

Exploiting TLM and Object Introspection for System-Level Simulation

G. Beltrame, D. Sciuto, C. Silvano
Politecnico di Milano
Milano, Italy
{beltrame,sciuto,silvano}@elet.polimi.it

D. Lyonnard, C. Pilkington
STMicroelectronics
Ottawa, ON, Canada
{damien.lyonnard,chuck.pilkington}@st.com

Abstract

The introduction of Transaction Level Modeling (TLM) allows a system designer to model a complete application, composed of hardware and software parts, at several levels of abstraction. The simulation speed of TLM is orders of magnitude faster than traditional RTL simulation; nevertheless, it can become a limiting factor when considering a Multi-Processor System-On-Chip (MP-SoC), as the analysis of these systems can be very complex. The main goal of this paper is to introduce a novel way of exploiting TLM features to increase simulation efficiency of complex systems by switching TLM models at runtime. Results show that simulation performance can be increased significantly without sacrificing the accuracy of critical application kernels.

1. Introduction

Transaction Level Modeling (TLM) has been introduced in the recent past as a modeling style to describe communication channels at a higher abstraction level with respect to Register Transfer Level. Although Transaction Level (TL) models offer high simulation speed, in some cases, they do not capture enough details about on-chip behavior. TLM simulation speed, in particular when considering cycle accurate models, does not keep up with the increasing complexity of Multi-Processor Systems-on-Chip (MP-SoCs). Furthermore, the design and tuning of an MPSoC often requires many simulation runs of the application on the platform, following each minor optimization or modification of the application.

In this paper, we propose a methodology based on the exploitation of TLM and the concept of *introspection* (from software engineering) to support the dynamic switching of multi-level models for the simulation of complex MPSoCs.

In particular, to support the optimization and debugging phases of small kernels of the target application, we propose a methodology to use a less accurate model for the uninteresting sections of the simulation and a refined model for the kernels under analysis. To the best of our knowledge, cur-

rent TLM simulators do not provide an effective way of abstraction level switching at runtime, and there is no consistent way of modeling multi-level systems.

In more detail, our approach distinguishes between channels and generic modules, to consider their different nature and the need to keep the system in a consistent state. The proposed approach can be used to perform fast simulations on uninteresting parts of an application lifespan, trading off accuracy. Once an interesting kernel is reached, the simulation speed is lowered, restoring accuracy. A simple example of such behavior is shown in Figure 1. Suppose that we

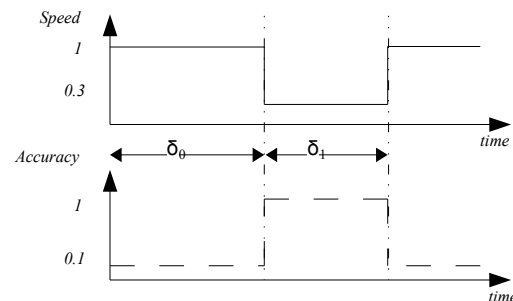


Figure 1. An example of use of the multi-level switching capabilities of the proposed approach

have modified an application and we want to test the results of the modification, but that the application is affected only during the interval δ_1 . During interval δ_0 simulation is running with low-accuracy modules and functional channels, going at full speed at the cost of accuracy. Accuracy is then re-established during interval δ_1 , allowing the designer to reach the interval under analysis in shorter time.

The proposed modeling approach has been validated by using the stepNP simulation platform [10], while executing a set of multimedia applications. The reported experimental results show a simulation speed-up of one order of magnitude, with respect to Bus-Cycle-Accurate (BCA)

simulation, for a selected set of benchmarks. Moreover, the reported results compare a low-accuracy and a cycle-accurate ARM processor module, showing an average saving of 59.4%.

The paper is organized as follows. Section 2 summarizes the most recent exploration frameworks for the design of MPSoC architectures, while Section 3 describes the proposed modeling approach. Some experimental results have been reported and discussed in Section 4, while some concluding remarks and future developments of the research are outlined in Section 5.

2. Previous Work

To deal with the increasing complexity of MPSoCs, Transaction Level Modeling (TLM) has been introduced in the past years as a modeling style to describe on-chip communication channels at higher abstraction level with respect to Register Transfer Level (RTL). At TLM level, IP modules can be modelled at a functional level and the system bus behavior can be viewed as an abstract channel independent of the target bus architecture or protocol implementation. Although TL models offer high simulation speed, in some cases, they do not capture enough details about the on-chip behavior. Recent works ([11], [5], [8]) have been directed towards the application of some concepts from TLM level to BCA level to model on-chip communication architectures and to guarantee simulation speed-up.

