Accurate yet Fast Modeling of Real-Time Communication

Gunar Schirner and Rainer Dömer
Center for Embedded Computer Systems
University of California, Irvine, USA
hschirne@uci.edu, doemer@uci.edu

ABSTRACT

Accurate modeling of communication is a necessary part of system level design for real-time safety-critical applications. For efficient prediction of a system's performance, Transaction Level Modeling (TLM) is often used which increases the simulation speed by orders of magnitude. The speed advantage, however, comes at the cost of low accuracy.

In this paper, we use a novel modeling technique, called Result Oriented Modeling (ROM), which yields 100% accuracy in timing, yet approaches the same speed as traditional TLM. ROM also abstracts away internal details of the communication but, in contrast to TLM, fully maintains accurate timing. ROM optimistically predicts the timing and retroactively takes corrective measures, if necessary.

In this paper, we compare the ROM technique to TLM at different levels of abstraction, using a Controller Area Network (CAN) bus example. Our results show that ROM yields a simulation speedup close to the traditional TLM, yet exhibits the same timing accuracy as a bus functional model. Thus, for safety-critical real-time applications, ROM is a viable replacement for the inaccurate TLM.

Categories and Subject Descriptors: I.6.5 [Simulation and Modeling]: Model Development

General Terms: Design, Performance, Measurement.

Keywords: TLM, Transaction Level Model, System Level Design, ROM, Result Oriented Modeling, CAN, Controller Area Network, Real-Time Communication.

1. INTRODUCTION

System design faces a gap between the production capabilities and time-to-market pressures. While the improvements in production capabilities allow an increase in design complexity, shorter product life cycles force at the same time an aggressive reduction of the time-to-market. Recent system-level research aims to address this gap, for one by using abstract models.

Early stages of the design process especially require fast simulation. This need has pushed Transaction Level Modeling (TLM) [5], which utilizes abstract communication models that execute dramatically faster than synthesizable, bit-accurate models. However, TLMs usually have the drawback of significantly reduced accuracy. At the same time, accuracy is a necessary requirement for modeling of real-time safety-critical communication. Up to now, this prevented using fast executing TLMs for real-time communication.

To address this need for accuracy, we apply a novel modeling technique; Result Oriented Modeling (ROM). ROM delivers similar speed as TLM while retaining 100% timing accuracy at the same time.

ROM simulates communication, like a TLM, at the level of user transactions and transfers contiguous blocks of data with a single *memcpy*. In order to reach the apparently conflicting accuracy goal, ROM relies on the the basic assumption that the effects of communication are only visible at the boundaries of a user transaction. Therefore ROM can eliminate internal state changes, rearrange events and avoid costly context switches. It immediately predicts the *end result* and retroactively corrects for any *disturbing influence*.

In this paper, we will describe ROM for the Controller Area Network (CAN) [12], providing 100% accuracy, as required by a safety-critical automotive application, at highest simulation speed. As a result, simulation of real-time communication can now profit from TLM speeds.

1.1 Related Work

System level modeling has become an important research area aiming to improve the SoC design process. Languages for capturing SoC models have been developed, e.g. SystemC [5] and SpecC [3]. Capturing and designing communication architectures using TLM [5] has received much attention.

Sgroi et al. [15] address SoC communication with a Network-on-Chip approach that follows the OSI structure. [16] describes SystemC^{SV}, an extension to SystemC with three different abstraction levels and a CAN example in [1]. Coppola et al. [2] propose abstract communication modeling in the IPSIM framework. Gerstlauer et al. [4] describes abstraction based on a decreasing number of OSI [7] layers. A common point is the loss in accuracy with abstraction, which this paper eliminates. OCP-IP [9] provides three TLMs with increasing abstraction. Only their most detailed TL-1 is cycle accurate.

Real-time communication has been analyzed in previous work, [17] for example, analyzes different CAN controllers. System level modeling of real-time communication is not explored as much. [18] describes 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. [10] describe a similar transaction-based abstraction and introduce the concept of a model that is cycle count accurate at transaction boundaries. It too takes advantage of the limited observability to increase simulation performance. However, only a very limited speedup of 55% over the bus functional model is achieved[1].

