Result-Oriented Modeling—A Novel Technique for Fast and Accurate TLM

Gunar Schirner, *Student Member, IEEE*, and Rainer Dömer, *Member, IEEE*

***Abstract*—Efficient communication modeling is a critical task in system-on-chip design and exploration. In particular, fast and accurate communication is needed to predict the performance of a system. Recently, transaction level modeling is used to speed up communication simulation at the cost of accuracy. This paper proposes a novel modeling technique, called result-oriented modeling (ROM), which removes the inaccuracy drawback of transaction level models (TLMs) in many cases. Using ROM, simulation models yield nearly the same speed as their traditional TLM counterparts, yet are still 100% accurate in timing. ROM utilizes the fact that internal states in the communication channel are not observable by the caller. Hence, ROM omits the internal states entirely and optimistically predicts the end result. Retroactively, the outcome of the prediction is checked, and if necessary, corrective measures are taken to maintain the accuracy of the model. We have applied the ROM concept to two examples: the industry standard AMBA AHB and the controller area network. To validate the proposed ROM approach, we have analyzed the models in detail for performance and accuracy. Our experimental results show the clear advantages of the ROM concept. For both bus systems, ROM achieves 100% accuracy and highest speeds. In essence, ROM eliminates the TLM tradeoff for a wide range of platforms. It frees the system designer from having multiple models for different purposes and extends the TLM idea to applications that require timing accurate simulation, such as real-time communication.**

***Index Terms*—Communication modeling, performance prediction/estimation, system level design, transaction level modeling.**

I. Introduction

SYSTEM-ON-CHIP (SoC) design faces a gap between the production capabilities and time-to-market pressures. The design space to be explored during SoC design grows with production improvements, while at the same time, shorter product life cycles force an aggressive reduction of the time-to-market. Addressing this gap has been the aim of recent research work. As one main approach, abstract models have been introduced to tackle the design complexity. For one, abstract models exhibit tremendous gains in simulation speed, allowing fast validation and extensive design space exploration.

For communication in particular, transaction level modeling has been proposed [1]. Transaction level modeling abstracts the communication in a system to whole transactions, abstracting away low-level details about pins, wires, and waveforms [2].[1] It uses timing annotations in simulating the time progress. This results in models that execute dramatically faster than bit-accurate models. This benefit, however, usually comes at the price of low accuracy. In previous work [3], [4], we have measured up to 47% average error in transaction duration for different standard bus protocols.

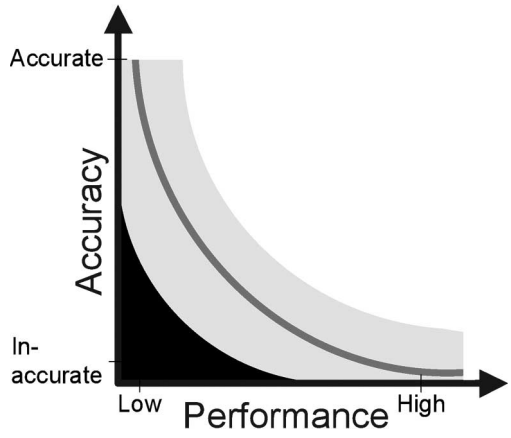

Fig. 1. TLM tradeoff.

A. TLM Tradeoff

In general, transaction level models (TLMs)[2] pose a tradeoff between an improvement in simulation speed and a loss in accuracy, as illustrated in Fig. 1. The tradeoff essentially allows models at different degrees of accuracy and speed. However, having both high speed and high accuracy at the same time is typically not possible. High simulation speed is traded in for low accuracy, and a high degree of accuracy comes at the price of low speed. For previously analyzed bus systems [3], [4] accurate models achieved less than 0.2 MB/s simulation bandwidth, while the TLMs exhibited up to 100 MB/s, however, at up to 47% error. Models with this tradeoff fall into the gray area of the diagram. Models in the dark area are obviously existent, but practically unusable, whereas models in the white area are highly desirable but typically not achievable.

Manuscript received September 29, 2006; revised December 14, 2006. This paper was recommended by Associate Editor S. A. Edwards.

The authors are with the Center for Embedded Computer Systems at the University of California, Irvine, CA 92617 USA (e-mail: hschirne@uci.edu; doemer@uci.edu).

Digital Object Identifier 10.1109/TCAD.2007.895757

[1]General modeling of SoCs consists of two parts, computation and communication. In this paper, we focus only on modeling of the communication.

[2]Note that TLM is not clearly defined in the literature. In this paper, we will use TLM as the name of the model at the granularity of an entire user transaction. A user transaction is an arbitrary sized block of data, potentially spanning multiple bus transactions.

B. Scope of This Work

In this paper, we introduce a novel modeling technique for TLM, called result-oriented modeling (ROM), which eliminates this TLM tradeoff for a wide range of platforms. In this paper, we consider designs for which the response time of a node is statically known or computable. Examples of such platforms include cell phones, multimedia SoCs, and automotive control systems [5]. Platforms for which a node's response time within a bus transaction cannot be predicted (e.g., random slave controlled wait cycles) are outside the scope of this paper.

ROM delivers simulation speeds similar to a TLM. At the same time, ROM retains 100% accuracy in timing for the considered range of platforms.[3] As a result, the system designer is relieved from the traditional speed/accuracy tradeoff and can focus on the design space exploration instead of model selection. Moreover, having a 100% accurate bus model at such a high simulation performance opens new applications of abstract communication modeling. For example, it allows for the first time that a real-time system simulation can profit from the speed advantage of abstract communication modeling.

We will apply the ROM concept to the modeling of two buses from opposite ends of the spectrum with different characteristics. We have chosen the advanced high-performance bus (AHB) of advanced microprocessor bus architecture (AMBA) [6], which is a parallel on-chip bus system with a centralized arbitration scheme, and the controller area network (CAN) [7], which is an off-chip serial bus with distributed arbitration. The analysis of these buses is based on our previous work [8], [9], where we have focused on each bus individually. In this paper, we will combine the results and provide additional details. We will also describe an algorithm for the optimistic prediction scheme used in ROM. Most importantly, we will generalize from our experience with ROM and derive general conclusions.

C. Outline

After a brief overview of relevant related work in Section II, we describe in Section III the general concepts of our novel ROM approach. Then, in Section IV, we explain in detail the application of ROM to communication modeling based on the two examples: AMBA AHB and CAN. We validate our approach and show the experimental results in Section VI. We then generalize the results in Section VII and conclude this paper in Section VIII.

II. Related Work

System level modeling has become an important research area that aims to improve the SoC design process. Languages for capturing SoC models have been developed, e.g., SystemC [1] and SpecC [10]. Capturing and designing communication architectures using TLM [1] have received much attention.

Sgroi *et al.* [11] address the SoC communication with a network-on-chip approach. Here, communication is partitioned into layers following the OSI structure. Software reuse is promoted with an increase of abstraction from the underlying communication. While this paper guides on the organization of communication, it does not directly address the TLM.

Siegmund and Müller [12] describe with SystemC$^{\mathrm{SV}}$ an extension to SystemC and propose SoC modeling at three different levels of abstraction: physical description at register transfer level (RTL), a more abstract model for individual messages, and a most abstract model utilizing transactions. The paper focuses on the interface description allowing a multilevel simulation. However, it does not address abstract modeling of multimaster buses. Brem and Müller [13] describe how the CAN bus is modeled using the aforementioned extension SystemC$^{\mathrm{SV}}$. The work also shows the three abstraction levels but does not give any experimental results on performance or accuracy.

