

Transaction-level Modeling and the ConvergenSC™ Products

Pete Hardee, Director, Product Marketing, CoWare, Inc.

1 Introduction

The purpose of this white paper is to overview the support for transaction-level modeling (TLM) in the ConvergenSC™ family of SystemC-based design tools for System-on-a-Chip (SoC). The transaction level is a recognized level of abstraction for communication in SystemC. In the context of today's increasingly complex SoC's, designs consist of embedded software running on multiple processor cores, connected to memory and peripherals. As complexity grows, an increasing proportion of the software and the hardware peripherals consist of re-used Intellectual Property (IP) blocks. A prevalent design style, known as Platform-based Design (PBD) is to configure these IP blocks into platforms; to reconfigure the platform architecture quickly to create a succession of derivative designs; and to add some custom blocks aimed at differentiating each derivative design for a particular application. In order to validate the architecture of these platforms and to verify the derivative designs, the designer must focus on whether, in the new configuration of IP blocks, the communication between the blocks is optimal and correct. This is often a much larger part of the architectural validation and system verification problem, than creating and verifying the small amount of new custom functionality. In order to achieve this on complex designs with adequate simulation performance, high-level models are needed not just for the IP blocks, but also critically for the on-chip-busses. The transaction level is emerging as the level at which the bus models, and the block interfaces, are coded.

2 Introduction to Transaction-level Modeling

What is transaction-level modeling? It is an abstraction level at which first, the behavior of functional blocks in a given system can be separated from the communication, and second, the communication portion is described in terms of sending of transactions between an initiator of the communication, and the target. The word transaction implies that lower level components of communication are bundled together into something more complete. In order to understand what this means, let's first consider the current prevalent level of modeling for hardware design – the Register Transfer Level (RTL).

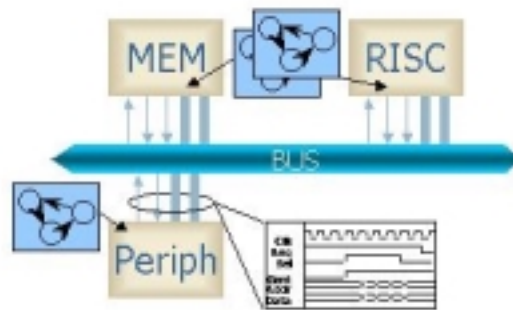


Figure 1: A Platform Representation at the Register Transfer Level

Figure 1 shows a simple RTL representation of a SoC platform. A RISC processor communicates with memory and one peripheral on a shared bus. The interfaces of the devices on the bus are modeled at the pin-accurate level – meaning that all the bus signals are individually represented. The bus itself is merely a collection of wires, including no intelligent functionality. The bus protocol, which controls all the bus signals in the manner, and with the correct timing, specified in the bus specification, is modeled individually in the interfaces of all the devices on the bus. This is represented diagrammatically by the three small Finite State Machine (FSM) symbols in figure 1. The waveform diagram shows what can be seen on

any of the bus interfaces during a bus transaction. The transaction consists of bus request, select and grant signals, and individual address and data signals all changing as prescribed by the bus protocol. As far as simulating such a model is concerned, all the signals are individually evaluated and updated at least on every cycle of the clock.

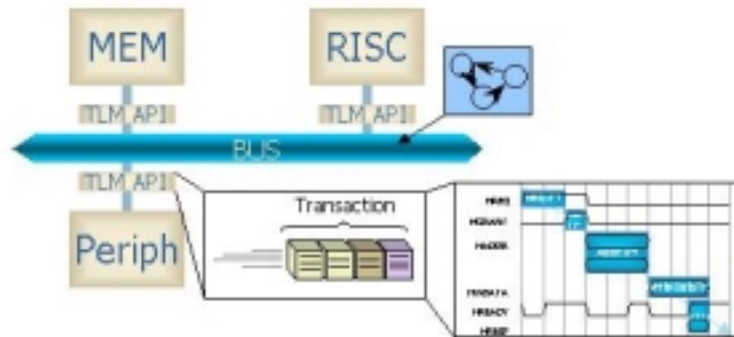


Figure 2: A Platform Representation at the Transaction Level

Now consider figure 2. With a transaction-level model, the bus is not just a collection of wires. The bus protocol is modeled in the bus model itself, not in the individual device interfaces. Each device now communicates with the bus model via a TLM API (application programming interface). The TLM API works by splitting a complete bus transaction into transfers – groups of signals that occur at the same clock edge. Instead of every signal being modeled individually, signals are grouped together and modeled as one. When simulating a system modeled this way, there are firstly, many fewer lines of code to simulate since the bus protocol is not needlessly duplicated around the system; secondly, there are many fewer bus signals modeled; and thirdly, there are many fewer events occurring on those signals. This results in considerably improved system simulation performance.