In Section 2, we will introduce the general concept of Result Oriented Modeling, independent of its application to communication modeling. Then, we will describe the application of ROM to communication using the Controller Area Network in Section 3. We will also analyze the limitations of traditional communication modeling and show the advantages of the ROM approach. Section 4 provides experimental results that clearly support the claimed benefits of ROM. Section 5 concludes this paper with a summary.

2. RESULT ORIENTED MODELING

Result Oriented Modeling (ROM) is a general concept for abstract *and* accurate modeling of a process. On the highest level, it can be compared to the "black box" concept.

ROM uses the underlying assumption of limited observability. It is not necessary to show intermediate results of the process to the user, as in a "black box" approach. ROM produces only the *end result* of the process, not any intermediate states.

By hiding the internal states, ROM as the opportunity for optimization. Intermediate states can often be entirely eliminated. Instead, ROM utilizes an optimistic approach and predicts the outcome (e.g. termination time and final state) of the process already at the start.

A *disturbing influence* throughout the runtime of the process may change the system state. Thus, the initially predicted results may no longer be accurate. Therefore, ROM checks at the end of the predicted time whether such a *disturbing influence* has occurred. In such a case, ROM retroactively adjusts for the *disturbing influence* and takes corrective measures. In other words, a mistake of an overly optimistic initial prediction is fixed at the end. Optimistically predicting the *end result* reduces the amount of computation. Skipping internal states increases the execution performance, if the cost for any corrective measures is low.

In contrast, the traditional abstract modeling approach reaches the *end result* through a set of incremental state changes, where each incremental state change takes any *disturbing influence* into account and updates internal states accordingly. Decreasing the granularity of state changes results in a higher abstraction and improves speed, but reduces accuracy. 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 by the following key points:

1. The process user is not aware of 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 initial assumptions still hold true, and takes corrective measures otherwise.

[1] Our results show a speedup of four orders of magnitude.

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

To illustrate the ROM approach, lets consider a process of predicting the arrival time of an airplane. The real process exhibits continuous changes to the ground speed depending on the wind as *disturbing influence*. A traditional abstract model approximates the result by incrementally calculating the ground speed in dependence of the wind in (coarse-grain) discrete time steps. On the other hand, ROM does not model the intermediate airplane speed at all. Instead, it makes one initial optimistic prediction for the arrival time and corrects at the end the prediction retroactively for the average wind condition.

3. MODELING A CAN BUS USING ROM

With the general ROM concept in place, we will now describe the application of ROM to the modeling of a communication system. In particular we have chosen the CAN automotive bus as a real-time communication example, which we will introduce first. We will then show a set of layer-based CAN models as a reference, and ultimately apply the ROM approach and analyze its benefits.

3.1 Introduction to the CAN Bus

The Controller Area Network (CAN) is a real-time serial communications protocol, introduced by the Robert Bosch GmbH [12] with a focus on automotive applications.

CAN is a serial multi-node broadcast bus. Frames, with up to 8 bytes 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.

The frame identifier also defines the priority. If multiple senders attempt a transmission, the collision free CSMA/CA arbitration guarantees that the highest priority frame will succeed undisturbed. After the start of frame bit (see Figure 1), the frame identifier is transmitted with the most significant bit first as a sequence of dominant and recessive bus states. During transmission, each sender compares the send and receive signal and backs off when detecting a difference.

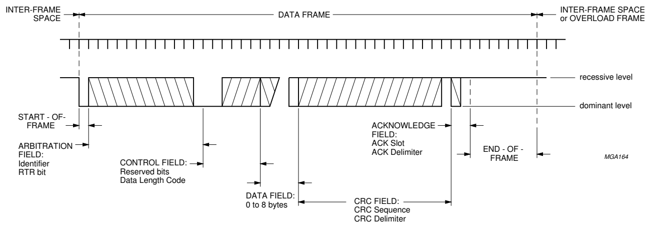
INTER-FRAME SPACE
DATA FRAME
INTER-FRAME SPACE or OVERLOAD FRAME
recessive level
dominant level
START - OF- FRAME
ARBITRATION FIELD: Identifier RTR bit
CONTROL FIELD: Reserved bits Data Length Code
DATA FIELD: 0 to 8 bytes
CRC FIELD: CRC Sequence CRC Delimiter
ACKNOWLEDGE FIELD: ACK Slot ACK Delimiter
END - OF - FRAME
MGA164

Figure 1: CAN data frame (source [11]).

