Transaction Level Modeling of Computation

Rainer Dömer

Technical Report CECS-06-11
August 17, 2006

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA
(949) 824-9007

doemer@uci.edu
http://www.cecs.uci.edu/

Transaction Level Modeling of Computation

Rainer Dömer

Technical Report CECS-06-11
August 17, 2006

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA
(949) 824-9007

doemer@uci.edu
http://www.cecs.uci.edu

Abstract

The design of embedded computing systems faces great challenges due to the huge complexity of these systems. The design complexity grows exponentially with the increasing number of components that have to cooperate properly. One solution to address the complexity problem is the modeling at higher levels of abstraction. However, it is generally not clear which features to abstract (and to what extend), nor how to use the remaining features to create an executable model that allows meaningful, efficient and accurate analysis of the intended system.

Transaction Level Modeling (TLM) is widely accepted as an efficient technique for abstract modeling of communication. TLM offers gains in simulation speed of up to four orders of magnitude, usually however, at the price of low accuracy. So far, TLM as been used exclusively for communication.

In this work, we propose to apply the concepts of TLM to computation. While the two aspects of embedded system models, computation and communication, are different in nature, both share the same concepts, namely functionality and timing. Thus, TLM, which is based on the separation of functionality and timing, is equally applicable to both, communication and computation. In turn, the tremendous advantages of TLM can be utilized as well for the abstract modeling of computation.

While traditional work largely has focused on refinement and synthesis tasks, this work addresses the modeling of systems towards efficient accurate estimation and rapid design space exploration. The results of this work will be directly applicable to established design flows in the industry.

Contents

Transaction Level Modeling of Computation

Rainer Dömer

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA

doemer@uci.edu
http://www.cecs.uci.edu

Abstract

The design of embedded computing systems faces great challenges due to the huge complexity of these systems. The design complexity grows exponentially with the increasing number of components that have to cooperate properly. One solution to address the complexity problem is the modeling at higher levels of abstraction. However, it is generally not clear which features to abstract (and to what extend), nor how to use the remaining features to create an executable model that allows meaningful, efficient and accurate analysis of the intended system.

Transaction Level Modeling (TLM) is widely accepted as an efficient technique for abstract modeling of communication. TLM offers gains in simulation speed of up to four orders of magnitude, usually however, at the price of low accuracy. So far, TLM as been used exclusively for communication.

In this work, we propose to apply the concepts of TLM to computation. While the two aspects of embedded system models, computation and communication, are different in nature, both share the same concepts, namely functionality and timing. Thus, TLM, which is based on the separation of functionality and timing, is equally applicable to both, communication and computation. In turn, the tremendous advantages of TLM can be utilized as well for the abstract modeling of computation.

While traditional work largely has focused on refinement and synthesis tasks, this work addresses the modeling of systems towards efficient accurate estimation and rapid design space exploration. The results of this work will be directly applicable to established design flows in the industry.

1 Introduction

As we enter the information era, embedded computing systems have a profound impact on our everyday life and our entire society. With applications ranging from smart home appliances to video-enabled mobile phones, from real-time automotive applications to communication satellites, and from portable multimedia components to reliable medical devices, we interact and depend on embedded systems on a daily basis.

While embedded systems typically do not look like computers due to their hidden integration into larger products, they contain similar hardware and software components regular computers are made of. Moreover, embedded systems are special-purpose devices, dedicated to one predefined application and face stringent constraints including high reliability, low power, hard real-time, and low cost.

Over recent years, embedded systems have gained a tremendous amount of functionality and processing power, and at the same time, can now be integrated into a single System-on-Chip (SoC). The design of such systems, however, faces great challenges due to the huge complexity of these systems. The system complexity grows rapidly with the increasing number of components that have to cooperate properly. In addition, expectations grow while constraints are tightened. Last but not least, customer demand constantly requires a shorter time-to-market and thus puts high pressure on designers to reduce the design time for embedded systems.

1.1 Motivation

The international Semiconductor Industry Association ITRS, in its design roadmap [67], predicts a significant productivity gap and anticipates tremendous challenges for the semiconductor industry in the near future. The 2004 update of the roadmap identifies system-level design as a major challenge to advance the design process. *System complexity* is listed as the top-most challenge in system-level design. As first promising solution to tackle the design complexity, the ITRS lists *higher-level abstraction and specification*. In other words, the most relevant driver to address the system complexity challenge is system-on-chip design using specification at higher levels of abstraction.

Higher-level of abstraction is in the ITRS report [67] also listed as a key long-term challenge for design verification. As simulation and synthesis, verification will have to keep up with the move to higher abstraction levels. In other words, system models must be properly abstracted with the intended design tasks in mind, including verification and synthesis.

1.2 Abstract Modeling

In system level design, the importance of abstract specification and modeling cannot be overemphasized. Proper abstraction and modeling is the key to efficient and accurate estimation and successful design space exploration. However, in contrast to the great significance of abstraction and modeling, most research has focused on tasks *after* the design specification phase such as simulation and synthesis. Little has been done to actually address the modeling problem, likely because this problem is not well-defined yet, and the quality of a model is not straightforward to measure and compare.

A model is an abstraction of reality. More specifically, an embedded system model is an abstract representation of an actual or intended system. Only a well-designed model will accurately represent and define the properties of the end product, while allowing efficient handling and fast examination.

Moreover, most embedded system models are executable. Execution of the model allows to simulate the behavior of the intended system and to measure properties beyond the immediate ones in the model description. For example, simulation enables the prediction of properties such as performance and throughput.

The choice of the proper abstraction level is critical. Ideally, multiple well-defined abstraction levels are needed to enable gradual system refinement and synthesis, adding more detail to the design model with every step. In other words, a perfect model retains only the essential properties needed for the job at hand, and abstracts away all unneeded features. Then, as the design process continues, incrementally more features are added to the model, reflecting the design decisions taken.

The main objective of this work is to improve the modeling of embedded systems, in particular the modeling of computation (as discussed later in Section 3). The goal of a "good" model is, with minimum modeling effort, to allow fast and accurate prediction of critical properties of the described system.

To achieve these goals, well-defined metrics and measurement setups need to be defined. Then, a systematic analysis of actual system models is necessary, so that essential properties and proper abstraction levels can be identified and efficient modeling techniques and guidelines can be developed.

1.2.1 Is modeling an art?

Sometimes, people think that modeling is an art or a task that requires artistic talents in the model designer. This may be true for creating a painting or designing a sculpture, but does not hold for modeling of embedded systems. The quality of a piece of art cannot be objectively determined, but the quality of an embedded system model can be quantitatively measured and compared.