In [14], Caldari *et al.* describe the results of capturing the AMBA rev. 2.0 bus standard in SystemC. The bus system has been modeled at two levels of abstraction: first, a bus-functional model at RTL, and second, a model at transaction level simulating individual bus transactions. The described state machine-based TLM reaches a speedup of 100 over the RTL model. Our approach proposed in this paper, however, reaches a higher speedup [three orders of magnitude over the bus functional model (BFM) for the AMBA AHB] by avoiding explicit internal states.

Coppola *et al.* [15] also propose abstract communication modeling. They present the IPSIM framework and show its efficient simulation. While the paper delivers a general overview of the SoC refinement and introduces their intramodule interface, it does not supply details of the bus modeling itself as we focus on in this paper.

Gerstlauer *et al.* describe in [16] a layered approach and propose models that implement an increasing number of International Organization for Standardization (ISO) Open System Interconnection (OSI) layers [17]. They present how to arrange communication and the granularity levels of simulation. However, they do not provide insight on the bus specific modeling.

Abstract communication is also used in Ptolemy as presented in [18] and [19], with an extension of dynamic switching between abstraction levels. A common point is the loss in accuracy with abstraction, which this paper eliminates.

Real-time communication has been analyzed in previous work. Tindell *et al.* [21], for example, analyze different CAN controllers. System level modeling of real-time communication is not explored as much. Van der Putten *et al.* [22] describe abstract real-time communication for a class of LAN protocols by modeling LAN characteristics. In contrast, we present in this paper an accurate CAN model that produces such characteristics.

Pasricha *et al.* [23] describe an approach using transaction-based abstraction. The paper introduces the concept of a model that is cycle count accurate at transaction boundaries (CCATB). This also takes advantage of the limited observability of a transaction to increase simulation performance. However, only a very limited speedup of 55% over the BFM is achieved. Their approach models individual bus transactions and uses an active thread for the bus simulation. Our ROM approach is conceptually different. We raise the abstraction to user transactions (potentially spanning multiple bus transactions) and

[3]ROM cannot guarantee timing accurate results for platforms outside of our scope, in which the node response time cannot be calculated.

avoid a dedicated thread. Consequently, ROM achieves a higher speedup of up to four orders of magnitude. In other words, while Pasricha *et al.* use an extra thread, in our approach, master and slave communicate directly through a shared channel without the need of a separate thread.

Timed abstract simulation has also been incorporated into commercial products. For example, the discrete event simulation engine in the virtual component co-design (VCC) environment [24] supports several delay models (e.g., explicitly distributed by the designer or by an automatic back annotation approach). VCC models preemption for software tasks and bus accesses by the use of suspend() and resume() messages to the simulation task, which are taken into account when a task executes a delay() function. With that, VCC uses explicit test points [i.e., the delay() call] to account for preemptions as a traditional TLM. While LaRue *et al.* [24] mostly focus on the simulation framework, this paper introduces a modeling technique (that actually can be implemented using the VCC framework). Our modeling technique ROM minimizes the interactions with the simulation engine to handle preemptive bus modeling.[4]

Real-time analysis, e.g., Audsley *et al.* [25], often captures a task's execution time as the task's computation delay (could be referred to as base time) and the sum of all external influences, such as preemptions, shared resource accesses, or communication. We use a similar terminology to describe the ROM approach. However, while the real-time analysis relies on an off-line static calculation, our approach computes the effects of any disturbances dynamically at runtime.

Ghenassia describes in [26] a TLM from an industry perspective, stating what is current and practical for industry applications. The paper also supports the general tradeoff between abstraction and accuracy.

In previous work [3], [4], we have analyzed TLM in detail, showing the tradeoff in traditional TLM for two different bus systems. In this paper, we propose a ROM that eliminates the TLM tradeoff for a wide range of systems delivering both speed and accuracy at the same time.

III. Result-Oriented Modeling (ROM)

ROM is a general concept for abstract and yet accurate modeling of a process. As such, ROM is similar to the "black box" concept.

A. Black Box Concept

The underlying assumption of ROM is the limited observability of internal state changes of the modeled process. As in a "black box" approach, it is not necessary to propagate intermediate results of the process to the user. The aggressive goal of ROM is to produce only the end result of the process, not any intermediate states.

Hiding of intermediate states gives ROM the opportunity for optimization. Often, intermediate states can be entirely eliminated. Instead, ROM can utilize an optimistic approach that predicts the outcome (e.g., termination time and final state) of the process already at the time the process is started. We characterize the prediction as optimistic, as it estimates the earliest possible termination time.

[4]Note that our approach does not require any change to existing discrete even simulation engines, such as SystemC [1] and SpecC [10]. ROM only uses the standard wait-for-time interface.

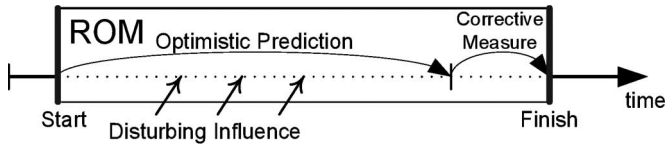

Fig. 2. Generic ROM concept.

B. Corrective Measures

Throughout the runtime of the process, a disturbing influence may change the system state, so that the initially predicted results are no longer accurate. Therefore, ROM checks at the end of the predicted time whether such a disturbing influence has occurred. If so, ROM retroactively adjusts to the new conditions and takes corrective measures. In other words, a mistake of an overly optimistic initial prediction is fixed at the end.

Optimistic prediction of the end result reduces the amount of computation and, thus, increases the execution performance, if internal states can be skipped and the cost for any corrective measures is low. This approach is in contrast to the traditional abstract modeling approach of reaching the end result through a set of incremental state changes. The traditional approach takes the disturbing influence incrementally into account and adjusts the intermediate states accordingly. ROM, on the other hand, records any disturbing influence over the predicted running time and makes any necessary adjustment at the end.

Generally speaking, the ROM approach can be characterized, as shown in Fig. 2, by the following items.

1) The process user does not need to observe internal states.
2) ROM does not model internal state changes. Instead, it optimistically predicts the end result using available system information at the beginning.
3) During the predicted runtime of the process, a disturbing influence may change the system state.
4) At the end, ROM checks if the optimistic assumptions still hold true, and takes corrective measures otherwise.

Repeating the "black box" comparison, ROM is a "black box" approach that additionally takes into account the interaction with the environment (as disturbing influence) and takes corrective measures in case the interaction is not as predicted.

C. Example

Fig. 3 illustrates the ROM approach using an example of predicting the arrival time of an airplane. The real process, Fig. 3(a), exhibits continuous changes to the ground speed dependent on the disturbing influence wind. The traditional abstract modeling approach transaction level modeling, Fig. 3(b), approximates the result by incrementally calculating the ground speed in dependence of the wind in (coarse-grain) discrete time steps. The ROM approach shown in Fig. 3(c), on the other hand, does not model the intermediate speed. Instead, it makes one

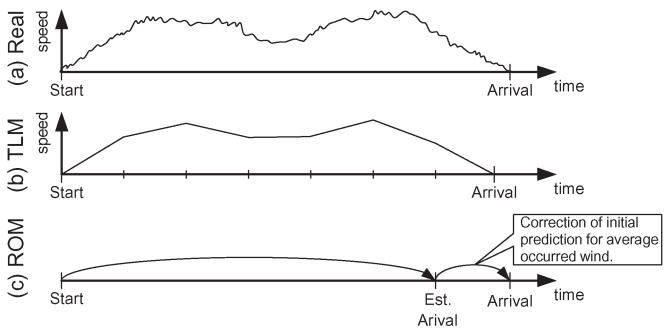

Fig. 3. ROM predicting an airplane arrival time.

initial optimistic prediction about the arrival time, and finally, it corrects its prediction retroactively for the average wind condition.