In order to ensure correctness of the received data, each CAN message includes a 15-bit 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.

The following features are candidates for abstraction:

- Serial protocol
- Bit synchronization
- Error detection and confinement
- Bit error detection using a 15 Bit CRC
- Bit stuffing
- Arbitration, bus access controlled by CSMA/CA

Now, we will describe our layer-based modeling of the CAN. We chose for each model a subset of the above listed features.

3.2 Layer-based Modeling

Following the ISO OSI reference model [7], we can model CAN using a layered architecture [13]. The CAN specification then falls into the second layer, the data link layer. For modeling, we consider the 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, and transfers these byte blocks using the protocol layer. The *protocol layer* transfers data as *bus transactions* which are bus primitives (e.g. a CAN data frame with up to 8 bytes data). It uses the services of the *physical layer* which provide a *bus cycle* access to sample and drive the bus wires.

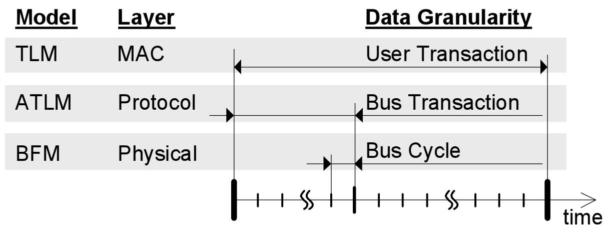

Figure 2: Layer-based bus modeling.

Figure 2 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 can define three models which we will refer to as TLM, ATLM, and BFM.

3.2.1 *Transaction Level Model (TLM)*

The TLM[2] is the most abstract model, implementing only the media access layer. Data, handled at the user transaction granularity, is transferred regardless of its size in one chunk using a single *memcpy*. Timing is simulated as a single *wait-for-time* statement, covering the entire user transaction. Neither CRC nor bit stuffing is observed, since both would require a bit inspection of each message. Arbitration is abstracted to a semaphore used once per user transaction.

3.2.2 *Arbitrated Transaction Level Model (ATLM)*

The ATLM models a bus access with CAN frames at the protocol level. It uses the MAC layer implementation of the bus functional model to split user transactions into bus transactions.

The ATLM accurately models the arbitration for each bus transaction (CAN frame) based on the message identifier. It collects all requests during start of frame, and proceeds with the highest priority message. The ATLM performs a bitwise inspection of the frame in order to calculate the CRC and perform stuff bit handling: a stuff bit is inserted/removed each time 5 bits of equal polarity are found. Thus, the physical frame length depends on the frame content.

[2] Note that TLM is not clearly defined in the literature. For this paper, we will use TLM as the name of the model at the granularity of an entire user transaction.

Table 1: Models and supported features.

Feature	BFM	ATLM	TLM
serial transmission, bit sync	yes	no	no
error detection, confinement	yes	no	no
CRC calculation, bit stuffing	yes	yes	no
CSMA/CA arbitration	yes	yes	no

3.2.3 *Bus Functional Model (BFM)*

The BFM is a synthesizable, cycle- and pin-accurate bus model. It implements all layers and covers all timing and functional properties of the bus definition.

The bus functional model implements all features of the specification. It protects the data by the CRC, handles stuff bits and performs arbitration. The frame data is sent and received serially and the node clocks are synchronized to the bit stream according to [12] and [6].

Table 1 outlines the features implemented by each model.

3.3 Result Oriented Modeling

Now, we will apply the ROM approach to model the CAN aiming at 100% accuracy *and* high simulation performance at the same time.

3.3.1 *Assumptions as with TLM*

As we have introduced earlier, ROM is based on separation of computation and communication. It hides the communication internals from the user and avoids using signals and individual wires. Instead, it implements data transfers by use of a single *memcpy* operation. As such, ROM uses the same principles as TLM.

