Chapter 11

Transaction Level Model Automation for Multicore Systems

Lucky Lo Chi Yu Lo
University of California Irvine, USA

Samar Abdi
University of California Irvine, USA

Daniel Gajski
University of California Irvine, USA

ABSTRACT

Model based verification has been the bedrock of electronic design automation. Over the past several years, system modeling has evolved to keep up with improvements in process technology fueled by Moore’s law. Modeling has evolved to keep up with the complexity of applications resulting in various levels of abstractions. The design automation industry has evolved from transistor level modeling to gate level and eventually to register transfer level (RTL). These models have been used for simulation based verification, formal verification and semiformal verification. With the advent of multicore systems, RTL modeling and verification are no longer feasible. Furthermore, the software content in most modern designs is growing rapidly. The increasing software content, along with the size, complexity and heterogeneity of multicore systems, makes RTL simulation extremely slow for any reasonably sized system. This has made system verification the most serious obstacle to time to market. The root of the problem is the signal-based communication modeling in RTL. In any large design there are hundreds of signals that change their values frequently during the execution of the RTL model. Every signal toggle causes the simulator to stop and reevaluate the state of the system. Therefore, RTL simulation becomes painfully slow. To overcome this problem, designers are increasingly resorting to modeling such complex systems at higher levels of abstraction than RTL. Transaction level models (TLMs) have emerged as the next level of abstraction for system design. However, well defined TLM semantics are needed for design automation at the transaction level. In this chapter, we present transaction level model automation for multicore systems based on well defined TLM semantics. TLMs replace the traditional signal toggling model of system communication with function calls, thereby increasing simulation speed. TLMs are
already being used for executable specification of multicore designs, for analysis, fast simulation, and debugging. They play an important role in early application development and debugging before the final prototype has been implemented. We discuss essential issues in TLM automation and also provide an understanding of the basic building blocks of TLMs.

**INTRODUCTION**

The rise in complexity, size and heterogeneity of modern embedded system designs has pushed modeling to new abstraction levels above RTL. Transaction level modeling using System-Level Design Languages (SLDL) such as SpecC (Gajski, Zhu, Doemer, Gerstlauer & Zhao, 2000; Gerstlauer, Domer, Peng & Gajski, 2001) or SystemC (OSCI, 2007) is emerging as a new paradigm for system modeling. But the transaction level modeling’s concept still has not been fully standardized in the industry. Different people have different notions of how TLMs should appear, both syntactically and semantically. This is because the original TLM definition did not provide any specific structure or semantics. However, the argument for establishing standards in TLMs is a very strong one. The reason is that without standards there is no possibility for sharing of models, for having common synthesis and analysis tools and so on. Ad hoc transaction level modeling may seem attractive for having fast simulation speed for a specific design, but that approach is not conducive to establishing TLM as a viable modeling abstraction like RTL.

**TLM BACKGROUND**

In 2003, a paper on establishing taxonomy for TLMs was published (Cai & Gajski, 2003) that opened up the debate on what are the useful system level models and how to position TLMs as an abstraction above RTL. The taxonomy was based on the granularity of detail in modeling the computation and communication for systems with multiple processing elements.

In a TLM, the details of communication amongst computation components are separated from the details of computation components themselves. Communication is modeled by channels, which are simply a repository for communication services. This is very similar to a class in C++. In fact, SystemC uses C++ classes to implement channels. The channel communication services are used by transaction requests that take place by calling interface functions of these channels. Unnecessary details of communication and computation are hidden in a TLM and may be added later in the design process. TLMs speed up simulation and allow exploring and validating design alternatives at a higher level of abstraction. However, the definition of TLMs is not well understood. Without clear definition of TLMs, any predefined TLMs cannot be easily reused. Moreover, the usage of TLMs in the existing design domains, namely modeling, validation, refinement, exploration, and synthesis, cannot be systematically developed. Consequently, the inherent advantages of TLMs don’t effectively benefit designers. In order to eliminate some ambiguity of TLMs, several TLMs are defined, each of which may be adopted for different design purpose.

In order to simplify the design process, designers generally use a number of intermediate models. The intermediate models slice the entire design process into several smaller design stages, each of which has a specific design objective. Since the models can be simulated and estimated, the result of each of these design stages can be independently validated. In order to relate different models, the