D. Analogy

As an analogy, our ROM approach can also be compared in a broader sense to optimistic timed cosimulation [27], which is a synchronization technique between two concurrent discrete event simulators. It allows each simulator to "run ahead" using internal events, before synchronizing with the other simulator. This minimizes the synchronization effort in contrast to a lock-step synchronization approach. However, it has to "roll back" in case of early external events, which our ROM does not require.

IV. Communication Modeling Using ROM

We will now describe how the general ROM concept can be applied to modeling of communication systems. We will use two different bus systems. The first is an example for an on-chip multiplexed bus system with a centralized arbitration scheme, the AMBA. Second, we model an off-chip serial bus with decentralized arbitration, the CAN.

After introducing each bus system, we will outline the traditional TLM style, and then, in contrast, apply the ROM concept.

A. AMBA AHB—Traditional Modeling

The AMBA defined by Advanced RISC Machines Ltd. (ARM) [6] is a widely used and industry-accepted standard for an on-chip bus system. We focus on the AHB, which is a system bus designed for connecting high-speed components, including ARM processors.

The AHB is a multimaster bus that operates on a single clock edge. High performance is achieved by a pipelined operation that includes address and data phases, and by the usage of burst transfers. Split and retry transfers allow the slave to free the bus if the requested data are temporary unavailable. The AHB also employs a multiplexed interconnection scheme to avoid tristate drivers.

1) Layer-Based Modeling: Following the ISO OSI reference model [17], we can model the AHB using a layered architecture [3]. The AHB specification then falls into the second layer, the data link layer. For modeling, we consider the

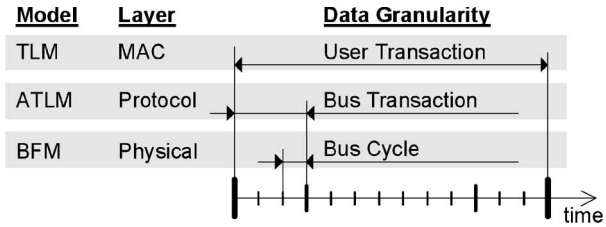

Fig. 4. Layer-based bus modeling.

media access control (MAC) and the protocol sublayer, as well as the physical layer.

Important for this discussion is the granularity of data handling in each of the layers. The media access layer provides a transmission service for a contiguous block of bytes, called a user transaction. This layer divides the arbitrarily sized user transaction into smaller bus transactions observing the bus addressing rules, and transfers these byte blocks using the protocol layer. The protocol layer transfers data as bus transactions, which are bus primitives (e.g., bytes, words, or four word bursts). It uses the services of the physical layer which provides a bus cycle access to sample and drive individual bus wires.

Fig. 4 shows the data granularity at each layer with respect to time. A user transaction is successively split into smaller units: bus transactions and bus cycles.

Following this layering, we define three models which we will refer to as TLM, ATLM, and BFM.

1) TLM. The TLM is the most abstract model, implementing only the media access layer. Data are handled at the user transaction granularity and are transferred regardless of their size in one chunk using a single memcpy. Timing is simulated as a single wait-for-time statement covering the entire user transaction. Arbitration is abstracted to a semaphore used once per user transaction.
2) Arbitrated TLM (ATLM). The ATLM models a bus access with AHB bus primitives at the protocol level. It uses the MAC layer implementation of the BFM to split user transactions into bus transactions. The ATLM accurately models priority-based arbitration, however, only once for each bus transaction. This model is not pin-accurate and not in all cases cycle-accurate.
3) BFM. The BFM is a cycle- and pin-accurate bus model. It implements all layers down to the physical layer and covers all timing and functional properties of the bus definition. The BFM handles arbitration per bus transaction and verifies the bus grant on each cycle of a burst. Additional active components, such as multiplexers, an arbiter, and an address generator are needed to accurately model the bus architecture.

2) Limitations of Layer-Based Models: To illustrate the limitations of the layer-based approach, let us consider an unlocked burst transfer. In a burst transfer, multiple data words are transferred over the bus as one block of data. An unlocked burst transfer may be preempted by a higher priority master. Hence, the active master has to check arbitration for every bus cycle (beat). In case of a preemption, the preempted master has to arbitrate again for the bus and, subsequently, resume the preempted transfer.

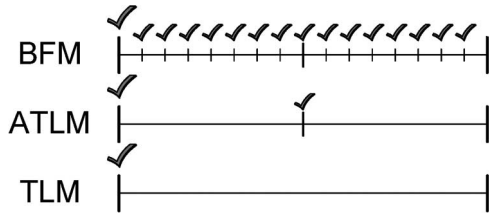

Fig. 5. Arbitration checkpoints when transferring two eight-beat bursts.

TLM in Fig. 5 shows the arbitration checkpoints as check marks for the models. A user transaction of 16 words is transferred in two eight-beat bursts. The BFM performs full arbitration at the beginning of each bus transaction and also verifies the arbitration at each cycle. The ATLM checks arbitration only at the bus transaction boundaries. The TLM performs the least amount of checking. It arbitrates only at the beginning of a user transaction.

Intuitively, we expect that the number of arbitration checks strongly correlates with the performance of the model. In order to reach an arbitration checkpoint, the model executes a wait-for-time statement (increasing the simulated time), which may result in a costly context switch in the simulator. Thus, implementing all the required arbitration checks is the slowest, but delivers accurate time modeling. The other extreme, the TLM, implements the fewest arbitration checks yielding the highest performance, but results in the worst accuracy.

3) Static Versus Dynamic Delay Model: One aspect that is common to all our traditional layer-based models is the static choice of the delay model. During the implementation, the model designer selects the granularity (or resolution) at which data and arbitration are handled. This determines the accuracy and speed achievable by the model. Our choices shown above range from modeling the individual bus cycles in the cycle-accurate BFM to the TLM that operates at a user transaction granularity. Each model executes a fixed number of wait-for-time statements for a particular transaction.

ROM, on the other hand, as we will describe in the following sections, models the timing with a dynamic delay model. Instead of distributing individual wait-for-time statements to different phases of the transaction (e.g., arbitration, address and data phase), it dynamically calculates the total transaction time and then performs a single wait-for-time statement to advance the simulation time.

The dynamic delay model becomes apparent for a preemption of the transfer, as shown in Fig. 6. Despite the fact that the model initially represented a transaction using a single wait-for-time statement, it can accommodate preemption on a cycle-by-cycle basis. For that, it records any disturbing influence during the delay time, recalculates later the transaction time using the updated system knowledge, and executes an additional wait-for-time statement. The process repeats, in case another preemption occurs in the updated time period, until no further preemptions are detected. As a consequence, the number of wait-for-time statements may vary when reexecuting the same transaction. In the optimal case without preemption, ROM will execute a single wait-for-time statement, same as the TLM.

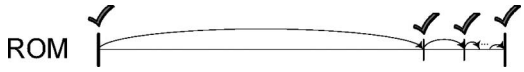

Fig. 6. Arbitration check points in ROM.

B. AMBA AHB—ROM

We will now apply the ROM approach to the AMBA AHB [8]. Unlike the traditional models, ROM does not exhibit the correlation between granularity and abstraction. It avoids unnecessary arbitration checkpoints. As a result, it reaches the BFM accuracy of 100% and near TLM performance.

1) Assumptions as With TLM: As discussed earlier, ROM is based on hiding of communication internals from the user. It avoids using signals and individual wires and implements data transfers by the use of a single memcpy operation. As such, ROM uses the same principles as TLM.