An application using a ROM model, 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. The application has no knowledge about splitting the user transaction into individual CAN frames and CSMA/CA arbitration.

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 bus functional model.

As in TLM, the main idea for 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 increased execution performance. This avoids running the scheduling algorithm in the simulation engine and thus also reduces the number of context switches.

3.3.2 *Optimistic Approach*

By use of an optimistic approach, ROM can freely rearrange and/or omit internal events and state changes within a transaction to eliminate costly context switches in the simulator. When a node requests a user transaction, the earliest finish time for this transfer is calculated and the node 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 node verifies whether the predicted time is still accurate. If so, the transaction is complete. Note that in this best case scenario only a single wait statement is used (as in the TLM).

To illustrate the advantage over the layer-based approach, let us consider four CAN nodes simultaneously requesting transmission of 16 bytes, resulting in two CAN frames each.

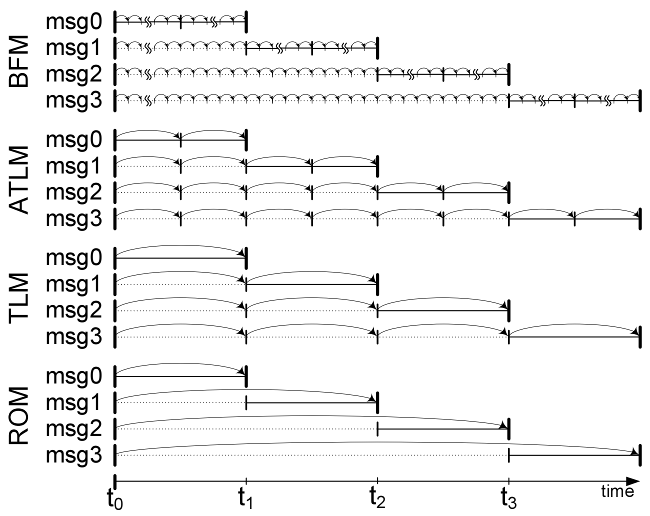

Figure 3: 4 Nodes send 16bytes in two CAN frames.

Figure 3 shows the *wait-for-time* statements as arcs for each model and Table 2 compares the numbers. All nodes request the transmission at t_0. The messages have decreasing priority. The highest priority *msg0* immediately gains bus access at t_0, while all others lose arbitration and retry again. The lowest priority *msg3* is delayed the longest until t_3, when all higher priority messages are completed.

The **BFM** constantly samples the serial bus to synchronize its clock and to maintain the frame structure. Synchronization requires oversampling[3], which results in the large amount of *wait-for-time* statements, almost 28 thousand for the nodes in this example. The **ATLM** operates abstractly on a bus transaction (CAN frame) granularity. It therefore waits once per frame and checks the CSMA/CA arbitration. If a node looses arbitration for one frame, it delays until start of the next frame for a new arbitration attempt (e.g. 6 attempts for *msg3*). Due to the abstract simulation, much fewer *wait-for-time* statements (20) are needed. The **TLM** simulates with a user transaction granularity. It arbitrates only at the beginning of a user transaction and thus exhibits even fewer *wait-for-time* statements (10). The **ROM** uses the least number of *wait-for-time* statements (4) in this optimal case. Since all requests are already known at t_0, ROM can immediately predict the accurate end time resulting in only a single *wait-for-time* statement per message.

Table 2: Wait complexity, optimal case.

	BFM	ATLM	TLM	ROM
wait-for-time	27,972	20	10	4
(in percent)	100%	0.071%	0.036%	0.014%

Reducing the usage of *wait-for-time*, which typically results in a costly context switch, improves the performance of a model. We expect the serial transmitting BFM with the most *wait-for-time* statements to be the slowest, and ROM to be the fastest. However, in order to accurately predict the bit length of a CAN frame, ROM needs to calculate the CRC and apply the bit stuffing rule. This results in a costly bit inspection and prevents ROM from outperforming TLM.

[3]Each bit on the bus is oversampled (e.g. 12 times, [6]) in order to detect the rising edge and for clock synchronization.