2.1 Proposed TLM Methods

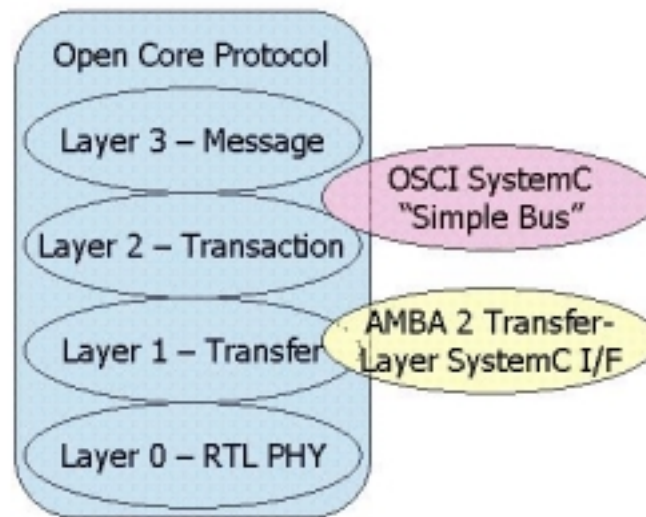


Figure 3: A Comparison of TLM Proposals

The method of TLM just described is not the only one. Figure 3 shows a diagrammatic representation of three proposals for TLM – the Open Core Protocol proposal (from the OCP-IP organization), the Open SystemC Initiative (OSCI) “Simple Bus” example (described in the Book “System Design With SystemC by Groetker et al), and the AMBA 2 Transfer-Layer SystemC Interface (A2TFI) from ARM Ltd.

The OCP-IP proposal is one example (there are many other methods that are not published) where a higher level than TLM is proposed. This is the message layer. It assumes a point-to-point connection between all components in the system can be made, and communication between the components takes place as a complete message consisting of many transactions. There is no attempt to model any given bus protocol. The OSCI simple bus example models a generic bus protocol – a scheme that is conceptually similar to many common busses today, but is simplified – at the transaction level. The A2TFI is specific to the AMBA 2.0 bus specification, and permits modeling at the transfer level. What is the transfer level versus the transaction level? As described earlier, a transaction consists of a number of transfers. It is possible to view the transactions as atomic, that is, always to group the transfers together as one, in which case one is modeling at the transaction level. Alternatively, the transfers can be viewed as atomic. What is the difference? The transfer level allows bus pipelining to be accurately modeled. This is analogous to Instruction Set Simulator (ISS) models of processors, which are commonly available as either instruction-accurate, or cycle-accurate. The major difference between these models is how accurately the processor pipeline is modeled, which is intimately tied to the relation between instruction cycles and clock cycles. Similarly, a system that leverages bus pipelining cannot be cycle-count accurate unless it is modeled at the transfer level.

We can now state that there are three somewhat independent drivers of accuracy levels for communication:

- How the structure of the message between two communicating blocks is split up, if at all (either message, transaction or transfer)
- Whether a bus protocol is modeled (either not at all, arbitration only, generic bus protocol, or bus-specific)
- The level of timing accuracy modeled in the structure and protocol (either completely untimed, correct order of events, or related to a clock)

2.2 TLM in the ConvergenSC Products

CoWare’s TLM method allows the message to be split to the transfer-level, and allows complete bus protocol specifications to be modeled, and allows cycle timing to be annotated also. If all of these are done, from the perspective of the bus communication (not necessarily from the perspective of the overall system as we shall explore later), the bus model is fully cycle-accurate. The TLM API need not be used at the transfer-level if the designer does not need to model bus pipelining. Transactions can be treated as atomic. CoWare’s Transactional Bus Simulators (TBS) are full representations of a given bus specification, modeled at the transfer level, but allowing transaction-level access too.

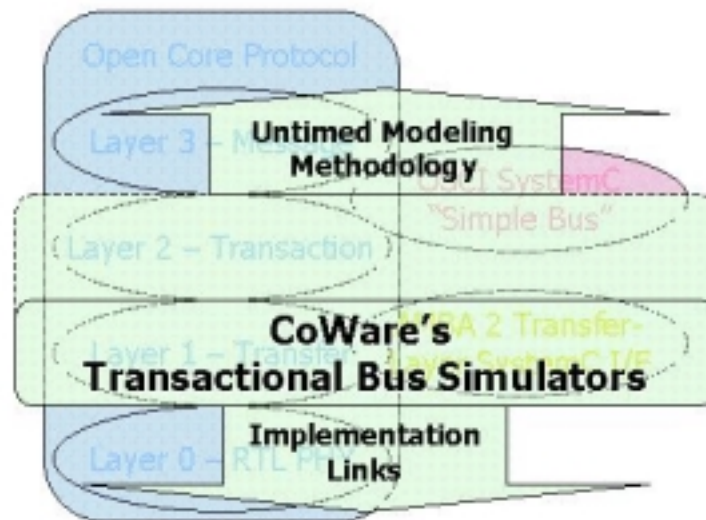


Figure 4: TLM in ConvergenSC in Relation to Other TLM Proposals

Furthermore, it is very easy to model generic bus protocols, or to use the available abstraction levels and capabilities of SystemC 2.0 to model at the message level. This is illustrated in figure 4.