A hierarchical modelling framework to support on-chip communication architectures has been proposed in [11], based on function calls instead of signal semantics to model AMBA2 and CoreConnect bus architectures. The approach in [5] models the AMBA2 bus at TL by using function calls for read/write operations, but also using SystemC clocked threads that can slow down simulation speed. The approach proposed in [8] also models AMBA2 read/write transactions but using low level handshaking semantics to support cycle-level accuracy. More recently, the TLM approach has been extended in [9] to propose a new and faster transaction-based modeling abstraction level (CCATB) to explore the communication design space. The CCATB abstraction level tries to bridge the gap between the TLM and BCA levels, thus yielding an average performance speed-up over pure BCA models.

Recently, commercial tools (such as [2], [4], [3]) have started to support system modelling at TLM, in addition to lower level RTL modelling tools. AHB Cycle-Level Interface Specification [1] has been released by ARM to define the requirements to model AHB at the cycle-accurate level in SystemC.

The *MPARM* multiprocessor simulation platform has been recently proposed in [7] for cycle-accurate power-performance estimation. The *MPARM* platform has been used in [6] for analyzing several on-chip communication

architectures (such as *AMBA* from ARM and *STBus* from STMicroelectronics).

All the above mentioned approaches are not able to fully exploit the trade-offs in terms of speed-up and accuracy of TLM with respect to BCA. To support these features, current TLM simulators do not offer a consistent way of switching abstraction level-modelling at run-time.

To overcome the limitations of previous approaches, we defined a flexible and efficient framework to support multi-level modeling and simulation for MPSoCs, that provides the capability of dynamically switching between a TLM description and a more accurate description of either a module or a communication channel.

3. Proposed Model

In this section, we show how we can exploit SystemC 2.0 TLM features together with Remote Procedure Call (RPC) and introspection to support multi-level models in order to trade-off simulation speed with accuracy at runtime. Multi-level models include both channels and generic modules.

Let us consider a channel: a TLM primitive channel provides one or more interfaces that can be accessed through *ports* by any *module*. A primitive channel can be refined into a *hierarchical channel*, i.e. a channel that can include processes, ports and also other modules and subchannels, in addition to interfaces. It is possible to wrap the model of a channel inside another channel. Multiple channels, modeled at different abstraction levels, can be merged into a single entity, allowing multi-level simulation as shown in Figure 2. In the following, we will refer to *channel* as the container of a set of *subchannels* that are models specified at different abstraction levels. Switching between subchannels introduces several issues:

- We need a mechanism to route requests to the correct channel, depending on the desired abstraction level
- We need to keep the system in a consistent state, avoiding any data loss during the switching of the abstraction level
- We need a mechanism to perform control-and-view of the abstraction level of each component at runtime

The first issue is the routing of the requests coming from the channel to the correct subchannel. As the channel interface method is called, the request is immediately sent to the currently active subchannel. To do so, each subchannel must present the same interface as the wrapping channel, or use an appropriate converter/adaptor solution. Considering a split-transaction bus model, this is done through the introduction of *Switchers* and *Mergers*: the former are bound to each master port of the channel and route transactions through the active channel, the latter get responses from both channels and route them to the correct destination, as

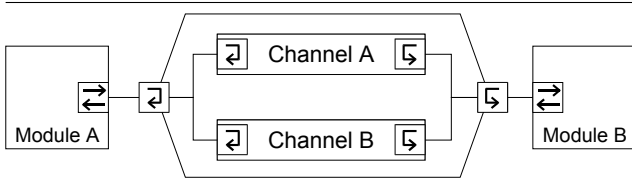


Figure 2. The architecture of a multi-level channel

shown in Figure 3. Switchers have a port directly bound to an external port for a connection to a master and the channel interface routes the packets to the correct switcher according to the transaction source. When the transaction response arrives from the subchannel, it is routed through the Switcher interface and forwarded to the external port, connected to the master external module, as shown in Figure 4. Mergers analogously receive requests from an interface, and route requests through their external ports, connected to the slave external modules.

