Chapter 2

TLM-BASED METHODOLOGY

This chapter examines a methodology that enables you to model your large system designs at higher level of abstraction and realize actual productivity gains offered by SystemC.

2.1 Transaction-Level Modeling Overview

In the past, when many systems were a more manageable size, a system could be grasped by a single person known by a variety of titles such as system architect, chief engineer, lead engineer, or project engineer. This guru may have been a software engineer, hardware engineer, or algorithm expert depending on the primary technology leveraged for the system. The complexity was such that this person could keep most or all of the details in his or her head, and this technical leader was able to use spreadsheets and paper-based methods to communicate thoughts and concepts to the rest of the team.

The guru's background usually dictated his or her success in communicating requirements to each of the communities involved in the design of the system. The guru's past experiences also controlled the quality of the multi-discipline trade offs such as hardware implementation versus software implementation versus algorithm improvements.

In most cases, these trade offs resulted in conceptual disconnects among the three groups. For example, cellular telephone systems consist of very complex algorithms, software, and hardware, and teams working on them have traditionally leveraged more rigorous but still ad-hoc methods.

These methods usually consist of a software-based model; sometimes called a system architectural model (SAM), written in C, Java, or a similar language. The model is a communication vehicle between algorithm, hardware, and software groups. The model may be used for algorithmic refinement or used as basis for deriving hardware and software subsystem specifications. The exact parameters modeled are specific to the system type

and application, but the model is typically un-timed (more on this topic in the following section). Typically, each team then uses a different language to refine the design for their portion of the system. The teams leave behind the original multi-discipline system model and in many cases, any informal communication among the groups.

With rapidly increasing design complexity and the rising cost of failure, system designers in most product domains will need a similar top-down approach but with an improved methodology. An emerging system design methodology based on Transaction-Level Modeling (TLM) is evolving from the large system design methodology discussed above. This emerging methodology has significantly more external and project design reuse enabled by a language like SystemC.

Transaction-level modeling is an emerging concept without precise definitions. A working group of Open SystemC Initiative (OSCI) is currently defining a set of terminology for TLM and will eventually develop TLM standards. In reality, when engineers talk of TLM, they are probably talking about one or more of four different modeling styles that are discussed in the following section.

The underlying concept of TLM is to model only the level of detail that is needed by the engineers developing the system components and sub-system for a particular task in the development process. By modeling only the necessary details, design teams can realize huge gains in modeling speed thus enabling a new methodology. At this level, changes are relatively easy because the development team has not yet painted itself into a corner with low-level details such as a parallel bus implementation versus a serial bus implementation.

Using TLMs makes tasks usually reserved for hardware implementations practical to run on a model early in the system development process. TLM is a concept independent of language. However, to implement and refine TLM models, it is helpful to have a language like SystemC whose features support independent refinement of functionality and communication that is crucial to efficient TLM development.

Before exploring this new design methodology we will explore some of the background and terminology around TLM.

2.2 Abstraction Models

Several sets of terminology have been defined for the abstraction levels traditionally used in system models. We are presenting a slight variation of a model developed and presented by Dan Gajski and Lukai Cai at CODES (HW/SW Co-Design Conference) 2003 that is illustrated in *Figure 2-1*.

The first concept necessary for understanding TLM is that system and sub-system communication and functionality can be developed and refined independently. In this terminology, the communication and functionality components can be un-timed (UT), approximately-timed (AT), or cycle-timed (CT).