Additionally, the TBS includes proprietary simulation technology that provides a cycle-accurate model and greatly increased levels of performance. Figure 5 shows this in relation to a performance/accuracy trade-off curve that is typical of current-generation simulation technology.

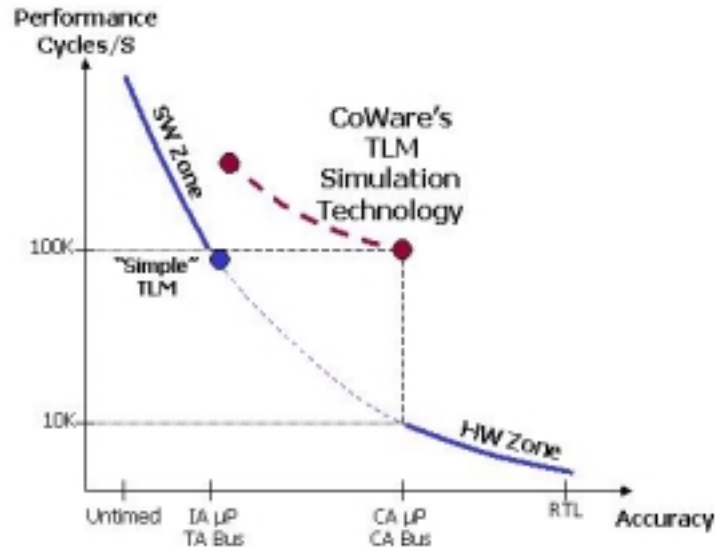


Figure 5: Performance and Accuracy of the TLM Simulation Technology in ConvergenSC

The performance axis gives a measure of overall system performance in simulation cycles per second. Traditionally, cycle accuracy has been used only for hardware implementation and verification. If the complete system, including processors, is modeled at RTL, maybe tens (or hundreds at best) of cycles per second are possible. If ISS models are used for the processors, as is the case in RTL-based hardware-software co-verification tools, performance can be raised to thousands, at most 10,000, cycles per second. In CoWare's experience, embedded software developers and system integrators demand a performance level of at least 100,000 cycles per second, for development and testing of all but a tiny proportion of the most hardware-dependent software. As figure 5 shows, TLM as represented by the simple bus model achieves the performance level, but does not model the specific bus protocol and does not model even the generic protocol cycle-accurately. CoWare's TLM technology achieves the performance level while remaining fully cycle-accurate. Furthermore, if cycle accuracy is not needed, even higher performance can be achieved.

3 SoC Design Tasks and TLM

3.1 Components of the Overall System Model

As previously mentioned, CoWare's TLM method allows a range of accuracies including fast cycle-accurate bus models. The bus is a very important component of the overall system accuracy, but not the only one. Figure 6 shows the other drivers for overall system accuracy, and infers the system design tasks for which different combinations of abstraction levels might be appropriate.

The abstraction levels of the processor models, busses and peripherals control overall system accuracy. Taking the processor model first, much application software development can be done without any representation of the target embedded processor at all. Application software developers can write code and test directly on their own PC for example. This is known as development on a host. Some more hardware-

dependent embedded software may need an instruction-accurate ISS, but does not need any accurate representation of the bus or peripheral hardware. Embedded software developers may use a processor development card in this case too. As soon as the software architecture design needs to be done, typically an RTOS is introduced and the software developer wants to model the communication between software tasks and high-level functional models of the hardware. Message level or high-level TLM (at transaction-level, probably without the full bus protocol being modeled, along with untimed functional code for the hardware peripherals, may be sufficient at this stage. For detailed development of timing critical device drivers and hardware-dependent software, collectively often known as the Hardware Abstraction Layer (HAL), a more accurate bus representation is necessary (such as provided by a TBS), and cycle timing may need to be introduced to the peripheral block behaviors. The Timed Functional abstraction level in SystemC is very convenient for modeling cycle timing either as lumped timing (cycle-accurate at the boundaries or cycle-count accurate) or fully describing the cycle-by-cycle behavior. In order to model the system cycle timing correctly, the processor model should be as cycle-accurate as possible.