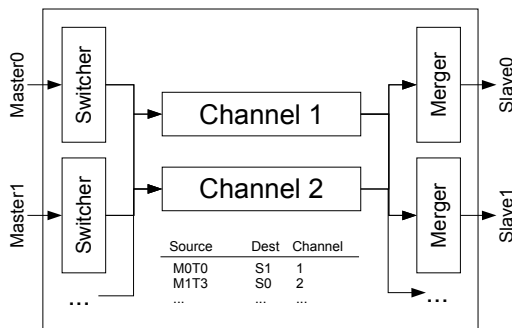


Figure 3. Internal architecture of a multi-level TLM split-transaction channel

To keep the system in a consistent state, each subchannel has to complete all the transactions that have been routed through it. The wrapping channel enforces this behavior by keeping track of all transaction requests: whenever the abstraction level is changed, the channel will start routing new requests to the appropriate subchannel, but all previous transactions will be still dealt by the previously selected subchannel. This is done by keeping a status request table (see Figure 3) and by associating each incoming request with a tag that specifies the subchannel that handled it. Routing of responses is performed according to the tag.

As an example, let us consider a processor P that is communicating with a memory M. Suppose that P is *multithreaded* and that communication is modeled as split-transaction and using two channels as described above. Sup-

pose also that P's first thread (T1) is reading data from memory and waiting for a result, while P's second thread (T2) is writing to another location in memory. Simulation is taking place at functional level. When T1 accesses the channel, the transaction is marked as belonging to the functional channel. Before the response to T1's read request is back to P, the user decides to change the accuracy level of the simulation to BCA. T2's transaction is routed into a different channel than T1's, and marked accordingly. When the memory is ready to return the read value to P, the information will travel in the *functional-level channel*, as T1's transaction was marked functional, while T2's write will be propagated through the BCA channel.

This means that channel models can be changed at any time without losing the state of the system, leaving the designer to decide which part of the simulation should be considered at an higher level of detail. Accuracy is lost at level-transition boundaries, when switching to a higher level of accuracy. The accuracy is affected by the channel models and the considered application. Assuming that switching points are sufficiently far to each other, we can suppose that after switching, the newly selected subchannel starts processing transactions in an empty state. To maintain accuracy, in the worst case, a number of transactions equal to the maximum contention of the subchannel has to be processed. In this way, it is possible to detect deadlocks or contention problems in the application.

Another issue concerns model control-and-view. In order to be able to change simulation speed and accuracy at runtime, we exploit the concept of introspection (from software engineering), using an interface definition language, SIDL. SIDL is a CORBA-like interface definition language that lets an external program perform remote method invocation (like Java RMI). By defining a standard interface for hot-switching, it is possible to determine or to change the abstraction level for each channel in the system. All these solutions have been implemented in a SystemC-based co-simulator, namely stepNP [10].

In practice, each TLM *entity* (i.e. a channel or a module) *extends* a SocObject class that provides methods and

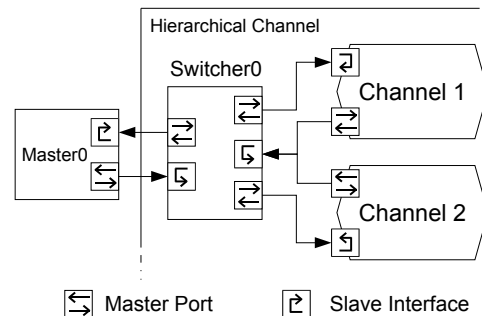


Figure 4. Architecture of a switcher

structures that export services to the processes executing out of the simulation space. These services let an external process read or modify the internal variables of the simulation. At the beginning of the simulation, each entity registers itself in an *Object Request Broker* (ORB). The ORB

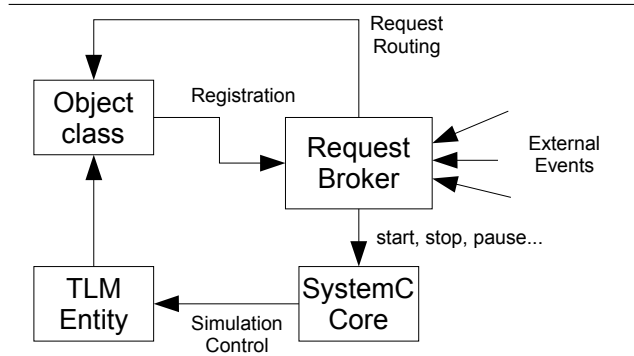


Figure 5. Extension to TLM to support introspection

acts as a name solver: it receives requests from the user or another program, and routes them to the correct TLM entity, as shown in Figure 5. The novelty with respect to a standard CORBA approach is that the ORB has to be synchronized with SystemC to avoid race conditions and inconsistent states in the simulator. When started, the ORB takes control of the simulation starting SystemC in a separate POSIX thread and uses *conditions* to pause the model when needed, without affecting simulation performance. To allow the access to variables and signals, we created a subclass of `sc_signal` called *probe*, that extends the Object class, and all `sc_signals` have been redefined as probes.