3.3.3 Adjusting for Disturbing Influence

In order to guarantee accurate timing, ROM verifies the initial optimistic prediction at the end of the predicted time. A higher priority message starting during a multi-frame low priority message will delay the low priority message. Then the initially predicted time will be too short. ROM detects this *disturbing influence* at the end of the low priority message, recalculates the predicted time and waits for it. This process is repeated until the prediction is verified to be correct. To minimize the effort for an update, ROM stores the physical length of each frame during the initial prediction and avoids the costly bit inspection during an update.

Note that an optimistic (short) prediction algorithm is necessary to allow for corrections. With a pessimistic (too long) prediction, a correction would need to go back in time, which obviously is not possible.

To compare ROM against the layered models, we analyze an example with *disturbing influence* as shown in Figure 4. A low priority *msg2* with 4 frames starts at t_0. During its second frame at t_1 a high priority *msg1* is released, which delays the completion of *msg2*.

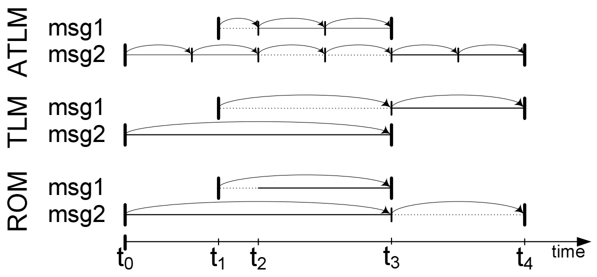

Figure 4: Contention in ATLM, TLM and ROM

In **ATLM**[4], the delay is correctly modeled. *Msg1* waits until the start of the next frame at t_2 when it wins the arbitration. *Msg2* looses the arbitration for its 3^{rd} 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 9 wait-for-time statements.

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 3 *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 higher priority transaction 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 finds no record of any higher priority transaction and completes its transfer. ROM is able to correctly simulate the example with only 3 wait-for-time statements.

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

Table 3 compares the wait-for-time statements performed by the models. ROM waits as often as the TLM, four orders of magnitude less frequently than the BFM.

Table 3: Wait complexity with disturbance.

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

3.3.4 *Multiple Prediction Updates*

Transferring long messages (split into many CAN frames) may require multiple prediction updates, due to multiple higher priority requests. Figure 5 shows an example where a long transaction is frequently delayed. Although there are 7 *high* priority requests during the *low* priority transfer, ROM needs only 3 prediction updates. A closer looks shows 4 requests during the initially predicted period, 2 in the next and finally only one. This exponential drop indicates that, for most transfers, only very few prediction updates are expected, even under high bus load.

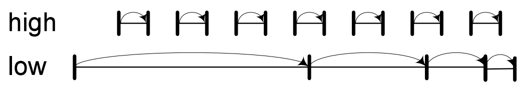

Figure 5: Exponentially decreasing updates.

3.3.5 *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 higher effort from the model developer.

4. EXPERIMENTAL RESULTS

To validate the benefits of our proposed ROM approach, we have implemented all four models in the SpecC SLDL [3][5] and executed them on the unmodified reference simulator for a detailed analysis. We examine three aspects: the accuracy as a necessary requirement for simulating real-time communication, the number of prediction updates, and the simulation performance.

4.1 Timing Accuracy

We use a test setup with 8 nodes[6] connected to the CAN bus. Four nodes act as senders and four nodes as receivers. Each sender uses an exclusive range of message identifiers. Since the message identifier defines the priority, we also refer to the senders by the priority of their messages.

For measuring the timing accuracy, we have established a setup with linear random distributed transfers. Each node sends a set of 5000 predefined messages that vary in message id (each from its own range), size (1-100 bytes), content and delay between two transfers. The same set of messages is transfered by each model, and we repeat the test for different amounts of bus contention. We record the duration for each individual transfer and compare them later against the cycle-accurate BFM.

[5] The ROM concept is not specific to SpecC. It is equally applicable to other SLDLs, like SystemC.