To measure the properties of a model (and the properties of the modeled system), metrics, measurement setups, and test environments are necessary. However, these can be clearly defined (as shown later in

Section 2.2.4, for example) and used to examine the model. Thus, embedded system modeling is not an art, it is a technical task based on scientific principles and concepts. Further, embedded system modeling can be learned simply by following technical modeling guidelines, the creation of which is one objective of this work.

1.2.2 What features should be abstracted?

It has been stated before that proper abstraction is the key to successful modeling of embedded systems. However, it is generally not obvious which features in a model are needed, and which can be abstracted away (and to what extend). Neither is it clear how to model the essential features in order to create an efficient executable model suitable for fast and accurate prediction and rapid design space exploration. This problem is aggravated by the fact that most desired features typically pose a trade-off, or are even contradictory (e.g. high simulation speed and detailed functionality).

It is the goal of this work to identify and include only desired features in models for common design scenarios (and to leave unneeded features out). Thus, we aim to identify minimal feature sets for effective models, and also develop modeling guidelines that maximize the value of a model such that it provides high simulation speed, accurate functionality and/or precise timing.

Careful measurements and systematic analysis will be necessary to identify and effectively navigate the trade-offs involved. For this work, we will focus on performance and accuracy metrics, and measure these properties using industry-standard applications such as MP3 codecs and video processing algorithms.

For most embedded applications, accurate timing and correct functionality are ”must-have” features in a model. Other important aspects include structural composition (such as the number and types of processing elements) and estimated power consumption (e.g. for battery-powered mobile devices). On the other hand, implementation details, such as pins, waveforms and interrupts, are abstracted away in specification models as these restrict the implementation, limit the design space, and unnecessarily increase the model complexity.

1.2.3 Architecture analogy

The architectural blueprints of a house can serve as a good analogy of well-abstracted modeling. An architect, charged with designing a new building, typically develops a set of models of her/his intended design in order to examine, document and exhibit the intended building features, the general floorplan, room sizes, etc. For example, to exhibit the aesthetic qualities of the design, the architect often builds an abstract paper model of the building that shows the three-dimensional design in small scale. For the actual building phase, however, a different model is needed. Here, the architect draws two-dimensional schematic blueprints for each floor which accurately show the location and dimensions of the walls, doors and windows, as well as water and gas pipes, etc.

In this analogy, the purpose of the abstract models is clear, as is the inclusion or omission of design features. Different models serve different purposes and therefore exhibit different properties. Important aspects are included and emphasized, and aspects of no interest are abstracted away.

Concluding the overview, the contribution of this work will be a systematic approach to abstraction and modeling of computation in embedded systems.

1.3 Outline

The remainder of this document further details this work. Section 2 discusses background material in order to appropriately position this work with respect to ealier and related work. Section 3 presents the main idea of modeling computation using TLM and outlines the technical approach to be taken. Then, Section 4 discusses the current status of this approach and outlines future work. Finally, Section 5 summarizes this document and concludes with the expected impact of this work.

2 Background

The following sections present an executive summary of four research tasks addressed by the author that build the background for TLM of computation, in the context of other related work.

2.1 Separation of Concerns

One of the fundamental principles in embedded system modeling is the clear *separation of concerns*. Addressing separate issues independently from each other very often leads to clear advantages over approaches where different aspects are intermixed. In particular, this applies to the design of system-level description languages (SLDL).

An excellent example of the separation of concerns is the SpecC [28, 31] approach defined in 1997 [82]. The SpecC language and methodology created a world-wide impact in industry and academia so that leading companies started the international SpecC Technology Open Consortium (STOC) [73] in 1999.

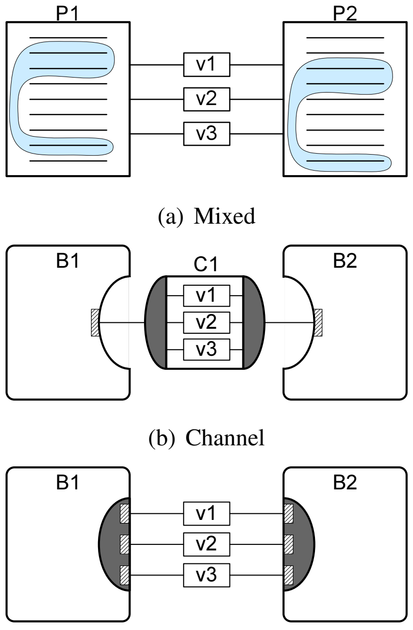
The SpecC language is based upon the clear separation of computation and communication, as well as on an orthogonal structure, which we will both briefly review in the following two sections.

2.1.1 Computation and communication

Clear separation of computation and communication is an essential feature of any good system model as it enables ”plug-and-play”. This feature also distinguishes modern SLDLs from traditional hardware description languages (HDL) such as VHDL [39] and Verilog [40].

A typical model of two communicating processes described in a HDL is shown in Figure 1(a). Here it is important to note that there are two distinct portions of code in the blocks, containing computation and communication (shaded), which are intermixed. In such a model, there is no way to automatically distinguish the code for communication from the code used for computation, because the purpose of the statements cannot be identified. Neither is it possible to automatically exchange the communication protocol, nor to switch to a new algorithm to perform the computation.

In order to allow automatic replacement of communication protocols and computation algorithms, *separation* of communication and computation is needed. This is supported in form of *behaviors* and *channels* in the SpecC language [27], as shown in Figure 1(b). Here, the computation is encapsulated in the behaviors and the communication is contained in the channel.



(a) Mixed

(b) Channel

(c) Inlined

Figure 1: Separation of computation and communication.

For the implementation of a channel, its functions are *inlined* into the connected behaviors and the encapsulated communication signals are exposed. This is illustrated in Figure 1(c). The channel has disappeared, the contained signals are exposed, and the communication protocol has been inlined into the behaviors. Note that in this final implementation model, communication and computation are no longer separated. The model again resembles the traditional HDL model, ready for implementation.

In summary, modern SLDLs clearly separate communication from computation such that both can be easily replaced if necessary (”plug-and-play”).

It should be emphasized that this separation of concerns, which originated in SpecC, has also been adopted in the SystemC [36] SLDL, version 2.0, in 2000 [54]. Today, SystemC is the de-facto industry-standard SLDL, supported by the Open SystemC Initiative [56]. Finally, it should be mentioned that the significance of the *separation of concerns* has later been emphasized as well in the Metropolis project at UC Berkeley [6].

2.1.2 Orthogonality of concepts

A second significant principle in system-level design is the orthogonality of concepts. Again, SpecC is a good example as it is based on the paradigm of providing orthogonal constructs for orthogonal concepts.