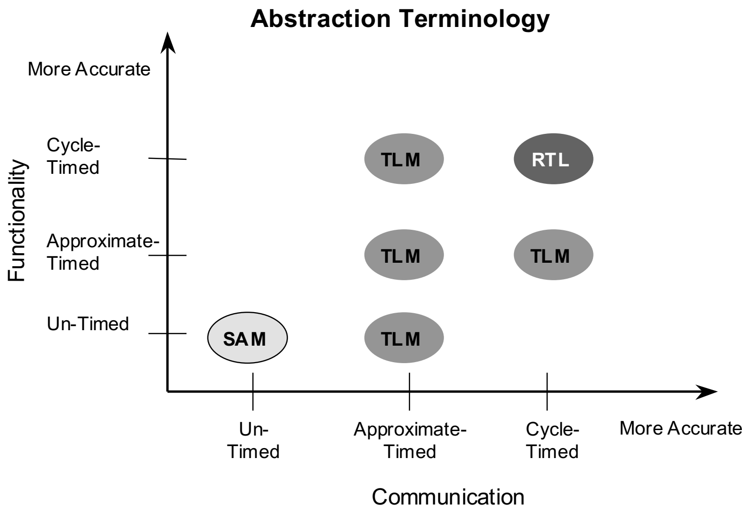

Figure 2-1. Abstraction Terminology

A model that is cycle-timing accurate for communication and for functionality is usually referred to as a register-transfer level (RTL) model. We refer to models with un-timed communication and functionality as a SAM. The RTL model is traditionally used for automatic synthesis to gates. Many times the SAM is used for algorithmic refinement and can be refined to approximately-timed communication and/or functionality.

The other four points plotted on the graph are usually collectively referred to as TLMs, and rely on approximately-timed functionality or communication. Approximately-timed models can rely on statistical timing,

estimated timing, or sub-system timing requirements (or budgets) derived from system requirements.

A model with cycle-timed communication and approximately-timed functionality has been referred to as a Bus Functional Model(BFM) in older methodologies and the label is retained here. The three remaining TLMs have not yet developed commonly accepted names. For now, we will use the names developed by Gajski and Cai.

Table 2-1. Timing of Transaction-Level Models

Model	**Communication**	**Functionality**
SAM	UT	UT
Component assembly	UT	AT
Bus arbitration	AT	AT
Bus functional	CT	AT
Cycle-accurate computation	AT	CT
RTL	CT	CT

All of these models are not necessary for most systems. In reality, most systems only need to progress through two or three points on the graph in Figure 2-1. With a language that supports refinement concepts, the transformation can be quite efficient.

2.3 Another Look at Abstraction Models

In this section, to build out your understanding of how TLM can be useful, we present a less rigorous and more example-based discussion of TLM. We will assume a generic system containing a microprocessor, a few devices, and memory connected by a bus.

The timing diagram in *Figure 2-2* illustrates one possible design outcome of a bus implementation. When first defining and modeling the system application, the exact bus-timing details do not affect the design decisions, and all the important information contained within the illustration is transferred between the bus devices as one event or transaction (component-assembly model).

Further into the development cycle, the number of bus cycles may become important (to define bus cycle-time requirements, etc.) and the information for each clock cycle of the bus is transferred as one transaction or event (bus-arbitration or cycle-accurate computation models).

When the bus specification is fully chosen and defined, the bus is modeled with a transaction or event per signal transition (bus functional or RTL model). Of course, as more details are added, more events occur and the speed of the model execution decreases.

In this diagram, the component assembly model takes 1 "event," the bus arbitration model takes approximately 5 "events," and the RTL model takes roughly 75 "events" (the exact number depends on the number of transitioning signals and the exact simulator algorithm). This simple example illustrates the magnitude of computation required and why more system design teams are employing a TLM-based methodology.

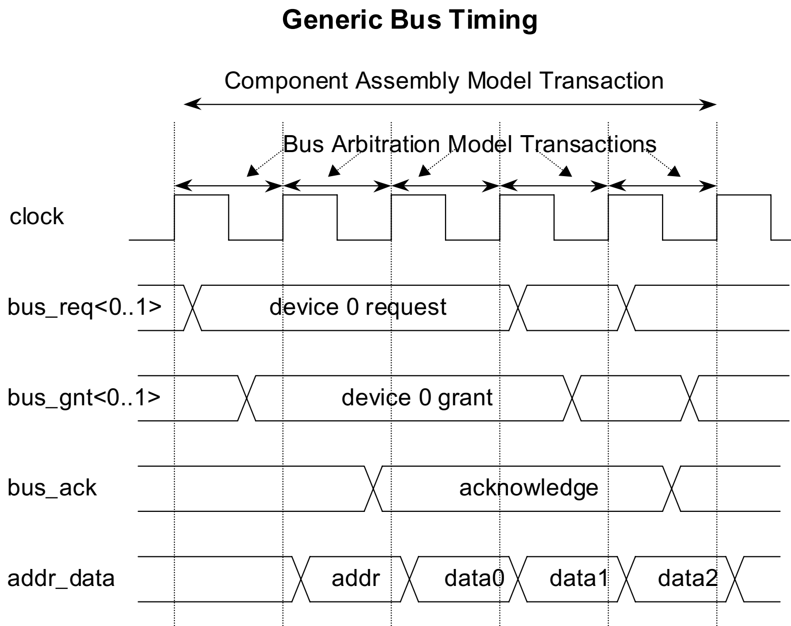

Figure 2-2. Generic Bus Timing Diagram

2.4 TLM-Based Methodology

Now that we have discussed some of the TLM concepts, we can look more closely at a TLM-based methodology as illustrated in *Figure 2-3*.