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

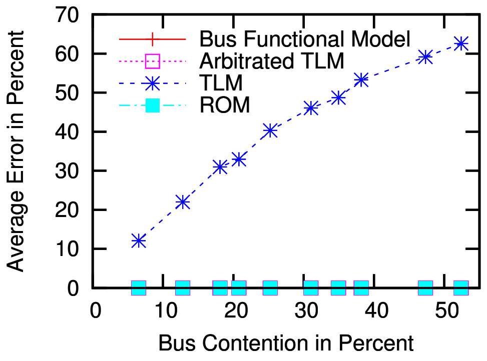

Figure 6: Accuracy for lowest priority node.

Figure 6 shows the average error in transaction duration for the node sending the lowest priority messages over a varying degree of bus contention. As targeted, the ROM shows 0% error for all measurements, lying right on top of the x axis (same as the BFM and the ATLM). Since CAN frames cannot be preempted, the ATLM is accurate. Arbitrating once per frame already yields accurate results.

In contrast, the TLM shows significant error rates, linear increasing with growing bus contention, passing 60% error at 50% contention. This inaccuracy does not allow the TLM to be used for accurate real-time communication simulation.

4.2 Prediction Updates

Previously we have stated that multiple prediction updates may be necessary for the accurate ROM. Therefore, we quantify now the number of updates in the same random transfer setup.

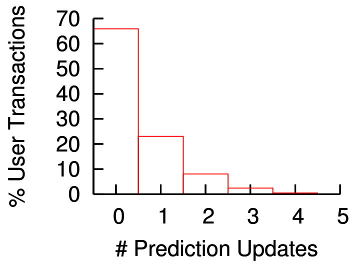

Figure 7: Prediction updates at 51% contention.

Figure 7 shows the histogram of prediction updates for the condition with the most updates: the node with the lowest priority messages at high bus contention. As expected, the number of transactions that require prediction updates reduces exponentially. 66% have no prediction updates (with the majority of single frame messages). 23% require one prediction update and only 0.5% need 4 updates. This distribution clearly shows that a low number of prediction updates can be generally expected for ROM.

4.3 Performance

Now, we analyze the performance advantage of ROM. In our scenario, three high priority senders produce a constant bus load totaling 50% by transferring 16 byte messages. We actually measure the low priority sender, which issues transactions of increasing size without delay in between[7]. We

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

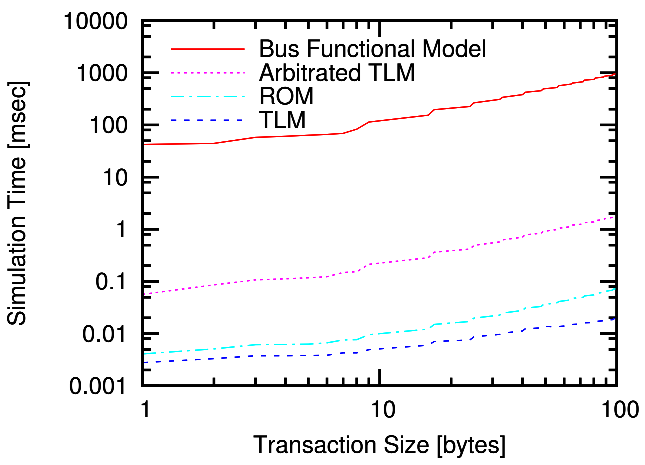

Figure 8: Transfer time with 50% base utilization.

analyze the simulation time for the system consisting of 4 senders and 4 receivers on a Pentium 4 at 2.8 GHz.

Figure 8 shows the simulation time of the low priority sender over an increasing message size while the high priority senders are running at the same time. All models exhibit a characteristic increase in simulation time when exceeding a size divisible by 8. Another CAN frame is needed, which increases simulation effort and increases the probability of additional higher priority messages.

ROM and TLM execute five orders of magnitude faster than the BFM. The TLM reaches this speed (0.006ms 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, it is only two times slower than the TLM: 0.012 ms for a 16 byte transaction. 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. In conclusion, ROM is the fastest to accurately model the communication, 12 700x faster than the BFM.

Note that in [14], we have also analyzed a standard on-chip bus architecture that does not require bit inspection. There, ROM showed an identical high performance as TLM.