Before designing the SpecC language, the authors identified the essential concepts required for system level design, including concurrency, hierarchy, communication, synchronization, exception handling, and timing [28]. Since in principle these concepts are independent of each other, i.e. orthogonal, the SpecC language was designed to provide exactly one independent syntactical construct for each concept. As a result, the SpecC language largely[1] is orthogonal in its basic constructs. This is extremly beneficial for CAD applications as this significantly simplifies the development of the tools working with the language.

In contrast, VHDL [39] can serve as a counter example. In VHDL, signals incorporate synchronization, data storage and timing. Without additional annotations or coding conventions, this makes it hard to identify for which purpose a particular signal is actually used, and thus an efficient implementation is aggravated.

2.2 Transaction Level Modeling of Communication

Our previous work, Transaction Level Modeling for Communication [65], can be seen as the counterpart of the work described in this document. The former has been applied to the communication, the latter will be applied to the computation aspects of embedded systems. As we have seen in Section 2.1.1, embedded system models consist of exactly these two parts, computation and communication. Thus, the combination of the former and this work will solve the whole modeling problem.

[1] This author later realized that the goal of orthogonality has not been completely reached in SpecC. Behavioral and structural hierarchy are not entirely independently implemented from concurrency.

2.2.1 TLM trade-off

Accurate communication modeling is an important issue for the design of embedded systems. However, efficient system level design requires also high execution performance.

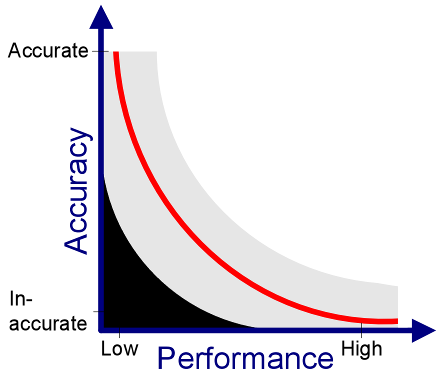


Figure 2: Transaction Level Modeling Trade-Off.

For efficient communication modeling, Transaction Level Modeling (TLM) has been proposed [36]. TLM abstracts the communication in a system to whole transactions, abstracting away low level details about pins, wires and waveforms [17]. This results in models that execute dramatically faster than synthesizable, bit-accurate models. This benefit, however, usually comes at the price of low accuracy.

In general, TLMs pose a trade-off between an improvement in simulation speed and a loss in accuracy, as illustrated in Figure 2. The trade-off 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. Models with this trade-off 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.

2.2.2 Systematic analysis

Although TLM has been generally accepted as one solution to tackle SoC design complexity, the TLM trade-off however, has not been examined in detail. Our aim is to systematically study and analyze the

TLM trade-off quantitatively. More specifically, we quantify the performance gains of TLM and measure the loss in accuracy for a wide range of bus systems. As a first step, we define our approach to TLM as based on the granularity of data and arbitration handling. We then define proper metrics and test setups for measuring the performance improvement and the accuracy loss. We apply our TLM abstraction approach to examples of different bus categories and create models at different abstraction levels, in particular two classes of TLMs (ATLM and TLM), and compare them against a fully accurate Bus Functional Model (BFM) as a reference.

2.2.3 Modeling approach

TLM allows a wide variety of modeling styles and abstractions [17]. For our modeling, we focus on the *granularity* of data and arbitration handling. We define three classes of granularity applicable to any bus protocol, and match these granularity classes to three model types. Figure 3 shows the granularity levels with respect to time and indicates the correlation to models and layers. A *user transaction* is the most coarse grain element of transferring a contiguous block of bytes with arbitrary length. It is split into *bus transactions*, which are bus transfer primitives, such as a word transfer. Each bus transaction is usually processed in several *bus cycles*, which represent the finest granularity in our modeling.

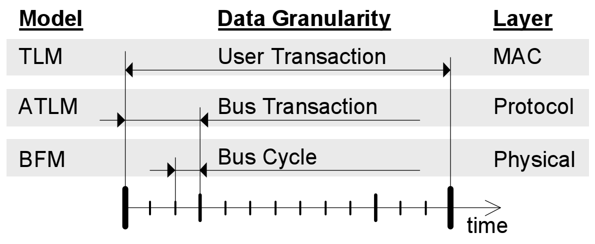


Figure 3: Model classes and their granularity.

Our classes of granularity correlate with the layers defined in the ISO OSI reference model [41]: the media access control (MAC), the protocol sublayer, and the physical layer. Each layer handles data and arbitration at its own granularity. According to our classes of granularity, we consider models at three different abstraction levels.

Transaction Level Model (TLM) The TLM is the most abstract model; it handles user data at the user transaction granularity and transfers data regardless of its size using a single *memcpy* and simulates timing by a single *wait-for-time* statement. Instead of modeling the bus arbitration, it resolves concurrent access using a semaphore.

Arbitrated TLM (ATLM) The ATLM simulates the bus access with bus transaction granularity (e.g. AHB bus primitives), at the protocol layer level. The ATLM accurately models the bus arbitration for each bus transaction.

Bus Functional Model (BFM) The BFM is a synthesizable, bus cycle-accurate and pin-accurate bus model. It implements all layers down to the physical layer, covering all timing and functional properties of the bus. It handles arbitration per bus transaction and has the capability to take arbitration decisions on a bus cycle granularity.

2.2.4 Analysis metrics and measurement setup

We focus on two aspects for the analysis: the simulation *performance*, since a performance gain is the main premise of TLM, and the *timing accuracy*, as a loss is expected with abstraction. Our metric for the performance is the simulation bandwidth. We define bandwidth as the amount of data transferred in the simulation per second of real-time, using a minimal scenario with a single master and a single slave. One aspect of model accuracy is the timing accuracy for each transmitted user transaction. As one metric, we utilize the transfer duration per user transaction and define the *duration error* as the percentage error over the bus standard, so that a timing accurate model exhibits 0% error:

$$\begin{aligned} d_{std} &: \quad \text{duration as per standard} \\ d_{test} &: \quad \text{duration in model under test} \\ error_i &= 100 * \frac{|d_{test} - d_{std}|}{d_{std}} \qquad (1) \end{aligned}$$

Since the actual experienced timing accuracy highly depends on the application and architecture specifics, we define a generic test setup and a procedure that covers a range of applications. The designer

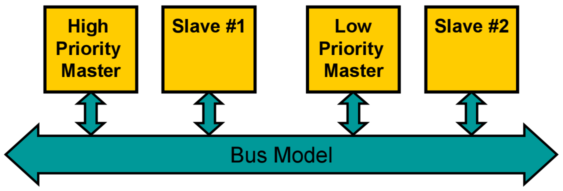


Figure 4: Dual master setup for accuracy measurements.

can then derive the expected accuracy for her/his particular setup.