In ROM, the application is only aware of the timing at the boundaries of a user transaction. All activities of the bus model within the user transaction are hidden from the communicating parties. Those are not aware that the transaction is split into multiple bus transactions and cycles, neither that there is arbitration involved.

Only the timing at the boundaries of the user transaction is important for the application. For accurate timing, the start and the end times of each transaction must match the times reported by a BFM.

Between the start and end times, ROM can freely rearrange and/or omit internal events and state changes in order to eliminate costly context switches in the simulator.

As in TLM, the main idea in speeding up the simulation is to replace the sequence of wait operations and arbitration checks with one single wait-for-time statement. Reducing the number of wait operations is the biggest contributor to the increased execution performance. This avoids running the scheduling algorithm in the simulation engine and, thus, also reduces the number of possible context switches.

2) Optimistic Modeling: The ROM implements an optimistic approach. When a master requests a user transaction, the earliest finish time for this transfer is calculated, and the master waits until that time. The time prediction takes the current state of the bus into account. In case a higher priority transaction is already active, the wait time is increased for its duration. After the calculated time has passed, the master verifies whether the predicted time is still accurate. If so, the transaction is complete. Note that, in this best case scenario, ROM uses only a single wait statement (same as the TLM).

With a disturbing influence of a higher priority master accessing the bus during a transaction, the predicted time will be too short. Then, ROM recalculates the predicted time and waits for it. This process is repeated until the prediction is verified to be correct.

Note that an optimistic (short) prediction is highly desired to allow for corrections. With a pessimistic (too long) prediction, a correction would need to go back in time. This, in turn, would require a simulation checkpoint and rollback mechanism, which introduces an undesired performance penalty.

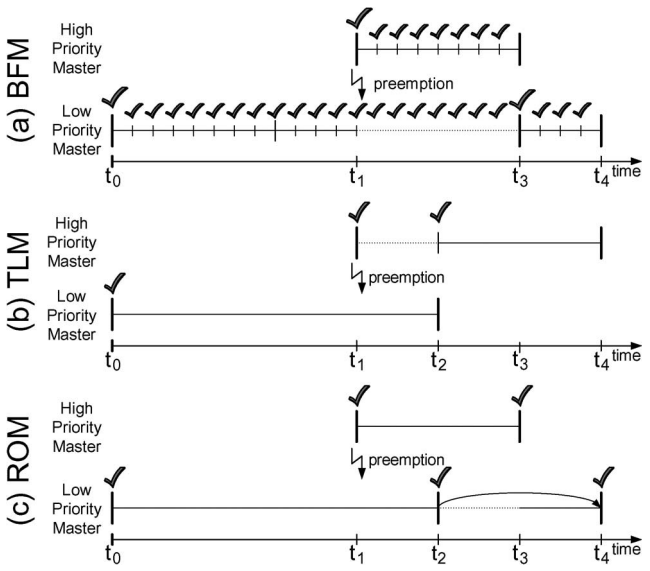

Fig. 7. Preemption in BFM, TLM, and ROM.

TABLE I
PREEMPTION COMPLEXITY COMPARISON

	BFM	TLM	ROM
Arbitration Checks (in percent)	32 100%	3 10.7 %	5 15.6%

3) Preemption: To compare the ROM against the layered models, we will analyze the case of a bus preemption in more detail, as shown in Fig. 7.

In Fig. 7(a) (BFM), a burst transaction starting at t_0 is preempted at t_1. The higher priority transfer completes at t_3. At that time, the preempted transfer resumes and terminates finally at t_4. Both masters perform arbitration checks for every bus cycle, a total of 32 in this example.

In Fig. 7(b) (TLM),[5] the low priority transaction is not properly preempted and still ends at t_2 (not at t_4). Instead, the high priority transaction is delayed until t_2 and ends at t_4 (not at t_3). Clearly, the abstract TLM is highly inaccurate in the finish times of both transfers but executes fast. Only three arbitration checks are performed.

In Fig. 7(c) (ROM), the inaccuracies of the TLM are corrected by three additional arbitration checks. The low priority transfer is initially predicted to finish at t_2. Then, it detects that it has been preempted at t_1 and recalculates its finish time for $t_3 - t_1$ time units later at t_4. The high priority master wakes up at t_3 and terminates its transaction since it was not preempted. At t_4, the low priority master wakes up, verifies that no other preemption has occurred, and thus, completes its transfer.

Note that the final two arbitration checks performed by the ROM are inexpensive because no further waiting is necessary and no context switch can occur.

Table I compares the arbitration checks performed by the models. The number of checks in ROM is close to the TLM case and an order of magnitude lower than in the BFM.

4) Pipelining: So far, we have simplified our description of the ROM for AMBA AHB. In the real implementation, the pipelining effects, as defined by the standard [6], are taken into account. Therefore, bus transactions of multiple masters may execute at the same time in different stages of the pipeline. Furthermore, if two masters transfer a set of nonsequential bus transactions, even an interleaved parallel transfer is possible.

The concept described above is still valid. However, the implementation has to deal with more details. For example, in an update of a preempted transaction, it is not sufficient to simply include the duration of the preemption. Instead, ROM considers a global bus scheduling each time a transaction is requested. It calculates for each known transaction the stage(s) it occupies for every bus cycle. Here, ROM uses static knowledge about the slaves to determine the slaves' response pattern (e.g., delay cycles). Based on this global schedule, the ROM determines the transaction duration (or the update) of a user transaction.

C. CAN—Traditional Modeling

The second bus example, CAN, is an off-chip serial bus with decentralized arbitration [9], which is very different from the earlier described AMBA AHB. In the following sections, we will introduce the CAN bus, touch briefly on the layer-based models, and describe the ROM for the CAN wherever it is different from the AMBA AHB implementation. The main focus will then rest on the analysis of the ROM benefits.

The CAN is a real-time serial communications protocol, introduced by the Robert Bosch GmbH [7], with a focus on automotive applications. Since it is often used in real-time safety critical systems, the timing accuracy of a bus model is crucial.

CAN is a serial multinode broadcast bus. Frames, with up to 8 B user data, are received by all bus nodes and distinguished by the frame identifier. Each bus node decides using local rules whether to process the frame or not. The frame identifier also defines the priority. If multiple senders attempt a transmission, the non-destructive bitwise arbitration guarantees that the highest priority frame will succeed undisturbed.

In order to ensure correctness of the received data, each CAN message includes a 15-bit cyclic redundancy check (CRC). In case of a CRC mismatch, a retransmission of the frame is triggered. The protocol also defines elaborate error detection and error confinement rules for protection against faulty bus nodes.

The serial CAN protocol operates without a centralized clock. Each node synchronizes on the bit stream of the sender. A bit stuffing rule guarantees sufficient edges for this synchronization. After transmitting 5 bits of equal polarity, a bit of opposite polarity is introduced. With the bit stuffing, the physical frame length depends on the frame content.

We have applied the same layer-based approach, as in Section IV-A, for modeling the CAN using the same three abstraction levels: the BFM, ATLM, and the TLM. The most abstract model, the TLM, is identical to the AHB implementation and simulates on a user transaction granularity. The ATLM, simulating individual CAN frames, already achieves accurate arbitration based on the CAN frame identifier. It also performs a bitwise inspection of each frame for CRC calculation and stuff bit handling. The BFM is our most detailed model implementing all features of the specification. It serially sends and receives

[5]For brevity, we will omit the similar ATLM case here.