In this methodology, we still start with the traditional methods used to capture the customer requirements, a paper Product Requirements Document (PRD). Sometimes, the product requirements are obtained directly from a customer, but more likely the requirements are captured through the research of a marketing group.

From the PRD, a SAM is developed. The SAM development effort may cause changes or refinement to the PRD. The SAM is usually written by an architect or architecture group and captures the product specification or system critical parameters. In an algorithmic intensive system, the SAM will be used to refine the system algorithms.

The SAM is then refined into a TLM that may start as a component assembly type of TLM and is further refined to a bus arbitration model. The TLM is refined further as software design and development and hardware verification environment development progresses.

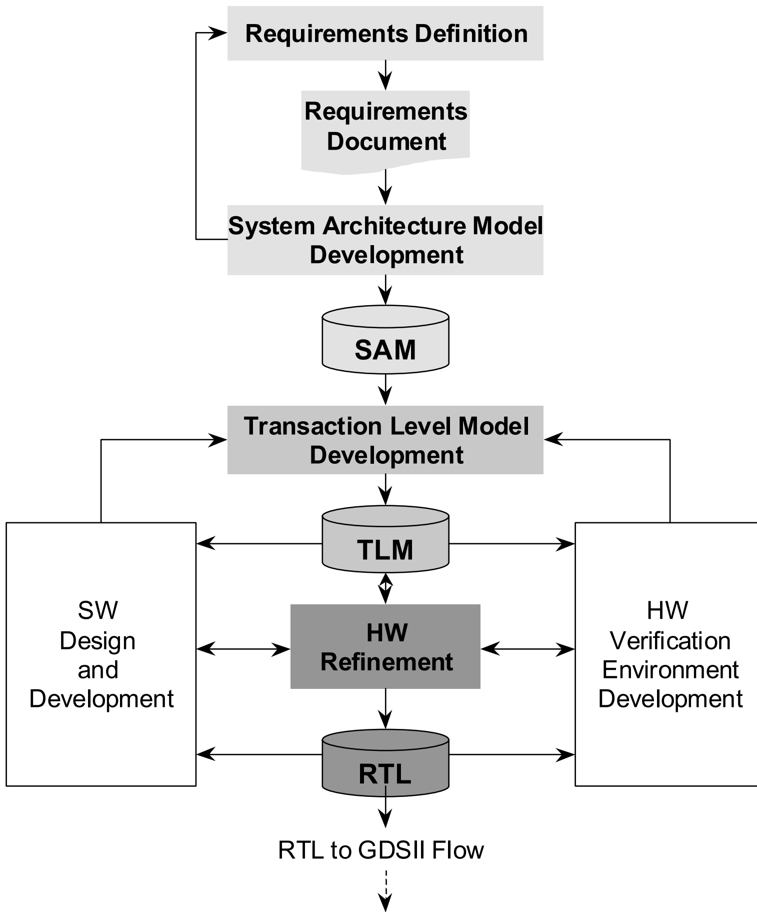

Figure 2-3. TLM-Based Flow

If the proper design language and techniques are used consistently throughout the flow, then the SAM can be reused and refined to develop the TLM. The TLM has several goals:

1. Refinement of implementation features such as HW/SW partitioning; HW partitioning among ASICs, FPGAs, and boards; bus architecture exploration; co-processor definition or selection; and many more
2. Development platform for system software
3. "Golden Model" for the hardware functional verification
4. Hardware micro-architecture exploration and a basis for developing detailed hardware specifications

In the near future, if EDA tools mature sufficiently, the TLM code may be refined to a behavioral synthesis model and be automatically converted to hardware from a higher-level abstraction than the current RTL synthesis flows. Today, the hardware refinement is likely done through a traditional paper specification and RTL development techniques, although the functional verification can now be performed via the TLM as outlined later in this chapter.

At first, development of the TLM appears to be an unnecessary task. However, the TLM creates benefits including:

- Earlier software development
- Earlier and better hardware functional verification test bench
- Creates a clear and unbroken path from customer requirements to detailed hardware and software specifications

After reading this book, you and your team should have the knowledge to implement TLMs quickly and effectively. The following section discusses in detail the benefits your team will bring to your organization when applying this methodology: early software development and early hardware functional verification.

