Transaction-Level Modeling Definitions and Approximations
EE290A Final Report
Trevor Meyerowitz
May 20, 2005

1. Introduction

Over the years the field of electronic design automation has enabled gigantic increases in design size and complexity by raising the level of abstraction for design. Thus design entry has proceeded from polygons, to transistors, to gates, and then to RTL languages such as Verilog and VHDL. For at least the past decade the search for a level of abstraction higher than RTL has taken place, leading to the emergence of the field of System Level Design. This area hasn’t been nearly as well defined as the other areas in design automation because these levels of abstraction include full systems and are often domain-specific in nature. Furthermore, there are many terms being used that have vague or multiple definitions. One of the more popular of these definitions is transaction-level modeling (TLM). In this report we survey recent definitions of TLM, and present an in depth view of levels of modeling for microprocessors.

2. Definitions of Transaction-Level Modeling

The definition transaction level modeling is one issue, and another issue is what it means. Most definitions [SYSTEMC, SPECC-TLM, SYSC-TLM] define transaction level models as models where the communication and computation of systems are separated, and the communication is implemented as function calls instead of signals. There are many types of models that fit this description, and yet are totally different in their characteristics. One of the first definitions that fits this description is Interface-Based Design [INTERFACE], where the authors advocate the separation of behavior and communication.

2.1. SpecC Definitions

SpecC [SPECC] is a system-level design language and environment developed at UC Irvine. It is a superset of the C language, and adds constructs for concurrent specification, and hardware modeling. Its core model of computation is called the Finite State Machine with Datapath (FSMD). It has paths to both hardware (by high level synthesis) and software by removing or translating the SpecC constructs from the mapped portion of the specification to yield regular C.

Figure 1: System Modeling Graph
(Copyright ©2003 Dan Gajski and Lukai Cai)
In [SPECC-TLM] Cai and Gajski define different levels of transaction level models in terms of the level of representation of the computation and communication. In particular they define three levels of abstraction for these things, untimed, approximate-timed, and cycle-timed. They also lay out different design points that make sense for the design methodology. A summary of these points is shown in figure 1. In the next section we build upon these definitions for the case of modeling microprocessor performance.

2.2 SystemC Definitions

SystemC [SYSC] is a popular system level design tool. It supports system modeling through a library of C++ API’s including a discrete-event execution kernel, low level signals, high level channels, and a variety of RTL-like functionality.

In [SYSC-TLM] Donlin presents an overview of different levels of transaction-level modeling and various use-cases for the models. In particular, he presents two classes of models: Communicating Processes and Programmer’s View. These models can be untimed or timed at various levels of accuracy. These models represent the official SystemC transaction-level modeling guidelines that were recently released at www.systemc.com.

2.2.1 Communicating Processes

Communicating Processes are concurrent processes (shown in the shaded boxes) communicating by point-to-point connections (shown in the rounded unshaded boxes). The example in figure 2 shows a producer communicating via fifo channels with two consumer processes. This view has no notion of resource sharing or the underlying implementation of the system. It is useful for high-level algorithm design and the modeling of hardware.

2.2.2 Programmer’s View
The Programmer’s View is typically at a lower level of abstraction than the Communicating Processes view. It is a model that corresponds to the physical implementation of the system. Resource arbitration, programmability, and state are all visible. This model is register accurate, and can be used to develop software for the platform. Figure 3 shows an example of a programmer’s view that has a CPU and DSP communicating via a shared bus to shared memory.

### 2.3 Metropolis Metamodel

The Metropolis metamodel [METRO] is a system-level design framework being developed by our research group. Like SpecC and SystemC it has explicit separation between communication and computation, but it also has separation between functionality and architecture. Figure 4 shows a mapped netlist with functionality on the left side and architecture on the right hand side. Events on the functional side are synchronized with those on the architectural side through mapping processes. The events in the mapping processes make calls to media representing the architectural resources. The scheduling of these resources is controlled by quantity managers that are represented by the diamond-shaped boxes.

### 2.4 Other Definitions

#### 2.4.1. Calypto

Calypto [CALYPTO] is a company that provides sequential equivalence checking. They define transactions as points at which the two models will be checked for equivalence, making it a more computational definition.
2.4.2 The Open Core Protocol

The Open Core Protocol [OCP, OCP-OSCI] defines their transaction-level models of different levels of modeling protocols at 4 different levels of abstraction. Level 0 is the RTL layer. Level 1 is the transfer layer which abstracts away particular signals but is still cycle accurate, Level 2 is the transaction layer which abstracts away implementation protocols and explicit clocks, and Level 3 is the message layer where resource sharing and timing are abstracted away.