The system simulation bandwidth, the total bandwidth of all senders, may be as well of interest to judge simulation performance. Table 4 lists how many MBytes are simulated per second of real time for an increasing number of nodes, each issuing 16 byte transfers. It reveals that ROM is fast and scales well with an increased number of nodes.

Table 4: System simulation bandwidth $[\frac{MBytes}{sec}]$

	BFM	ATLM	ROM	TLM
2 Nodes	0.0005	0.117	3.51	12.7
4 Nodes	0.0003	0.113	2.97	6.6
8 Nodes	0.0002	0.110	2.57	5.2

5. CONCLUSION

In this paper, we have applied a novel modeling concept, Result Oriented Modeling (ROM), to modeling of real-time safety-critical communication in embedded system design.

ROM is a novel modeling approach that hides internal state changes like a TLM. Moreover, ROM utilizes an optimistic paradigm and predicts the end result already at the start. It takes corrective measures at the end, if a disturbing influence has occurred, and reaches 100% accuracy.

We have implemented the ROM concept for the CAN automotive bus and compared the new model against traditional layer-based models. Our experimental results show tremendous benefits of ROM. While the traditional TLM suffers from a significant loss in accuracy, ROM delivers 100% accuracy and high execution speed (12 700x faster than the BFM) at the same time. Therefore, ROM is a viable and attractive solution for accurate simulation of real-time communication at TLM speed.

6. REFERENCES

[1] D. Brem and D. Müller. Interface based system modeling of a CAN using SVE. In *Proceedings of the EkompaSS Workshop*, Hanover, Germany, April 2003.

[2] M. Coppola, S. Curaba, M. Grammatikakis, and G. Maruccia. IPSIM: SystemC 3.0 enhancements for communication refinement. In *DATE*, Munich, Germany, March 2003.

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

[4] A. Gerstlauer, D. Shin, R. Doemer, and D. Gajski. System-Level Communication Modeling for Network-on-Chip Synthesis. In *ASPDAC*, Shanghai, China, January 2005.

[5] T. Grötker, S. Liao, G. Martin, and S. Swan. *System Design with SystemC*. Kluwer Academic Publishers, 2002.

[6] F. Hartwich and A. Bassemir. The Configuration of the CAN Bit Timing. www.can.bosch.com, 1999.

[7] Internation Organization for Standardization (ISO). *Reference Model of Open System Interconnection (OSI)*, second edition, 1994. ISO/IEC 7498 Standard.

[8] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Applications*. Kluwer Academic, 1997.

[9] OCP-IP. Open Cores Protocol. www.ocpip.org.

[10] S. Pasricha, N. Dutt, and M. Ben-Romdhane. Fast exploration of bus-based on-chip communication architectures. In *CODES and ISSS*, Stockholm, Sweden, September 2004.

[11] Philips. P8xc592: 8-bit microcontroller with on-chip can. www.semiconductors.philips.com, 1996.

[12] Robert Bosch GmbH. *CAN Specification*, 2.0 edition, 1991. www.can.bosch.com.

[13] G. Schirner and R. Dömer. Abstract Communication Modeling: A Case Study Using the CAN Automotive Bus. In A. Rettberg, M. Zanella, and F. Rammig, editors, *From Specification to Embedded Systems Application*, Manaus, Brazil, August 2005. Springer.

[14] G. Schirner and R. Dömer. Fast and Accurate Transaction Level Models using Result Oriented Modeling. In *ICCAD*, San Jose, CA, USA, November 2006.

[15] 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 *DAC*, June 2001.

[16] R. Siegmund and D. Müller. SystemC^{SV}: An extension of SystemC for mixed multi-level communication modeling and interface-based system design. In *DATE*, Munich, Germany, March 2001.

[17] K. W. Tindell, H. Hansson, and A. J. Wellings. Analysing Real-Time Communications: Controller Area Network (CAN). In *Real-Time Systems Symposium*, December 1994.

[18] P. van der Putten, J. Voeten, M. Geilen, and M. Stevens. System level models for real-time communication. In *EUROMICRO*, September 1999.