The implementation of generic multi-level modules is less straightforward: it is necessary to keep track of the module state, as it can be significantly different for each level of abstraction. As an example, a functionally modeled processor state contains much less information with respect to the same processor modeled at the internal bus signal level. It is not useful to keep two instances of the same module running at different levels of abstraction to maintain state consistency, as the slower model would be the bottleneck for simulation speed. Nevertheless, it is possible to increase simulation speed by de-synchronizing the module from the rest of the system. If we consider a standard event-driven simulation, clock distribution plays a noticeable role in the overall simulation complexity. Every time the module modifies some signals, it generates events that are put into a queue and scheduled by the simulation kernel. The system clock generates is responsible for generating most of these events. What we propose is the *internal* de-synchronization of a component, disregarding the sys-

tem clock and leaving each module to run at its own pace, in order to increase simulation speed. Synchronization is regained only *at module boundaries* when accessing interconnection channels.

This can lead to a substantial speed-up: in fact, running SystemC code without any clock wait state is like running native C++ code. By selectively removing `wait()` statements from SystemC code, but keeping internal synchronization, it is possible to trade-off simulation time with simulation accuracy. Models have to be properly designed to keep internal synchronization when not using the clock. This is done through the use of dynamic sensitivity lists and synchronization events (i.e. the `sc_event` structure). Dynamic sensitivity lists do not interfere with cycle-accurate simulation, but can be deactivated at runtime through SIDL to speed up cycle-accurate simulation. It is also possible to enforce synchronization every n clock cycles, triggering the `wait()` statement every n calls.

The activation of `wait()` statements is controlled by SIDL. This is implemented through a redefinition of the statements, using a value to determine the desired speed, i.e. the number of calls to the `wait()` before execution is actually suspended:

```

void multiLevelWait() {
    static int speed = DEF_SPEED;
    static int counter = 0;

    if( speed > 0 && counter < speed) {
        counter++;
        return;
    } else {
        counter = 0;
        wait();
    }
}
  
```

The model is synchronized only after a certain amount (*speed* in the code) of calls to the `wait` statement. This can be to eliminate synchronization completely. Whenever the module tries to access an external channel, synchronization with other modules is maintained by the TLM infrastructure: transactions (split transactions) are blocking for the module (thread) that generated them.

In our experiments, the model of a multi-threaded ARM can double its speed when synchronizing every 100,000 cycles. We implemented a cycle-accurate Instruction Set Simulator (ISS) inside a SystemC wrapper, synchronizing to the clock only on request, and leaving it to run at full speed when simulation accuracy is not needed, as shown in Figure 6. This approach is different from channel models in that it does not have two different objects wrapped in a single module. Rather, the module is appropriately disconnected from the clock signal. Hence, there is no need to keep track of the state of the module, nor any routing involved in inter-

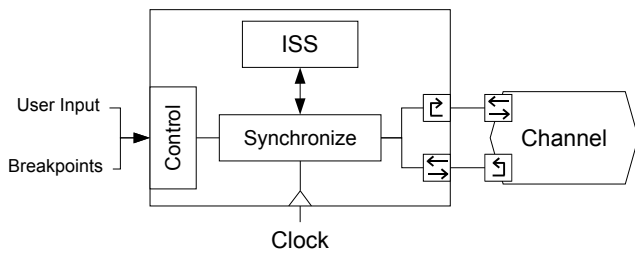


Figure 6. A multi-level module, with SIDL control-and-view

face calls, provided that the module can keep internal synchronization.