In a generic test setup with two masters and two slaves connected to the same bus (Figure 4), each master transfers a predefined set of user transactions. The transactions vary linear randomly in the base address, size, and the delay to the next transaction (simulating the application's computation). We record the timing information of each individual transaction for later analysis. We repeat the analysis over different levels of bus contention, which allows to infer the accuracy over a range of applications.

2.2.5 Results

We have applied our granularity-based abstraction to three common bus systems covering diverse communication protocols. We have chosen the AHB of AMBA [3], as a representative of parallel on-chip bus systems with centralized industry-accepted standard for on-chip bus systems. For the second category of off-chip serial busses with distributed arbitration, we have selected the Controller Area Network (CAN) [11]. This bus system dominates in automotive applications. Third, we analyze the category of custom processor-specific busses that are typically much simpler than the general purpose standard busses. Here, we have chosen the Motorola ColdFire Master Bus [53] that is used by the popular ColdFire MCF5206 processor. We have modeled, validated, and systematically analyzed each bus using our performance and accuracy metrics.

Figure 5 shows the performance results for one example, the AMBA AHB. It confirms the TLM expectations: the simulation speed increases significantly with abstraction. The performance raises with each TLM abstraction by two orders of magnitude. As expected, the TLM executes the fastest. The simulation bandwidth is independent of the transaction size, since a constant number of operations is executed for each transfer. The ATLMs are two orders of magnitude slower due to the finer granularity of modeling individual bus transactions. Starting with the ATLMs, the graphs exhibit a saw tooth shape due to the non-linear split of user transactions into bus transactions (e.g. 3 bytes are transferred in 2 bus transactions: byte + short, whereas 4 bytes are transferred in 1 bus transaction: a word). The BFM is again two orders of magnitude slower than the ATLMs due to the fine grain modeling of individual wires and additional active components (e.g. multiplexers).

On the other hand, the accuracy significantly degrades with abstraction. One example, again from the AMBA AHB, is shown in Figure 6. The x-axis denotes the amount of bus contention in percent, each measurement point for a model reflects the average error over the analyzed 5000 user transactions. The actual accuracy depends strongly on the bus contention in the measurement scenario. Under the absence of bus contention, all models are accurate. For the abstract models ATLM and TLM, the timing error increases with a rising bus contention. The finer grained ATLM (irrespective of the (a) and (b) variant that we have analyzed), is more accurate than the most abstract TLM. The TLM peaks with 45% error at 50% bus contention due to its coarse grained simulation at the user transaction granularity and the absence of arbitration.

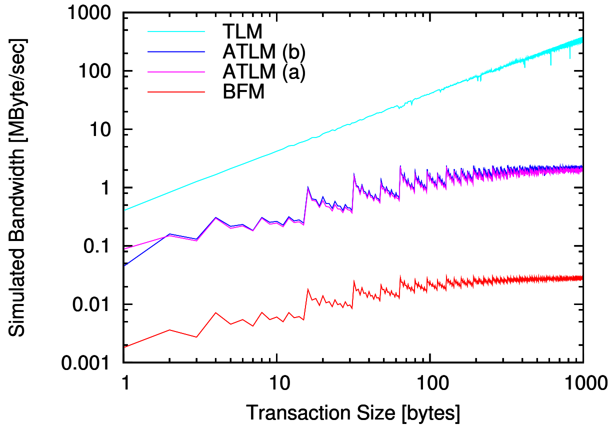


Figure 5: Performance for the AMBA AHB models.

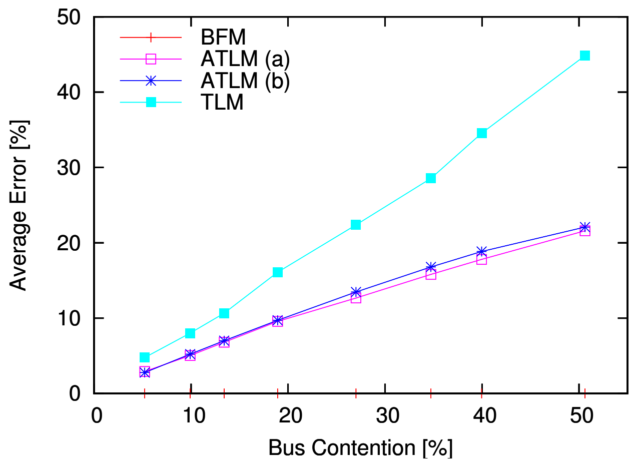


Figure 6: Individual timing accuracy for the AMBA AHB models.

2.2.6 Generalization

Combining the performance and accuracy results of all modeled bus systems, we generalize this data for a broader perspective on the benefits and drawbacks of TLM for communication. Figure 7 combines our measurements and depicts the TLM trade-off for the three representative bus models. On the x-axis, Figure 7 denotes the simulation performance in terms of simulation bandwidth for a 100 byte user transaction. The y-axis denotes the accuracy as the average error of individual transfers. The error is measured for the low priority master for a contention of 40%. Note that the error inceases towards the x-axis (downwards), hence a measurement point on the top signifies 100% accuracy.

The graph confirms the expected results as depicted earlier in Figure 2. Abstract modeling poses a trade-off, with either fast or accurate results. The accurate BFMs are slow with less than 0.2 MByte/s bandwidth. On the other hand, the fast TLMs with a bandwidth of up to 100 MByte/s produce an average error from 32% to 47%. The ATLMs are found in the middle, simulating at about 1 MByte/s bandwidth. Some are accurate, since the modeled granularity matches the granularity of the actual protocol. If the granularity does not match, or in case of additional feature abstraction, the ATLM generate an average error from 18% to 39%.

Based on the analysis results of the individual examples, we have then derived general conclusions. Abstraction based on a decreasing (coarsening) granularity yields at least an order of magnitude improvement per granularity level. On the other hand, abstracting features at the same granularity level only yields marginal performance improvements. In other words, proper model granularity is the key to efficient communication abstraction.

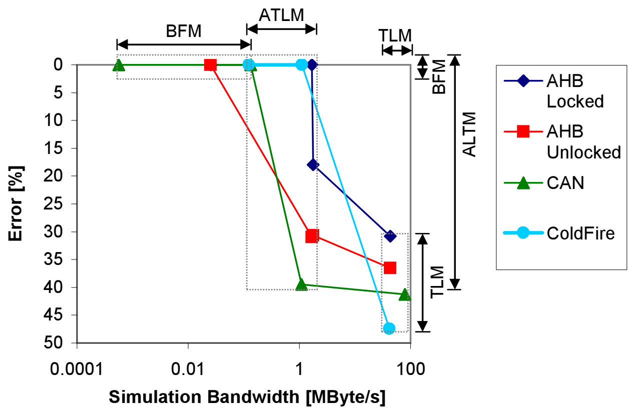


Figure 7: TLM trade-off summary.