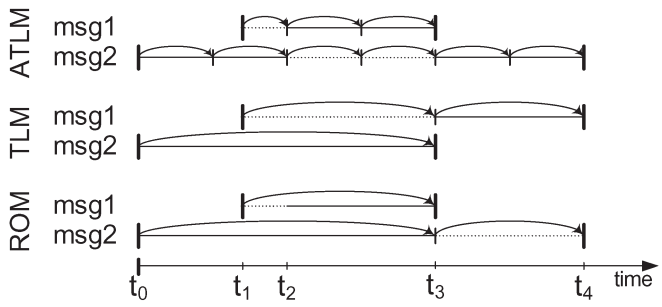

Fig. 8. Contention in ATLM, TLM, and ROM.

the data and synchronizes each node's clock to the bit stream according to the definitions in [7] and [28].

D. CAN—ROM

The basic concepts of ROM for the CAN are identical to ROM of the AMBA bus. ROM abstracts to the level of a user transaction, combining multiple CAN frames using a single memcpy and minimizing the number of potentially costly wait-for-time statements. As described for the general case, the CAN ROM relies on the limited observability in rearranging internal events. The application is unaware of splitting the user transaction into individual CAN frames and arbitration details.

The ROM optimistically predicts the duration of the whole user transaction, taking into account the split into CAN frames and the current status of the bus with all pending CAN frames.

Note that, due to the bit stuffing rule, the number of physical bits needed for the transmission of a CAN frame also depends on the content of the message, not only the user data length. The bit stuffing rule also extends to the CRC field of a CAN frame. Therefore, ROM calculates the CRC for each CAN frame and takes into account the number of stuff bits that would be inserted. As we will see later, the additional effort for the bit inspection prevents the ROM from outperforming the TLM.

In order to guarantee accurate timing, the ROM verifies the initial optimistic prediction at the end of the predicted time. In case a higher priority message starts during a multiframe low priority message, the ROM detects the recorded disturbing influence. It then recalculates the predicted time and waits for it. This process repeats until the prediction is correct. To minimize the computation effort for an update, the ROM stores the physical length of each frame during the initial prediction and, thus, avoids the costly bit inspection during an update.

To illustrate the ROM in contrast to the layered models, we analyze an example with disturbing influence, as shown in Fig. 8. A low priority msg2 with four frames starts at t_0. During its second frame at t_1 a high priority msg1 is released, which delays the completion of msg2. In the example, we depict the wait-for-time statements to estimate the model's performance.

In **ATLM**,[6] the delay is correctly modeled. Since a CAN frame is not preemptable, modeling at the CAN frame granularity is sufficient. Msg1 waits until the start of the next frame at t_2 when it wins the arbitration. Msg2 looses the arbitration for its third frame at t_2 and retries twice until t_3, when msg1 terminates. Then, the last two frames of msg2 are transmitted until t_4. In total, the sending nodes execute nine wait-for-time statements.

[6]For brevity, we omit the BFM case here.

TABLE II
WAIT COMPLEXITY WITH DISTURBANCE

	BFM	ATLM	TLM	ROM
wait-for-time (in percent)	11,888 100%	9 0.075%	3 0.025%	3 0.025%

In TLM, the example is not accurately simulated due to the coarse granularity. The ongoing msg2 cannot be delayed and ends at t_3 (not at t_4). Only after that msg1 starts and finishes too late at t_4. Clearly, the abstract TLM is highly inaccurate in the finish times of both transfers, but executes fast with only three wait-for-time statements.

In ROM, the low priority msg2 is initially predicted to finish at t_3. At t_1, msg1 is predicted to start after the current frame of msg2 at t_2 and finish at t_3. At t_3, the node sending msg1 wakes up and terminates since no disturbing influence has occurred. At the same time, the node sending msg2 wakes up and detects the disturbing influence of msg1. It then updates its finish time for $t_3 - t_2$ time units later at t_4 and waits until then. When it wakes up again at t_4, it detects no other disturbing influence and completes its transfer. ROM is able to correctly simulate the example with only three wait-for-time statements.

Table II compares the wait-for-time statements performed by the CAN models. ROM waits as often as the TLM, four orders of magnitude less frequently than the BFM.

V. ROM ALGORITHM

To give an additional level of understanding in ROM, we present now a generic algorithm that implements ROM for basic bus-based communication. For ease of understanding, we omit extra complexities, such as pipelining, in this description. Algorithm 1 outlines how ROM implements a user transaction.

Algorithm 1 ExecuteTransaction(prio, address, length, mode)

```
1: waitTime = PredictWaitTime(List, prio, address, length, mode);
2: InsertTransaction(List, prio, address, length, mode);
3: repeat
4:    waitfortime(waitTime);
5:    waitTime = GetDelayDueToPreemptions(List);
6: until waitTime == 0
7: TransferData(address, length);
8: RemoveTransaction(List, prio);
```

Algorithm 1 shows the characteristic elements of ROM: the initial prediction of the transaction time (waitTime, line 1), waiting for the predicted time (line 4), and updating the prediction for any disturbing influence (line 5). GetDelayDueTo-Preemptions() examines past disturbing influence and returns the accumulated delay of preempting transactions. The loop

repeats the waiting until no further preemptions have occurred. A global List, sorted by priority and transaction start time, keeps track of the scheduled transactions. Each transaction is added to that List (line 2) and removed upon completion (line 8).

PredictWaitTime() calculates the initial prediction; Algorithm 2 outlines its implementation.

Algorithm 2 PredictWaitTime(List, prio, address, length, mode)

```
1:  active = GetCurrentOngoingTransaction(List);
2:  if BusIsUnused(List) then
3:    timeToStart = TimeToBusClockCycle();
4:  else if prio < Priority(active) then
5:    timeToStart = TimeToNextPreemption(active);
6:  else
7:    pred = FindLeastHigherPrioTransaction(List, prio);
8:    timeToStart = TimeToFinish(pred);
9:  end if
10: duration = TransactionDuration(address, length,
    mode);
11: return timeToStart + duration;
```

PredictWaitTime() first determines the delay until the transaction can be scheduled (timeToStart) for three cases. If the bus is unused (line 2), the requested transaction starts immediately, only aligned to the next bus clock cycle. Line 4 tests if the requested transaction is of higher priority than the currently active transaction, which GetCurrentOngoingTransaction() retrieves from the global List. In this case, the requested transaction is delayed until the next preemption point of the active transaction. In the remaining clause starting at line 7, FindLeastHigherPrioTransaction() searches the global List for the predecessor transaction with the lowest priority higher than the requested priority. The requested transaction will be scheduled after the found transaction pred finishes, as calculated by TimeToFinish().

After determining timeToStart, the minimal transfer duration is calculated by TransactionDuration(). This takes into account the user transaction length, its base address (for alignment), and mode (e.g., burst enabled). It also relies on static knowledge about response time of the selected slave. As a result of the scheduling decisions, line 11 finally returns the sum of the wait times for requested transaction, which is its earliest finish time given the current bus conditions.

VI. Experimental Results

To validate the benefits of our proposed ROM approach, we have implemented for both buses all four models using the SpecC [10] system level design language (SLDL). Please note that we implemented the ROM without any modification to the simulation engine, using only regular wait-for-time statements and events.

We analyzed all models in detail and examined three aspects: 1) the accuracy, since our main premise is to reach 100% accuracy; 2) the simulation performance to validate that ROM actually reaches TLM speeds; and 3) the number of prediction updates that are needed by ROM.

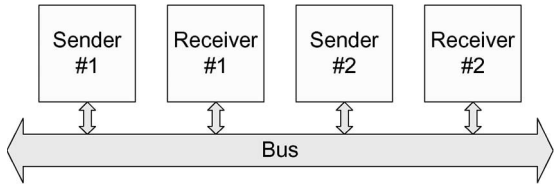

Fig. 9. Multinode setup.

