the host machine to run the simulation. The last two columns are related to simulation accuracy and speed. We notice that the VA level simulation achieves a considerable speedup comparing to RTL, while maintaining a reasonable accuracy (error within 20%). In fact, since we use a high abstraction level, we should tolerate an acceptable error. This level helps validating application and facilitates bugs avoidance before RTL coding is complete.

VI. CONCLUSION

In this paper, we presented a methodology that takes advantage of the SW layered architecture on one hand and the OSCI TLM paradigm on the other hand, to define an extended TLM for SW communication refinement at the Virtual Architecture level.

Along this work, we have been able to:

- propose a methodology for SW refinement at TLM level;
- split up SW TLM into 3 layers, these layers consist in low level layer, PV and PVT;
- enable TLM support for RTOS simulation model.

Thanks to this methodology, SW design flow merges with the HW one at TLM level enabling simulation speed-up and architecture exploration early in the design flow. Future work will concentrate on developing an automatic code generation tool for communication drivers based on the proposed SW TLM concepts.

TABLE I. SIMULATION TIME RESULTS

Abstraction Level	***Execution time***	***Simulation time***	***Accuracy***	***Speedup***
Functional	--	< 1 ms	0%	~ 10^6
Virtual Architecture	0.90 s	20 s	77%	~ 1260
RTL	0.73s	~ 7 h	100%	--

REFERENCES

[1] A. Bouchhima, S. Yoo and A. Jerraya, "Fast and Accurate Timed Execution of High Level Embedded Software using HW/SW Interface Simulation Model", Design Automation Conference, 2004. Proceedings of the ASP-DAC 2004. Asia and South Pacific.

[2] A. Gerstlauer, H. Yu and D. Gajski, "RTOS Modeling for System Level Design", Proc. Of Design, Automation & Test in Europe, March 2003.

[3] A. Rose, S. Swan, J. Pierce and JM. Fernandez, Transaction Level Modeling in SystemC. OSCI TLM Working Group, 2005.

[4] B. Vanthournout, "Transactional level as the new design and verification abstraction above RTL", Coware Inc, Leuven, Belgium, 2003.

[5] D. Desmet, D. Verkest and H. De Man, "Operating System based Software Generation for Systems-on-Chip", Proc. Design Automation Conference, June 2000.

[6] F. Ghenassia, Transaction Level Modeling with SystemC: TLM concepts and applications for embedded systems, Springer, November 2005.

[7] J. R. Bammi, W. Kruijtzer, L. Lavagno, E. Harcourt and M. T. Lazarescu, "Software performance estimation strategies in a system level design tool", in Proc of the eighth international workshop on hardware/software codesign, pp. 82-86, May 2000.

[8] L. Cai and D. Gajski, "Transaction Level Modeling in System Level Design", CEC Technical Report 03-10, March 28, 2003.

[9] P. Paulin, C. Pilkington and E. Bensoudane, "StepNP: A system-level exploration platform for network processors", in IEEE Design and Test of Computers, vol. 19, part 6, pp. 17-26, 2002.

[10] S. Pasricha, "Transaction level modeling for SoC with Systemc 2.0", in Synopsys User Group Conference, India, 2002.

[11] SystemC, available at: http://www.systemc.org.

[12] Virtual Component Codesign, Cadence Design Systems Inc.

[13] W. Wolf , "The future of Multiprocessor Systems-on-Chip", Design Automation Conference, 41st Conference on (DAC'04), pp. 681-685, June 2004.

[14] X. Zhu and S. Malik, "A hierarchical modeling framework for on-chip communication architectures", Proc. ICCAD, 2002.