4. Experimental Results

The proposed modeling approach has been validated using the stepNP simulation platform [10] with several benchmarks and a large multimedia application, namely a MPEG4 encoder. The chosen benchmarks are: VMUL, a parallel vector multiplication application that exploits both multi-cycle ARM instructions and accesses to memory; PI, a parallel π calculator that makes full use of the processors computing ability; MemTest, a parallel mass memory read/write application; Sort, a sorting application that exploits bandwidth and uses little computing power; and MPEG4, a fully fledged MPEG4 encoder, used to prove the methodology on a real industrial application. MemTest and Sort are good benchmarks for the channel switching methodology because they fully use the bandwidth offered, introducing a certain amount of contention, and can show the accuracy trend when switching between two channels. PI and VMUL are appropriate to test the de-synchronization of the ARM processor because they focus on computing more than communication. Experiments considered the simulation of an MPSoC. The target architecture is formed by 5 ARM processors, two levels of caches, local memories for each processor, an external bus and a scratchpad. All components are connected by the STMicroelectronics interconnection network, STBus. All components are modeled at the timed functional level, except for the processors and STBus.

Concerning channel modeling, the BCA model of the STBus interconnection network in crossbar mode (supplied by STMicroelectronics) has been simulated in parallel with a functional crossbar. Figure 7 shows the simulation speed of the aforementioned benchmarks when switching to Timed Functional (TF) and Untimed Functional (UTF) simulation, relative to Bus-Cycle-Accurate simulation. The speed approximately doubles for the selected benchmarks when switching from BCA to TF and the speed-up almost reaches an order of magnitude when UTF is used. The overhead introduced by the switchers and mergers is negligible when

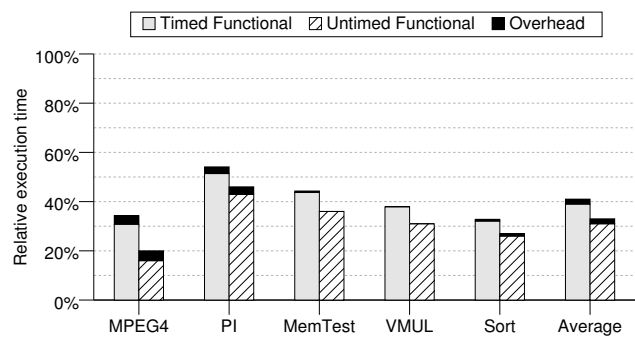


Figure 7. Simulation time of functional channels compared to BCA ones, including the overhead of switchers and mergers

compared to the actual simulation lengths, as shown in Figure 7. It is worth noting that the greater the use of the communication channel by the application, the greater the difference between the functional and BCA models.

Figure 8 shows the effect of the accuracy change; TF simulation of channels is less accurate than Bus-Cycle (BC), but error remains at a low average 27% that can be traded off for increased speed. UTF is not considered since it disregards accuracy completely in favor of fast simulation.

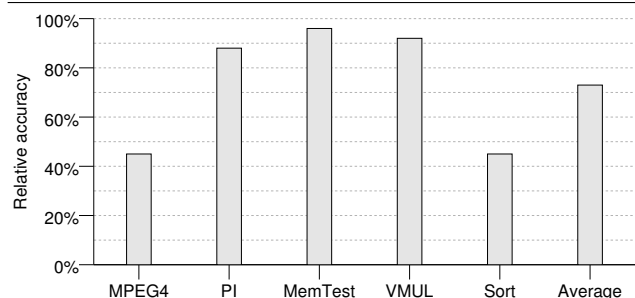


Figure 8. Timed Functional simulation accuracy, as compared to Bus-Cycle accuracy

A second set of experiments was done by using a boundary synchronized ARM processor module, i.e. synchronization is done only when the ARM accesses the communication channel. Results are shown in Figure 9. There is no noticeable overhead introduced when de-synchronizing the ARM modules. Accuracy decreases, but the error is biased negatively, and a correction factor can be introduced. This is due to the fact that the processors are not synchronized and most cycles are not considered.