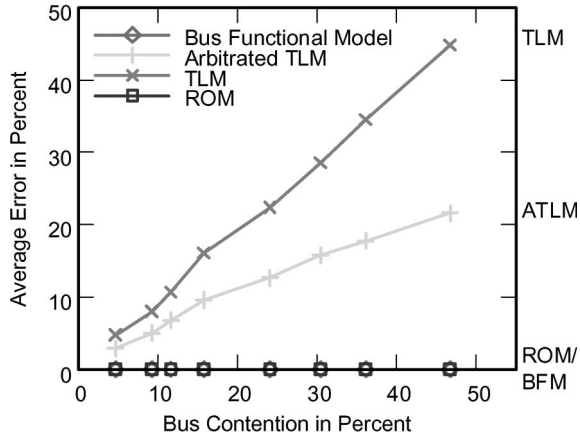

Fig. 10. Accuracy of the AMBA AHB models.

A. Accuracy

In analyzing the accuracy of our models, we use the metrics and setup described in [3]. In a multinode setup such as depicted in Fig. 9, each node sends a set of 5000 predefined user transactions that vary in user transaction id (each from its own range), size, content, and delay between two transfers. The same set of transactions is transferred by each model. We repeat each test for different amounts of bus contention. During the test, we record the duration for each individual transfer and compare that later against the cycle-accurate BFM.

1) AMBA AHB: We have measured the accuracy of the AMBA AHB models in a two master setup. Fig. 10 shows the average error in transaction duration for the high priority master over a varying degree of bus contention.[7]

As targeted, the ROM shows 0% error for all measurements, lying right on top of the x-axis (same as the BFM). In contrast, the TLM and ATLM show significant error rates, linearly increasing with growing bus contention. At 45% contention, the TLM reaches 45% error, making any system timing analysis based on TLM questionable.

2) CAN: We use a test setup with eight nodes[8] connected to the CAN bus. Four nodes act as senders and four nodes as receivers. Each sender uses an exclusive range of message identifiers (in effect, a different priority). Fig. 11 shows the average error in transaction duration for the node sending the lowest priority messages over a varying degree of bus contention.

[7] A more detailed error analysis can be found in [3].

[8] Other tests, omitted for space reasons, show similar results.

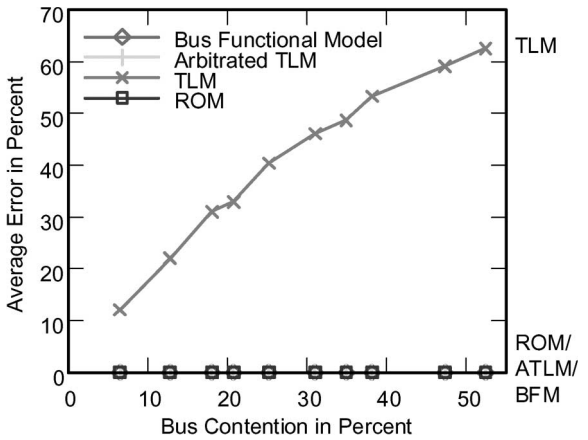

Fig. 11. Accuracy of the CAN models.

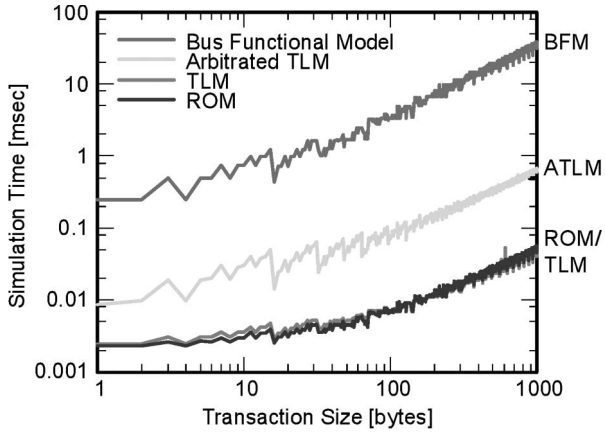

Fig. 12. Transfer time using AMBA models.

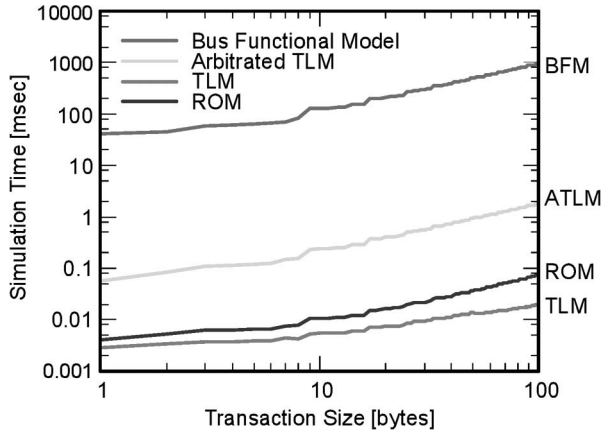

Fig. 13. Transfer time using CAN models.

For the CAN as well, ROM shows 0% error for all measurements. Its graph lies right on top of the x-axis (same as the BFM and the ATLM). Note that, in the CAN protocol, a CAN frame cannot be preempted; therefore, the ATLM is already accurate in this case, since arbitrating once per frame matches the protocol definition. In contrast, the TLM shows significant error rates, linearly increasing with growing bus contention, passing 60% error at 50% contention.[9]

In summary, for both the AMBA AHB and the CAN, we have reached our goal of 100% accuracy in timing. Next, we analyze the simulation performance of ROM in comparison to the traditional models.

B. Performance

We have measured the simulation performance also in a multinode setup. One group of nodes, with higher priority, accesses the bus in a regular interval, producing a base utilization of the bus system. At the same time, the lowest priority node issues transactions of increasing size without a delay in between. Hence, low priority transactions are preempted or interspersed by higher priority ones. We have measured the simulation time of the whole system during the lowest priority transactions (including the higher priority transactions that are transmitted in between).[10] We executed the test on a Pentium 4 PC at 2.8 GHz.

1) AMBA AHB: We use an architecture with two masters and two slaves in analyzing the AMBA AHB. The high priority master generates an equally distributed base load of 33% on the bus by sending eight-beat burst transactions. Fig. 12 shows the time to simulate the low priority master's transactions over an increasing message size, while the high priority master is running at the same time.

Fig. 12 reveals the tremendous performance benefit of ROM. Both ROM and TLM are equally fast, three orders of magnitude faster than the BFM and one order of magnitude faster than the ATLM. Both the BFM and the ATLM show a characteristic saw tooth shape, due to the nonlinear split into bus transactions, equivalent to what we observed in [3].

2) CAN: In our measurement scenario for the CAN, three high priority senders produce a constant bus load totaling 50% base utilization by transferring 16-byte messages of 16 byte size. Again, we measure the lowest priority sender. Fig. 13 shows the simulation time in transmitting a low priority transaction over an increasing size, running concurrently with the high priority senders.

All models exhibit a characteristic increase in simulation time when exceeding a size divisible by eight. Then, another CAN frame is needed, which increases simulation effort and the probability of additional higher priority messages.

ROM and TLM execute five orders of magnitude faster than the BFM. The TLM reaches this speed (0.006 ms for 16 bytes) with the coarse grain operation on user transactions. Although ROM optimizes the wait-for-time statements, it does not outperform the TLM. With the costly bit inspection necessary for accurate timing, it is only two times slower than the TLM: 0.012 ms for a transaction with 16 byte size. The ATLM is about 24 times slower than the ROM, since it incrementally simulates each CAN frame. The cycle-accurate BFM with its serial simulation executes the slowest. Transferring 16 bytes takes 155 ms. For the CAN, ROM is the fastest to accurately model the communication, 12 700× faster than the BFM.