However, TLM of communication results in a serious loss in accuracy if the modeled granularity is more coarse grain than the granularity present in the actual bus protocol. This defines the TLM trade-off. In general, a model is either fast or accurate. Our fast TLMs with up to 100 MBytes/s bandwidth show an error of up to 47%. Accurate models, on the other hand, are slow. Our BFMs simulate with less than 0.2 MByte/s bandwidth.

In summary, our systematic quantitative analysis of performance and accuracy in communication TLM, based on a diverse set of major bus architecture standards, confirms the TLM promise of high simulation speed. We can identify conditions for abstract and accurate models. As a result, we can provide general guidelines that allow the system designer to navigate the TLM trade-off effectively. Successful navigation is beneficial in two aspects, first, for the designer of abstract communication in selecting features and an abstraction method. Second, the user of communication profits from our analysis when choosing the most suitable model for the given application with fast and accurate results. More detailed information and the individual results can be found in [66, 61, 62].

2.3 Computer-Aided Re-coding

We have stated already that creating and optimizing the model of an embedded system is a critical task toward successful SoC design, that we want to improve in this work. We will now present a new technique, called *re-coding*, that reduces the time of modeling through interactive automation.

2.3.1 Coding bottleneck

One critical aspect neglected in CAD efforts so far is the design specification phase, where the intended design is captured and modeled for use in the design flow. Each design methodology expects a specific type of input model (and most methodologies also depend on intermediate models) for interaction between tools and the designer. These models need to be either hand-written from scratch, or modified from a reference model. While much of the research has focused on synthesis and refinement tools, little has been done to support the designer in forming these models.

Figure 8 shows a generic top-down system design methodology [28]. Using this design flow, we have studied several industry-size design examples, including a MP3 audio decoder [23] and a GSM voice codec [34]. For the MP3 design example, Figure 8 shows the design time spent for creating the specification model in comparison to the refinement time using automated synthesis tools. Manually re-coding the reference implementation into a proper specification model took 12-14 weeks, whereas the following implementation was completed in less than one week of time.

All these examples indicate that about 90% of the system design time is spent on coding and re-coding of the SLDL models, even in the presence of initial algorithms given in the form of C code. Moreover, we need to emphasize that specification capturing is not a one time task. When a change in the design model is required for a refinement step down in the design flow, it is often necessary to modify or *re-code* the input model.

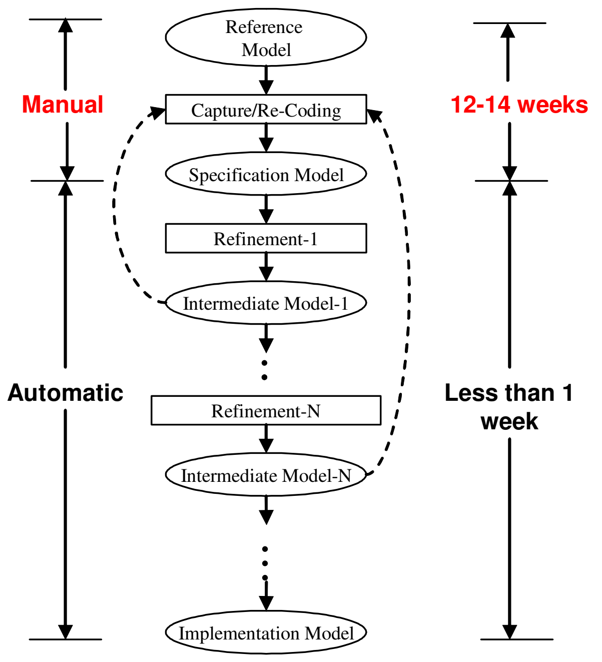


Figure 8: Design time and extent of automation in todays' refinement-based design flow.

2.3.2 Re-coding approach

Creating a well-written embedded system model involves separation of communication and computation, introducing proper granularity, and exposing concurrency. These tasks typically consist of (and are accompanied by) smaller tasks such as adding ports to a block, routing a signal through the design hierarchy, re-scoping a variable, etc. Regardless of their complexity, these operations are usually performed by the designer manually, using a plain text editor to manipulate the model described in a SLDL, such as SystemC [36], SpecC [28] or SystemVerilog [74].

Note that there is a large discrepancy between the task the designer wants to perform, and the commands a regular text editor offers (i.e. adding and deleting characters and lines, etc.). Thus, automation of such re-coding operations is desired. Textual re-coding operations, such as introducing blocks, changing the scope of variables, and grouping of functions, can be performed automatically, if the decision to apply the operation is made (and required parameters are provided) by the designer. This way, the designer is relieved from mundane text editing tasks and can focus on actual modeling decisions.

2.3.3 Interactive source re-coder

To aid the designer in coding and re-coding, we have implemented a source re-coder. Our source re-coder is a controlled, interactive approach to implement analysis and refinement tasks. In other words, it is an intelligent union of editor, compiler, and powerful transformation and analysis tools. Unlike other program transformation tools, our re-coder keeps the designer in the loop and provides complete control to generate and modify a model suitable for her/his design flow. By making the re-coding process interactive, we rely on the designer to concur, augment or overrule the analysis results of the tool, and use the combined intelligence of the re-coder and the designer for the modeling tasks.

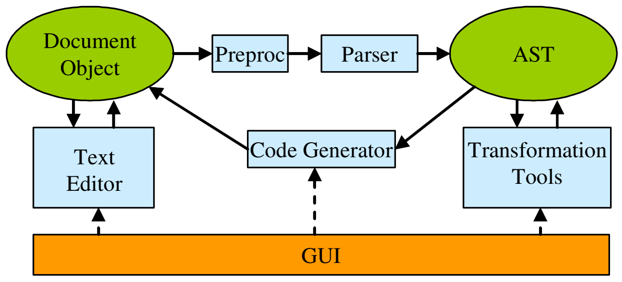


Figure 9: Conceptual Structure of the Source Re-Coder.

The conceptual structure of our source re-coder is shown in Figure 9. It consists of 5 main components, a text editor, a complex data structure (abstract syntax tree, AST), a preprocessor and parser, a code generator, and a set of analysis and transformation tools. The AST data structure [77] is needed for analyzing and transforming the design model. It is created by the preprocessor and parser when the design file is initially loaded. Subsequently, it is incrementally maintained and synchronized with the designer's actions in the text editor. The transformation and analysis tool set is the heart of our source re-coder. All re-coding tasks invoked by the user are implemented by these refinement tools. When the designer points to an object in the source window, a node corresponding to the pointed co-ordinates is located in the AST, and a list of available and possible operations is provided in a context menu. For example, when the designer points to a global variable, then the list of possible transformations includes renaming, deleting, changing the scope, and finding dependents.