Extensive experimentation has been done to determine the accuracy lost at the abstraction level boundary. Accu-

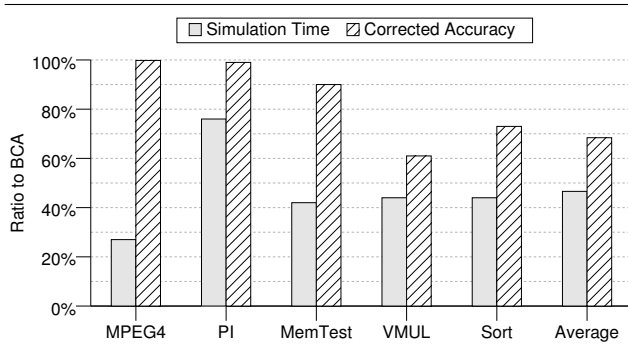


Figure 9. Comparison of simulation performance for de-synchronized ARM processor modules, as compared to BC accuracy

accuracy switching was performed on all the benchmarks, in different sections of the code. Results show that no accuracy is lost, and sections executed in BCA mode maintain clock-cycle accuracy. Finally, putting together both channel and

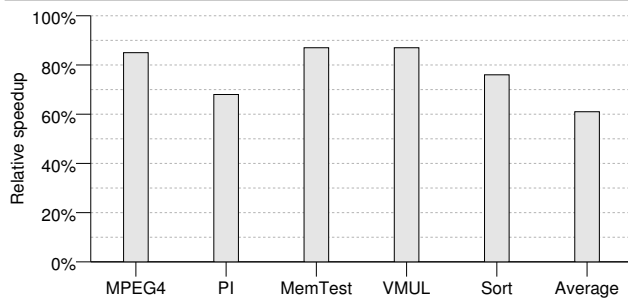


Figure 10. Relative speed-up when applying both methodologies

module methodologies, it is possible to save up to one order of magnitude in simulation speed, as shown in Figure 10 with negligible overhead.

5. Conclusions

In this paper, a methodology based on the exploitation of TLM and the concept of introspection has been proposed to support multi-level system models for the simulation of complex MPSoCs. Multi-level models include both channels and generic modules of the system. In particular, to speed up the simulation and debugging phases of small kernels of the target application, we have proposed a modeling and simulation methodology to use less accurate system models for the uninteresting sections of the simulation and accurate system models for the kernels under analysis. To increase simulation speed, while maintaining

multi-level simulation capabilities, we have proposed to de-synchronize a single module from the rest of the system, disregarding all synchronization events and leaving each module to run at its own pace. In this way, a system model can run at several levels of accuracy, depending on how often the module is synchronized with the system.

The modeling approach proposed in this paper has been validated using the stepNP simulation platform while executing a set of applications. Further research activities are directed towards the support of the hardware/software co-exploration phase.

References

- [1] AHB CLI Specification. <http://www.arm.com/armtech/ahbcli>.
- [2] Cadence NCSysmC. <http://www.cadence.com>.
- [3] CoCentric System Studio. <http://www.synopsys.com>.
- [4] CoWare. <http://www.coware.com>.
- [5] M. Caldari, M. Conti, M. Coppola, S. Curaba, L. Pieralisi, and C. Turchetti. Transaction-level models for amba bus architecture using systemc 2.0. In *DATE '03: Proceedings of the conference on Design, Automation and Test in Europe*, page 20026, Washington, DC, USA, 2003. IEEE Computer Society.
- [6] M. Loghi, F. Angiolini, D. Bertozzi, L. Benini, and R. Zafalon. Analyzing on-chip communication in a mpsoe environment. In *Proceedings of the conference on Design, automation and test in Europe*, page 20752. IEEE Computer Society, 2004.
- [7] M. Loghi, M. Poncino, and L. Benini. Cycle-accurate power analysis for multiprocessor systems-on-a-chip. In *Proceedings of the 14th ACM Great Lakes symposium on VLSI*, pages 410–406. ACM Press, 2004.
- [8] O. Ogawa, S. B. de Noyer, P. Chauvet, K. Shinohara, Y. Watanabe, H. Niizuma, T. Sasaki, and Y. Takai. A practical approach for bus architecture optimization at transaction level. In *DATE '03: Proceedings of the conference on Design, Automation and Test in Europe*, page 20176, Washington, DC, USA, 2003. IEEE Computer Society.
- [9] S. Pasricha, N. Dutt, and M. Ben-Romdhane. Extending the transaction level modeling approach for fast communication architecture exploration. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 113–118, New York, NY, USA, 2004. ACM Press.
- [10] P. G. Paulin, C. Pilkington, and E. Bensoudane. StepNP: A system-level exploration platform for network processors. *IEEE Design and Test of Computers*, pages 2–11, 2002.
- [11] X. Zhu and S. Malik. A hierarchical modeling framework for on-chip communication architectures. In *ICCAD '02: Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 663–671, New York, NY, USA, 2002. ACM Press.