Processor	Bus	Peripheral	Design tasks
Host	None	None	SW Application Dev't
Instruction Accurate	Message Level	Untimed	SW Application Dev't SW Architecture Dev't
	High-level TLM		
Cycle Approximate Cycle Accurate	Cycle-Accurate TLM (Transfer Level)	Timed Functional	HAL & R/T SW Dev't SoC Architecture Dev't System Integ'n/Co-verif.
RTL	RTL	RTL	Implementation Final Verification

Figure 6: Abstraction Levels for SoC Design Tasks

It is also at this level, using a cycle-accurate processor, a bus model at the transfer level, and relevant peripheral block behaviors modeled at the Timed Functional level, that most of the SoC platform architecture work can take place. The timing in the peripheral block behaviors can be refined to be able to produce a full cycle-accurate platform for system integration and hardware-software co-verification. Then block behaviors can be replaced with the full RTL implementation models to continue verification. Finally, both the bus (and processor if necessary) RTL implementation models can be used for a final verification.

3.2 Recommended Methodology for SoC Design and Verification

Cycle accuracy is necessary for a true picture of bus contention and memory and cache accesses. CoWare’s analysis tools (see ConvergenSC Technology Backgrounder) can readily be used to see exactly how bus bottlenecks, cache misses, and other problems in the system are being caused, and the ConvergenSC tools allow the designer to readily change the architecture, for example to add another bus layer and reconfigure the bus masters and slaves that communicate on the various layers, to remove sources of contention and thus optimize the architecture. ConvergenSC’s TLM technology helps greatly to speed up verification of RTL blocks too. As shown in figure 7, the TBS can be configured with a mix of TLM API and RTL ports. This allows the designer to successfully “divide and conquer” the task of verifying the RTL implementations, by inserting them into the transaction-level system model block-by-block. Finally, the TBS can be substituted for the RTL bus implementation for a verification of the complete RTL implementation. Simulating with as much of the system as possible at the transaction level, for as long as possible can cut a great amount of verification time.

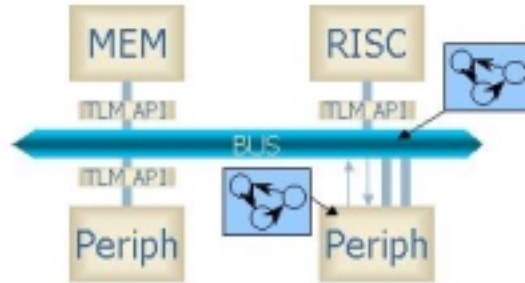


Figure 7: Mixed TLM-RTL Design

4 Transactional Bus Simulators

The AMBA TBS, shown in figure 8, is provided as a fully supported product in the ConvergenSC family. The AMBA bus specification is one of the most widely used, and having an off-the-shelf representation at transaction level, fully compliant with the AMBA specification, that can be easily configured for a given bus architecture, can save months of design time.

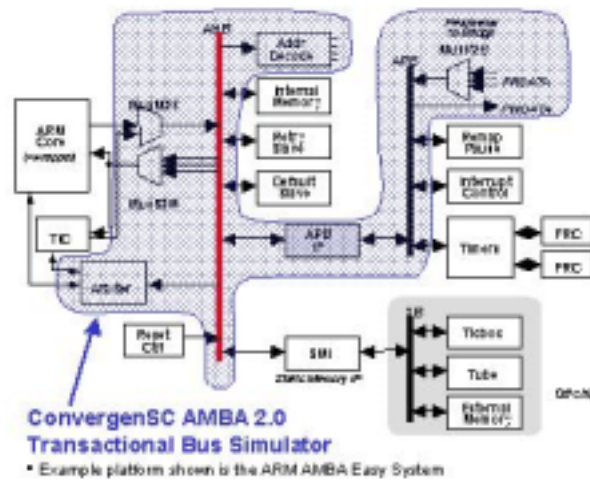


Figure 8: AMBA Transactional Bus Simulator

4.1 Custom Busses

The CoWare TLM methodology is also available for use with proprietary busses, and CoWare is working with a number of customers to automate system-level design around their in-house bus specifications. Please contact CoWare if you are interested in a TBS for your custom bus.

4.2 TLM API

CoWare’s TLM API can be made available to customers and partners who wish to code IP models to be used in transaction-level SoC platforms. Please contact CoWare.

5 Conclusions

This paper has provided an overview of TLM, both CoWare’s method in detail, and in relation to other proposed methods. The advantages of CoWare’s TBS, in terms of performance and accuracy, have been outlined. A refinement methodology has been suggested for various design and verification tasks and the abstraction levels necessary to perform them. The interested reader is invited to contact CoWare for more details.