2.3.4 Productivity gains

To assess the productivity gains resulting from the automated re-coding approach, we have applied it to a set of design examples, as listed in Table 1. For each example, we have used the re-coder to create SoC design models with computation and communication aspects separated. This task of exposing communication in SoC models involves three main operations, localizing variable accesses, creating explicit connectivity, and introducing synchronization channels.

Table 1: Productivity gain for different examples

Properties	JPEG	MP3	GSM
Global Variables localized	8	70	83
New Ports added	2	146	163
New Channels added	1	6	2
Re-coding time (secs)	27	246	260
Estimated Manual time (mins)	53	497	585
Productivity gain	**117x**	**121x**	**135x**

Table 1 shows the times measured using the re-coder in comparison to the times we estimated for manual editing. Table 1 also shows the productivity gains achieved for the three examples, all more than a factor 100. Using our source re-coder, the complex transformations can be realized instantly with a click of button. Thus, tedious time consuming tasks like exposing communication can be achieved in the order of seconds instead of hours.

2.4 Communication Synthesis

The fourth background topic relevant to this work is synthesis of communication architectures [33]. This work goes hand in hand with the modeling of communication discussed in Section 2.2 as both projects are based on the same well-defined abstraction levels for communication.

2.4.1 Layer-based approach

Automated system-level communication design with efficient design space exploration capabilities is becoming increasingly important. We have developed a system-level design environment for the generation of bus-based SoC communication architectures. Our approach supports a two-stage design flow with automated communication refinement [69, 70, 71] towards custom, network-oriented communication architectures beyond a traditional single shared bus. Starting from an abstract specification of the desired communication channels, our environment automatically generates tailored implementations of bus networks at various levels of abstraction. At its core, a systematic, layer-based refinement approach is utilized.

We use a layering of communication functionality within each stage of the design flow [32]. In other words, we divide the SoC communication into several layers based on separation of concerns, grouping of common functionality, dependencies across layers, and early validation of critical issues for rapid and efficient design space exploration. Our layers are stacked on top of each other where one layer provides services to the next higher layer by building upon and using the services provided by the next lower layer. In general, at its interface to higher layers, each layer provides services for establishing communication channels and for performing transactions over those channels.

2.4.2 Automatic model generation

Our communication design environment automatically generates models and implementations of system communication through refinement from an abstract description of a system architecture. Automatic refinement tools are used to produce communication models at various levels of abstraction in order to trade-off simulation accuracy and speed. Our communication design flow is targeted to support complex, non-traditional architectures with communication over a heterogeneous network of buses or other communication media.

Figure 10 shows a design model at TLM abstraction generated by our environment. The model accurately describes the communication architecture of the design down to the level of individual bus protocol transactions. As such, it abstracts away pin-related protocol details. Instead, the system components communicate via instances of TLM channels inserted from the bus database. Inside the components, protocol stacks (shown as connected half-channels) down to the media access layer have been generated and inserted.

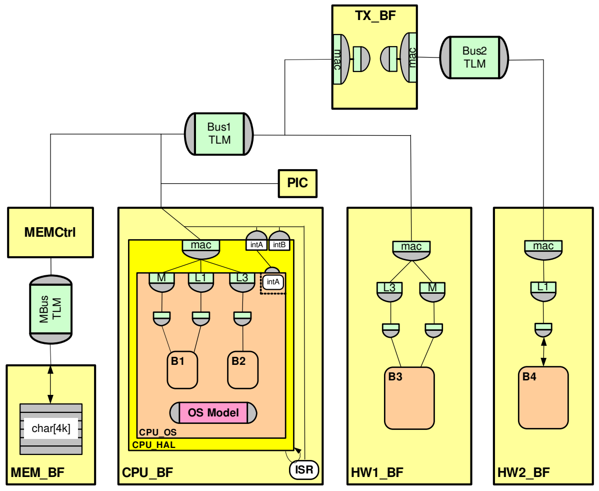


Figure 10: Automatically generated TLM design.

We have demonstrated the feasibility and benefits of our communication synthesis approach using several industrial-strength examples. Our models and layers have been systematically defined such that they can be automatically generated. Automating the tedious and error-prone process of refining high-level communication abstractions into an actual implementation results in significant gains in productivity, thus enabling rapid, early exploration of the communication design space.

2.5 Other Related Work

We will now briefly review other related work relevant to this work.

2.5.1 TLM of communication

Using TLM [36] for capturing and designing communication architectures has received much attention. Cai and Gajski [17] provide an initial taxonomy of TLM. Sgroi et al. [68] address SoC communication with a Network-on-Chip approach that follows the OSI structure. [72] describes SystemC^{SV}, an extension to SystemC with three different abstraction levels and a CAN example in [14]. Coppola et al. [25] propose abstract communication modeling in the IPSIM framework. Gerstlauer et al. [32] describes abstraction based on a decreasing number of OSI [41] layers. Abstract communication is also used in Ptolemy as presented in [47] and in [38] with an extension of dynamic switching between abstraction levels. In [20] Caldari et al. describe the results of capturing the AMBA rev. 2.0 bus standard in SystemC at two levels of abstraction. A common point for all solutions is the loss in accuracy with abstraction. OCP-IP [37] provides three TLM with increasing abstraction, with only the TL-1 being cycle accurate.

2.5.2 Communication synthesis

There are several approaches dealing with automatic generation, synthesis and refinement of communication [22, 75], which however do not provide intermediate models breaking the design gap into smaller steps required for rapid, early exploration of critical design issues. Furthermore, even in more recent work [43], only specifically crafted target implementations are supported. Historically, a lot of work has focused on automating the decision making process for communication design [79, 30, 29, 55, 44] without, however, providing corresponding design models or a path to implementation. More recently, work has been done to target automatic generation [10], refinement [51, 21] or estimation [46, 84, 48] of communication, but in all cases, the approaches are usually limited to specific target architecture templates or narrow input model semantics.

2.5.3 Computation abstraction

At the very high abstraction level of application modeling, Ptolemy [15] uses a modeling environment that integrates different models of computation (such as petri nets and boolean dataflow) in a hierarchically connected graph.

Traditionally Instruction Set Simulators play an important role in software simulation [24, 16, 52, 58]. Significant research effort has been spent to improve the performance of ISS [49, 59, 85, 83, 13, 19]. The traditional approach of an ISS based co-simulation is provided by several commercial vendors, such as ARM's SoC Designer with MaxSim Technology [2], VaST Systems' [76] virtual system prototyping tools and CoWare's [26] Virtual Platform Designer. In addition, ISS based co-simulation is used in many academic projects, such as the MPARM [8] platform and in [78].