2.4.1 Early Software Development

In complex systems where new software and new hardware are being created, software developers must often wait for the hardware design to be finalized before they can begin detailed coding. Software developers must also wait for devices (ICs and printed circuit boards) to be manufactured to test their code in a realistic environment. Even then, creating a realistic environment on a lab workbench can be very complex. This dependency creates a long critical path that may add so much financial risk to a project that it is never started.

Figure 2-4 illustrates a traditional system development project schedule. The arrows highlight differences a TLM-based methodology would make. The time scale and the duration of each phase depend on the project size, project complexity, and the makeup of the system components (hardware, software, and algorithms).

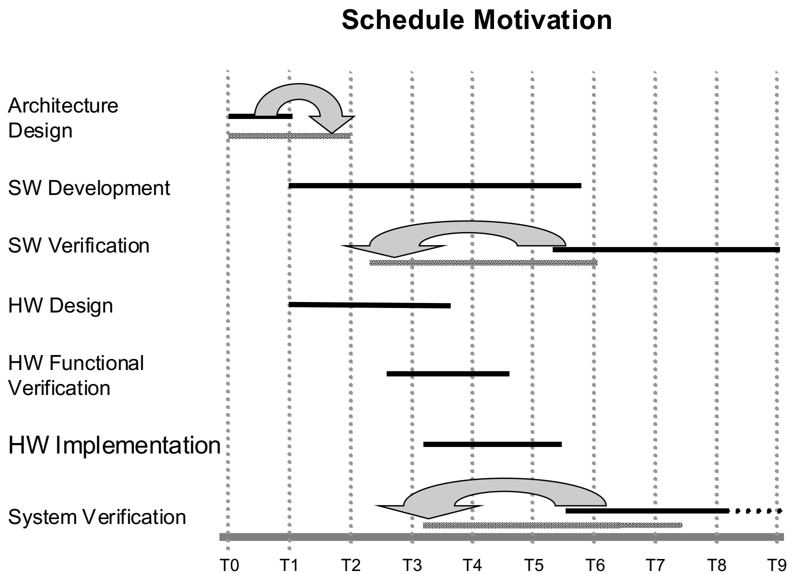

Figure 2-4. Schedule Benefits of Earlier Software Development

Creating a TLM from the SAM slightly lengthens the architectural design phase of a project, but it offers several potential benefits:

- Ability to start refining and testing software earlier, thereby reducing the overall development cycle
- Ability to provide earlier and more realistic hardware/software trade off studies at a time when changes are easier, thus improving overall system quality
- Ability to deliver executable models to customers both for validating the specification and driving changes, and acceleration of product adoption
- Ability to cancel (or redefine) an unrealistic project before spending even larger sums of money

Any opportunity to begin the software development work earlier warrants consideration. Indeed, the bottom line financial returns for just starting software development earlier, may dictate the adoption of this new methodology without the other benefits listed above.

2.4.2 Better Hardware Functional Verification

System design teams are always looking for ways to provide more and better functional verification of the hardware. The number of cases required to functionally verify a system is growing even faster than the actual system complexity.

Verifying the hardware interaction with the actual software and firmware before creating the hardware is becoming increasingly more important. With the chip mask set costs exceeding several hundred thousand dollars, finding out after making chips that a software workaround for the hardware is impossible or too slow is not acceptable. As a result, many teams are developing simulation and emulation techniques to verify the hardware interaction with the software and firmware.

Additionally, with the increase in size and complexity of the hardware, it is increasingly important to verify that unforeseen interactions within the chip, between chips, or between chips and software do not create unacceptable consequences. Debugging these interactions without significant visibility into the state of the chip being verified is very tough.

Very large Verilog or VHDL simulations along with emulation strategies have traditionally been used for system-level functional verification. With increasing system complexity, Verilog and VHDL simulations have become too slow for such verification. Hardware emulation techniques have been

used when simulation has been too slow, but emulation techniques often have limited state visibility for debugging, and they can be very expensive.

When a design team develops a TLM, it is straightforward to refine the model to a verification environment through the use of adapters as outlined in the following section.

2.4.3 Adapters and Functional Verification

This section is a very brief overview of how a TLM model can be used as part of an overall system functional verification strategy. With modern systems, the hardware design is not fully debugged until it is successfully running the system software. This approach enables functional verification of the hardware with the system software prior to hardware availability. More details about implementation of this approach are given in Chapter 13, Custom Channels, and other sources[4].