[9] A more detailed error analysis can be found in [4].

[10] For a fair comparison, we also ensure that all models transfer the same amount of user transactions regardless of the model accuracy.

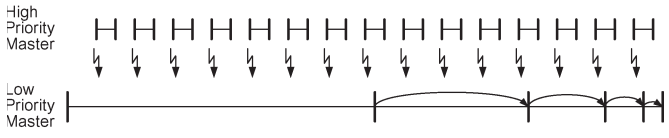

Fig. 14. Exponentially decreasing number of prediction updates.

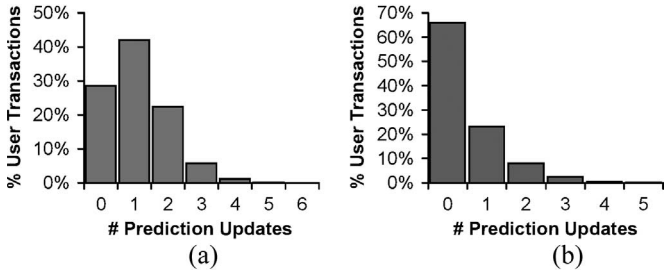

Fig. 15. Histogram of number of prediction updates. (a) AMBA AHB. (b) CAN.

C. Prediction Updates

As we have seen for both bus systems, our proposed ROM performs excellent despite the presence of frequent disturbing influences. In particular, with long bus transactions, higher priority bus activities preempt frequently the measured lower priority transaction. This excellent performance indicates that only few prediction updates are necessary, even in a high contention situation. In the next paragraphs, we will first reason about this phenomenon and then show empirical results.

Fig. 14 shows an example where a long transaction is frequently preempted. Although this transfer is preempted 15 times, only four prediction updates are needed by the ROM. A closer look shows eight preemptions during the initially predicted period, four in the next, then two, and finally only one. This exponential drop indicates that, for most transfers, only very few prediction updates are expected, even under high bus load.

In order to validate this expectation, we have measured the actual number of prediction updates when performing linear random transfers.

1) AMBA AHB: In the setup with the AMBA AHB, two masters transfer transactions of random size (1–200 bytes) with a random delay in between. By controlling the maximum delay between transfers, we adjust the system to 50% bus contention.

Fig. 15(a) shows the number of prediction updates per low priority transaction in a histogram over 100 000 transfers. Most transactions require only a single prediction update, despite the high amount of contention. As expected, the number of transactions with more than one prediction update reduces exponentially. Only 1.1% require four prediction updates. Note that 27.5% of the transactions complete without a single update. In other words, the initial prediction is actually correct for about a third of all cases. These are mainly small transactions (58% of the transactions without updates are 50 byte or smaller in size).

2) CAN: We have validated the same fact using the CAN with eight nodes (four senders, four receivers). The histogram in Fig. 15(b) shows the results for the condition in which we expect the most updates: the node with the lowest priority messages at a high bus contention of 51%.

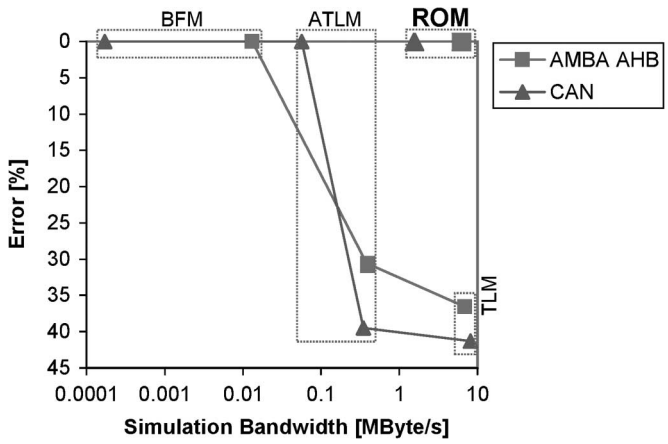

Fig. 16. ROM beats the TLM tradeoff.

The percentage of transactions requiring prediction updates reduces exponentially with the number of updates. Two third of the transactions require no prediction update at all, with the majority being single frame messages. Only 23% of the transactions require one prediction update, and only 0.5% need four updates.

In both cases, the AMBA AHB and the CAN, the probability of updates drops exponentially with the number of updates. From this fact, we conclude that the excellent performance of ROM can be more generally expected.

VII. Generalization

Combining the results of the two bus systems with complementary characteristics, we now derive more general conclusions. While, for the existing multitude of embedded bus architectures, an extrapolation based on two individual models is difficult in general, our data are a strong indication of generality. The modeled bus systems are on opposite ends of the spectrum. While the AMBA AHB is an on-chip parallel bus system with a centralized arbitration scheme, the CAN is an off-chip serial bus with distributed arbitration. We estimate that the results of the AHB modeling are a good indicator in abstracting the IBM CoreConnect Processor Local Bus [29]. Furthermore, we believe that the CAN modeling experience can be used as an indicator for other serial bus systems such as I^2C [30] and Ethernet.

A. Escaping the TLM Tradeoff

Fig. 16 shows the combined results for the AMBA AHB and the CAN in a graph with actual measurements. Note that Fig. 16 resembles the generally expected tradeoff depicted in Fig. 1 in the beginning of this paper. Again, the graph illustrates the tradeoff between accuracy and speed. The x-axis denotes the bandwidth of the lowest priority node when using 100-byte transactions in a multinode setup with two senders. The y-axis denotes the average error in transfer duration for the lowest priority node when the bus shows 40% contention.

Our experimental results show that the traditional models (TLM, ATLM, and BFM) follow the TLM tradeoff. They are either accurate (i.e., the BFMs and the ATLM for the CAN) but

slow with less than 0.4 MB/s, or they are fast (i.e., the TLMs) but exhibit an error of more than 35%.

Our ROM, on the other hand, escapes the TLM tradeoff for both bus systems. At the same time, it is fast and 100% accurate. It accurately simulates the AHB with a bandwidth of 6.6 MB/s, and the CAN with 1.6 MB/s.

B. Complexity Considerations

It should be noted that the advantages of ROM come at the price of a more complex model implementation. The BFM and TLM implementations, on one hand, incrementally advance time and can therefore use step-by-step decisions. ROM, on the other hand, implements all bus scheduling decisions explicitly at the boundaries of a user transaction. This requires the model to keep track of outstanding transactions and reevaluate decisions if they were overly optimistic, requiring a significant higher effort from the model developer. For our implementation, we experienced about double the time to model the ROM designs.

VIII. Conclusion

In this paper, we have introduced a novel modeling concept, result oriented modeling (ROM), and its application to the abstract modeling of communication in SoC design. ROM is a modeling approach similar to TLM that hides internal states and minimizes them in order to gain execution speed. Moreover, ROM is based on an optimistic paradigm. It predicts the end result at the beginning. Under a disturbing influence, corrective measures are taken at the end in order to adjust the prediction so that 100% accuracy is achieved.

We have applied the ROM concept to two complementary buses: the industry standard on-chip parallel bus system AMBA AHB with a centralized arbitration scheme and the off-chip CAN bus system with its decentralized arbitration. We have compared the proposed ROM implementations against traditional layer-based models. Our detailed analysis shows that the cost of corrective measures is low due to an exponential decreasing number of necessary prediction updates. As a result, the ROM reaches near the TLM performance.

Our experimental results demonstrate the benefits of ROM. While the traditional models suffer from a significant speed/accuracy tradeoff, ROM eliminates this tradeoff for a wide range of platforms, delivering highest speed and 100% accuracy at the same time. This allows rapid design space exploration with high fidelity at the abstract system level. ROM also opens the opportunity to utilize the TLM benefits for real-time sensitive applications, where 100% timing accuracy is required.