Estimation and profiling of target specific software performance, as a necessary step towards faster simulating models, has been the aim of multiple projects [18, 7, 50]. More recent research work focuses on abstraction of a CPUs in the system level design. Bouchhima et al. [12] describe an abstract CPU subsystem that allows execution of target code on top of a hardware abstraction layer that simulates the processor capabilities. Kempf et al. [42] introduce their Virtual Processing Unit for analysis of task mapping and scheduling effects using a quantitative model.

3 Transaction Level Modeling of Computation

Transaction Level Modeling (TLM) has been proposed and is widely accepted as an efficient technique for abstract modeling of *communication*. TLM enables high-speed system simulation and rapid design space exploration. We have shown in Section 2.2 that industry-standard communication protocols, carefully designed using TLM, offer performance gains in simulation speed of up to four orders of magnitude. Due to an inherent trade-off, however, the speed-up comes usually at the price of lower accuracy. Up to now, TLM as been used exclusively for communication.

In this work, we propose to apply TLM to *computation*. As we have seen in Section 2.1.1, embedded system models clearly separate computation and communication aspects to enable automatic replacement, but both are still described using the same SLDL concepts and constructs. It is therefore surprising to see the benefits of TLM applied only to one half of the model. In other words, transferring the successful abstraction techniques of TLM also to the other half of embedded system models promises the tremendous benefits again.

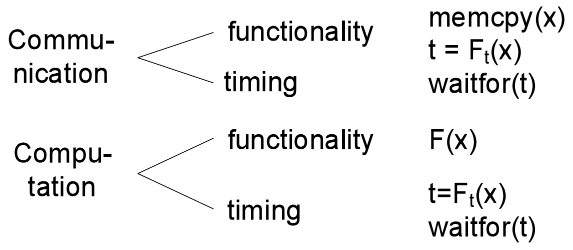


Figure 11: Two level separation of concerns.

3.1 Separation of Functionality and Timing

During our in-depth analysis of TLM for communication, we have gained the necessary insights to apply TLM also to computation. One key aspect is a second level of separation of concerns, namely the clear separation of functionality and timing. Figure 11 illustrates the two levels of separation, first communication and computation, then functionality and timing. Communication TLM is modeled as a function call containing the functionality of a data transfer, i.e. one `memcpy` operation, and the timing of this transfer in form of computation of the duration ($F_t(x)$) and a `waitfor` statement. Computation, on the other hand, is also modeled as code for functionality, i.e. a function call, and duration computation followed by a `waitfor` statement.

3.2 Granularity of Timing

Given the clear separation of functionality and timing, we need to identify the proper granularity/abstraction level. To obtain high simulation speed, the number of `waitfor` statements must be minimized because these usually lead to costly context switches in the simulator. TLM for communication therefore uses a single `memcpy` and `waitfor` statement to simulate a whole communciation transaction. Applying this to the computation counterpart means that, instead of many small $F(x)$ and `waitfor` statements (as executed by a slow Instruction Set Simulator, ISS), we need to arrage the code such that large blocks of functionality and few longer `waitfor` statements are used (as fast compiled ISS do). Figure 12 illustrates the more coarse-grained granularity of timing in TLM of computation compared to traditional instruction-set simulation. Note that this comparison is a clear indication that our approach will hold the promise of high performance gains.

3.3 Computation Abstraction

To apply TLM to computation, proper abstraction levels need to be identified, as discussed earlier in Section 1.2. Computation in embedded systems is performed by two different processing elements, namely programmable processors for software, and dedicated hardware units.

For software execution, traditional abstraction levels range from C code down to assembly code. Both extremes can be simulated in a SLDL as native compiled code (usually referred to as untimed specification) or using an ISS, respectively. Between these, we find the abstraction levels of host-compiled ISS and computation TLM, the latter one to be defined in detail in this work. Figure 13 illustrates these four abstraction levels for modeling software execution on programmable processors. For reference, Figure 14 shows the software compilation flows used to generate the resulting simulation executables by the four different approaches.

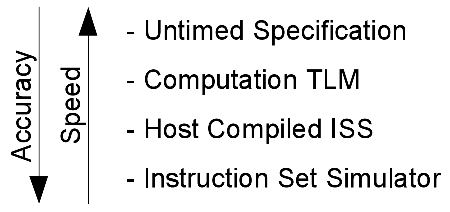


Figure 13: Abstraction levels of computation using programmable processors.

For dedicated hardware, potential abstraction levels range from behavioral simulation (i.e. C code) down to the cycle-accurate Register Transfer Level (RTL). Potential candidates for TLM abstraction are likely similar to the abstraction level at the software side, but also include the five different levels defined by the Accellera semantics [1].

3.4 Result-Oriented Modeling

Towards TLM of computation, we will now describe a novel technique called Result Oriented Modeling

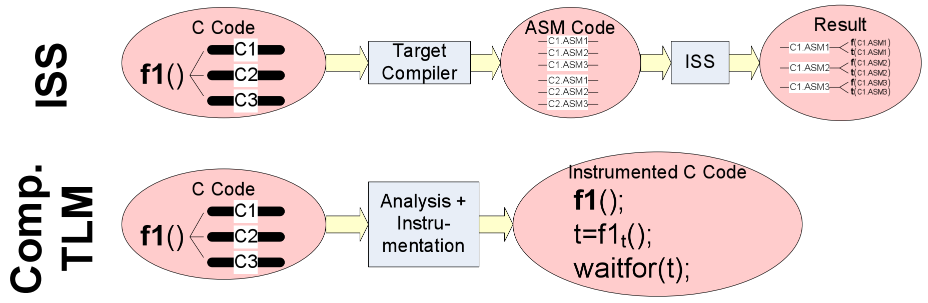
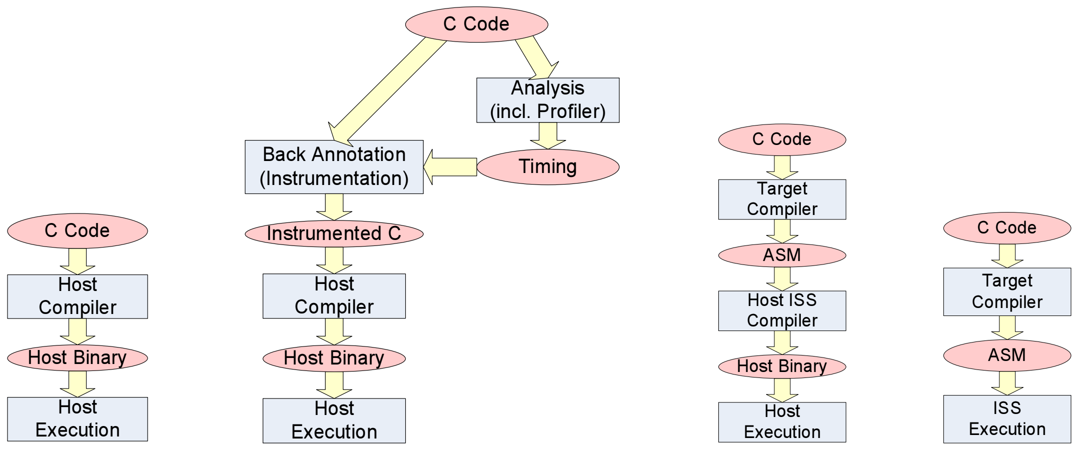


Figure 12: Granularity of timing in TLM of computation vs. traditional ISS.