To show one way that adapters can be applied to a TLM to create a verification environment, we will assume a generic system that looks like *Figure 2-5*. The generic system is composed of a microprocessor, memory, a couple of devices, and a bus with an arbiter.

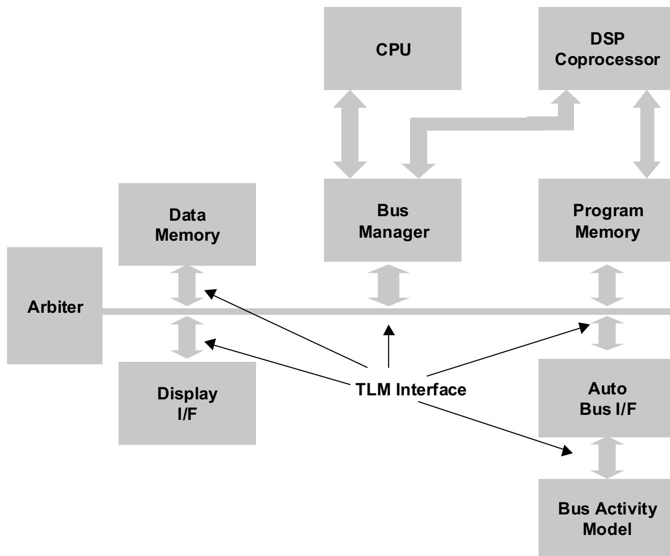

Figure 2-5. Generic System

[4] Grotker, T., Liao, S., Martin, G., Swan, S. 2002. *System Design with SystemC*. Norwell Massachusetts: Kluwer Academic Publishers.

For our discussions, we will concentrate on communication refinement and assume that the functionality of the devices, the memory, and the microprocessor will be approximately-timed or cycle-timed as appropriate throughout the design cycle.

In this very simple example, we assume that RTL views of the microprocessor and memory are not available or not important at this point in the verification strategy. In this case, the RTL for the two devices could be functionally verified by insertion of an adapter as illustrated in *Figure 2-6*.

This approach dictates that the adapter converts the timing-accurate signals of the bus coming from the RTL to a transaction view of the bus. The RTL sees the bus activity that would be created by the microprocessor, memory, and arbiter. Bus activity is propagated only to the non-RTL portion of the system after the adapter creates the transaction. This propagation creates a very high performance model compared to a traditional full RTL model.

This approach is just one way of applying adapters. The system-critical parameters, the system size, the system complexity, and more will contribute to a verification plan that will define a system-specific approach for application of adapters.

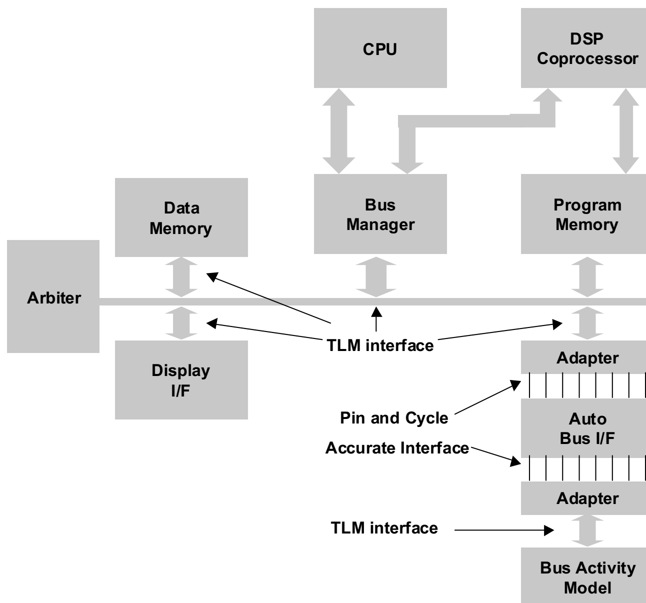

Figure 2-6. Adapter Example

2.5 Summary

A new TLM-based methodology is emerging to attack the design productivity of complex systems. The benefits of adopting this style of methodology are derived from early software development, early functional verification, and higher system quality. The productivity improvements derived from TLM-based methodology are huge and are the major motivation for adoption. Now, it is time to explore SystemC, a language that enables this new methodology.