References

[1] T. Grötker, S. Liao, G. Martin, and S. Swan, *System Design With SystemC*. Norwell, MA: Kluwer, 2002.

[2] L. Cai and D. Gajski, "Transaction level modeling: An overview," in *Proc. Int. Conf. Hardware/Softw. Codesign and Syst. Synthesis*, Newport Beach, CA, Oct. 2003, pp. 19–24.

[3] G. Schirner and R. Dömer, "Quantitative analysis of transaction level models for the AMBA bus," in *Proc. DATE Conf.*, Munich, Germany, Mar. 2006, pp. 230–235.

[4] G. Schirner and R. Dömer, "Abstract communication modeling: A case study using the CAN automotive bus," in *From Specification to Embedded Systems Application*, A. Rettberg, M. Zanella, and F. Rammig, Eds. Manaus, Brazil: Springer-Verlag, Aug. 2005.

[5] G. Schirner, G. Sachdeva, A. Gerstlauer, and R. Dömer, "Embedded software development in an system-level design flow: Case study for an ARM processor," in *Proc. Int. Embedded Syst. Symp.*, Irvine, CA, Jun. 2007.

[6] Advanced RISC Machines Ltd. (ARM), *AMBA Specification (Rev. 2.0), ARM IHI 0011A*, 1999. [Online]. Available: http://www.arm.com/products/solutions/AMBA_Spec.html

[7] *CAN Specification*, 1991, Robert Bosch GmbH. [Online]. Available: http://www.can.bosch.com

[8] G. Schirner and R. Dömer, "Fast and accurate transaction level models using result oriented modeling," in *Proc. ICCAD*, San Jose, CA, Nov. 2006, pp. 363–368.

[9] G. Schirner and R. Dömer, "Accurate yet fast modeling of real-time communication," in *Proc. Int. Conf. Hardware/Softw. Codesign and Syst. Synthesis (CODES + ISSS)*, Seoul, Korea, Oct. 2006, pp. 70–75.

[10] D. D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, and S. Zhao, *SpecC: Specification Language and Design Methodology*. Norwell, MA: Kluwer, 2000.

[11] M. Sgroi, M. Sheets, M. Mihal, K. Keutzer, S. Malik, J. Rabaey, and A. Sangiovanni-Vincentelli, "Addressing the system-on-a-chip interconnect woes through communication based design," in *Proc. Des. Autom. Conf.*, Jun. 2001, pp. 667–672.

[12] R. Siegmund and D. Müller, "SystemC^{SV}: An extension of SystemC for mixed multi-level communication modeling and interface-based system design," in *Proc. DATE Conf.*, Munich, Germany, Mar. 2001, pp. 26–32.

[13] D. Brem and D. Müller, "Interface based system modeling of a CAN using SVE," in *Proc. EkompaSS Workshop*, Hanover, Germany, Apr. 2003.

[14] 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 *Proc. DATE Conf.*, Munich, Germany, Mar. 2003, pp. 26–31.

[15] M. Coppola, S. Curaba, M. Grammatikakis, and G. Maruccia, "IPSIM: SystemC 3.0 enhancements for communication refinement," in *Proc. DATE Conf.*, Munich, Germany, Mar. 2003, pp. 106–111.

[16] A. Gerstlauer, D. Shin, R. Doemer, and D. Gajski, "System-level communication modeling for network-on-chip synthesis," in *Proc. Asia and South Pacific Des. Autom. Conf.*, Shanghai, China, Jan. 2005, pp. 45–48.

[17] *Reference Model of Open System Interconnection (OSI)*, 1994, International Organization for Standardization (ISO). iSO/IEC 7498 Standard.

[18] M. Lajolo, C. Passerone, and L. Lavagno, "Scalable techniques for system-level co-simulation and co-estimation," *Proc. Inst. Electr. Eng.—Computers and Digital Techniques*, vol. 150, no. 4, pp. 227–238, Jul. 2003.

[19] K. Hines and G. Borriello, "Optimizing communication in embedded system co-simulation," in *Proc. Codes/CACHE*, Braunschweig, Germany, Mar. 1997, pp. 121–125.

[20] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Applications*. Norwell, MA: Kluwer, 1997.

[21] K. W. Tindell, H. Hansson, and A. J. Wellings, "Analysing real-time communications: Controller area network (CAN)," in *Proc. Real-Time Syst. Symp.*, Dec. 1994, pp. 259–263.

[22] P. van der Putten, J. Voeten, M. Geilen, and M. Stevens, "System level models for real-time communication," in *Proc. EUROMICRO*, Sep. 1999, pp. 496–501.

[23] S. Pasrich, N. Dutt, and M. Ben-Romdhane, "Fast exploration of bus-based on-chip communication architectures," in *Proc. Int. Conf. Hardware/Softw. Codesign and Syst. Synthesis (CODES+ISSS)*, Stockholm, Sweden, Sep. 2004, pp. 242–247.

[24] W. LaRue, S. Solden, and B. Bhattacharya, "Functional and performance modeling of concurrency in VCC," in *Proc. Concurrency and Hardware Des.—Advances Petri Nets*, J. Cortadella, A. Yakovlev, and G. Rozenberg, Eds., Nov. 2002, vol. 2549, pp. 191–227.

[25] N. C. Audsley, K. W. Tindell, and A. Burns, "The end of the line for static cyclic scheduling?" in *Proc. 5th Euromicro Workshop Real-Time Syst.*, Oulu, Finland, 1993, pp. 36–41.

[26] F. Ghenassia, *Transaction-Level Modeling With SystemC: TLM Concepts and Applications for Embedded Systems*. New York: Springer-Verlag, 2005.

[27] S. Yoo and K. Choi, "Synchronization overhead reduction in timed cosimulation," in *Proc. IEEE Int. High Level Design Validation and Test Workshop*, Nov. 1997, pp. 157–164.

[28] F. Hartwich and A. Bassemir, *The Configuration of the CAN Bit Timing*, 1999. [Online]. Available: http://www.can.bosch.com
[29] *128-Bit Processor Local Bus Architecture Specification*, 4th ed. Armonk, NY: IBM, 2004. SA-14-2538-04.
[30] *The I^2C-Bus Specification*, 2nd ed. Eindhoven, The Netherlands: Philips Semiconductors, Jan. 2000. document order number: 9398 393 40011.
[31] A. Gerstlauer and D. D. Gajski, "System-level abstraction semantics," in *Proc. Int. Symp. Syst. Synthesis*, Kyoto, Japan, Oct. 2002, pp. 231–236.

Gunar Schirner (S'04) received the diploma in electrical engineering and computer science from Berufsakademie Berlin, Germany, in 1998, and the M.S. degree in the same area from the University of California, Irvine, in 2005, where he is currently working toward the Ph.D. degree.

From 1998 to 2003, he was an SW Design Engineer with Alcatel, first in Germany and then in the USA. He worked on real-time embedded software for telecommunication products and on communication infrastructure for a next-generation digital loop carrier. His research interests include system-level design and modeling, embedded software modeling and synthesis.

Rainer Dömer (S'96–M'00) received the Ph.D. degree in information and computer science from the University of Dortmund, Dortmund, Germany, in 2000.

He is currently an Assistant Professor in electrical engineering and computer science at the University of California, Irvine (UCI). His research interests include system-level design and methodologies, embedded computer systems, specification and modeling languages, system-on-chip design, and embedded software.

Dr. Dömer is a member of the Center for Embedded Computer Systems (CECS) at UCI.