2.4.3 CCATB

Pasricha and Dutt defined an improved performance transaction level model called Cycle Count Accurate at Transaction Boundaries [CCATB]. Performance is improved by passing the timing information along with the transaction token and only update the timing once the transaction has finished instead of at every point of the transaction (note: transactions may involve multiple function calls). The trade-off here is that intra-transaction delays are no longer visible, making debugging harder.

3. Microprocessor Models

A key element of embedded system design is software executing on one or more microprocessors. Often it is necessary to start implementing the software before the full platform has been built. This leads to the need to model the software together with a higher level model of the system architecture (such as the Programmer’s View).

Based on the requirements for timing accuracy and simulation speeds of the processor model system architects select an appropriate level of abstraction to model software executing on the processor. Figure 5 shows the classical continuum of models plotted along a curve, and also a newer approach called timing annotation. Timing annotation writes back results of the software executing on a lower-level model to a higher level model (such as the original source code), to allow better accuracy without sacrificing significant simulation speed.

The fastest performance is the program compiled on the user’s computer. To measure the number of instructions used for the program an Instruction Set Simulator
(ISS) will be used. An ISS sequentially executes the instructions and has no notion of concurrency of microarchitecture. Cycle-Accurate Simulators simulate the microarchitecture at the clock-cycle level, and are by far the most common type of simulator used. If further accuracy is needed, then RTL-level models can be used. These would typically just be used for testing interfaces and race conditions because they are too slow to meaningfully execute software.

3.1 Definitions of Problem Space

The processor microarchitecture is defined by a function $\text{ExecTrace}$ that yields $\text{AppTrace}$ a trace of the commit times for the instructions of a given application trace, and optionally might have the memory access times. $\text{ExecTrace}$ is based on the application trace ($\text{App}$), the communication state ($\text{CommS}$), and the computation state ($\text{CompS}$). Each of these can be modeled exactly, approximately, or ideally (i.e. no delay). Below is the definition of $\text{ExecTrace}$.

$$\text{ExecTrace} : \text{App} \times \text{CommS} \times \text{CompS} \rightarrow \text{AppTrace}$$

$$\text{AppTrace} = \{ (\text{commit}(i), i) \mid \text{for All instructions } i \text{ in App} \}$$

$\text{App}$: Application trace is the sequence of instructions to be executed. Its state is the program counter, and possibly the status of pending instructions.

$\text{CompS}$: The computation state consists of the processor pipeline state, the execution resource states, internal buffers (e.g. reservation stations).

$\text{CommS}$: The communication state consists of the memory hierarchy, the bus interfaces, and any sort of buffers associated with the memory system. It is a function of the passage of time and modifications to it from $\text{CompS}$.

3.2 Communication System Definitions

For the communication system we examine both the memory hierarchy as well as the interconnection system. The memory hierarchy consists of memories, caches, and buffers. The interconnection system consists of various channels and buses along with different arbitration policies. The performance of these different systems can be nicely described as partial orders with ideal performance on the left side, and the worst-case performance on the right side. Figures 6 and 7 show these partial orders for the memory hierarchy and the methods of interconnection.

![Figure 6: Memory Partial Order](image-url)
Other characteristics of the communication system include:

- Increasing cache size and associativity increases performance.
- Adding buffers increases performance.
- Replicating busses and increasing their speed and/or throughput increases performance.

### 3.2 Computation System Elements

In this section we list the different elements found in the processor microarchitecture and their impact on performance.

- **Resources**: This is anything that is potentially shared between instructions and could cause one instruction to delay its execution. These resources include execution units and register files. Each resource has a latency and throughput, usually defined via clock-cycles. These are general pieces and are also on the communication side. Decreasing the latency and/or increasing the throughput of resources increases performance. Duplicating resources will generally increase performance.

- **Physical Pipeline**: represents the delays between resources instruction path. Increasing pipeline length improves the cycle time of a processor, but results in greater penalties for structural hazards and increased cycle counts for programs.

- **Buffers**: Stores pending requests for resources (be they computation or communication), to avoid stalling (e.g. reservation stations). Buffers generally increase performance.

- **Scalarity**: This is a property related to the parallelism present in any stage of the computation process (fetch, issue, dispatch). It allows greater throughput and enables speculation.

- **Speculation**: This is a property that involves making a choice before there is enough knowledge to know if it is correct. Adding speculation usually produces an increase in performance, but it can degrade performance if it causes the communication system to be flooded with incorrect speculations.

### 4. Final Words

In this report we’ve explained a variety of definitions for transaction-level modeling as well as different levels of abstraction for modeling the performance of embedded system software. Our research focus is in this area developing high level models for microarchitectures and examining the performance relationships between these different levels of abstraction as well as exploring the mixing between levels (e.g. performance back annotation with partial cache/communication-system simulation).
References

[CALYPTO] Calypto Company Web Site: http://www.calypto.com


[OCP] Open Core Protocol Web Site: www.ocpip.org