(a) Untimed Spec (b) Computation TLM (c) Host Compiled ISS (d) ISS

Figure 14: Flows for computation simulation.

(ROM) [63, 64] that delivers the same high performance as regular TLM, but retains 100% accuracy in timing. In order to reach these apparently conflicting accuracy goals, ROM relies on the the basic assumption that the effects of the functionality are only observable at the boundaries of a user transaction. Therefore ROM can eliminate internal state changes, rearrange events and avoid costly context switches.

3.4.1 ROM approach

Result Oriented Modeling (ROM) is a general concept for abstract yet accurate modeling of a process. As such, ROM is similar to the "black box" concept. The underlying assumption of ROM is the limited observability of internal state changes of the modeled process. It is not necessary to show intermediate results of the process to the user, as in a "black box" approach. The only goal of ROM is to produce the *end result* of the process fast. 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.

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 performance, if the cost for any corrective measures is low. This approach is in contrast to the traditional 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 time and makes any necessary adjustment at the end.

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.

In contrast to ROM, traditional TLM achieves speedup by modeling the internal state changes at a more coarse granularity and reaches the final result through an incremental approach taking into account any disturbing influence at each step. ROM on the other hand, optimistically predicts the end result, records the disturbing influence and only makes a correction in the end when it deems necessary.

3.4.2 Airplane arrival analogy

Figure 15 illustrates the ROM approach using the example of predicting the arrival time of an airplane. The real process (a) exhibits continuous changes to the airspeed dependent on the disturbing influence wind. The traditional abstract modeling approach (b) approximates the result by incrementally calculating the air speed in dependence of the wind in (coarse-grain) discrete time steps. The ROM approach (c), on the other hand, does not model the intermediate airplane speed. Instead, it makes one initial optimistic prediction about the arrival time, and finally corrects its prediction retroactively for the average wind condition.

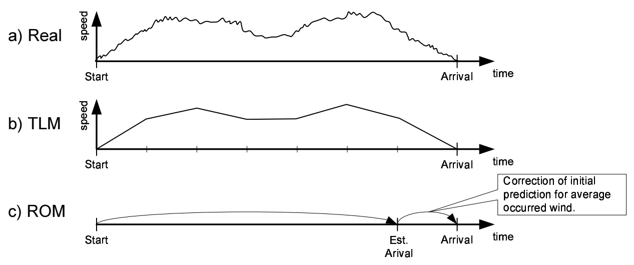


Figure 15: ROM predicting an airplane arrival time.

3.4.3 ROM of computation

To apply our ROM concept to modeling of computation, we take advantage of the separation of functionality and timing as described in Section 3.1. For fastest functionality, we will use direct compilation of the code, similar to the host compiled approach of ISS. For the timing, will use the optimistic prediction of ROM.

Traditionally, the target timing is achieved by using discrete wait statements with a fixed wait time. Applying the ROM concept, we can optimistically calculate the wait time during the simulation, taking into account the virtual processor state. Interrupts, pipeline and cache effects can be modeled as disturbing influences. The optimistic prediction of the execution time will cover a block of code until the next observable activity on the outside. At the end of the predicted time interval, the initial calculation is verified. In case of no disturbing influence, the initial prediction will proof correct, the observable activity can be committed, and the simulation proceeds. In case of a disturbing influence, the initially predicted time will be too short, but can be updated using the now available status information. The simulation will then wait for the extended time, after which the process of prediction updates is repeated.

In summary, we are confident that the ROM idea is applicable to TLM of computation, and we expect highest simulation speeds with high accuracy.

3.4.4 Initial Experiments

To estimate the benefits of our proposed modeling concept, we have applied the generic ROM approach to a bus system [64], the AMBA AHB [3].

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.

When a bus 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 only a single wait statement is used.

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 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.

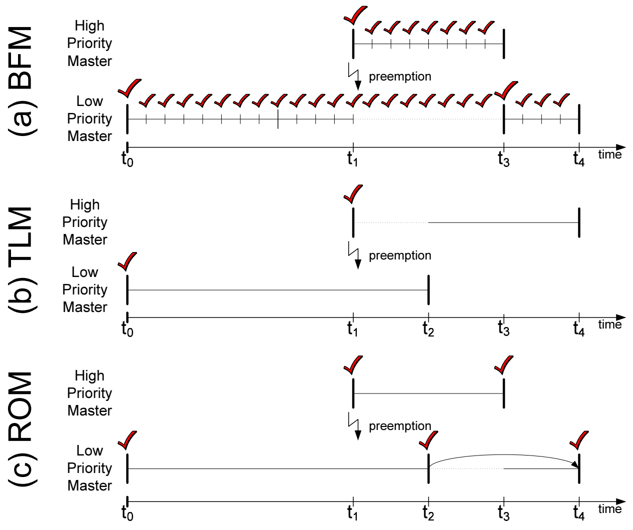


Figure 16: Preemption in BFM, TLM, ROM

To compare ROM against other modeling approaches, we will analyze the case of bus preemption in more detail, as shown in Figure 16. In (a) BFM, a burst transaction starting at t_0 is preempted at t_1. The higher priority transfer completes at t_3 when the preempted transfer resumes, terminating finally at t_4. Both masters perform arbitration checks for every bus cycle, a total of 32 in this example.

In (b) traditional TLM, 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 two arbitration checks are performed.

In (c) ROM, the inaccuracies of the TLM are corrected by 3 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.

3.4.5 Initial Results

Our initial results for the ROM are extremely promising. ROM executes as fast as the most abstract traditional TLM, however it is 100% accurate. Thus it completely eliminates the trade-off as it exists for traditional TLM.

We have examined the performance with prediction updates using two masters and two slaves. The high priority master puts an equally distributed base load of 33% on the bus by sending 8-beat burst transactions. The low priority master issues transactions of increasing size without a delay in between, as before[2].

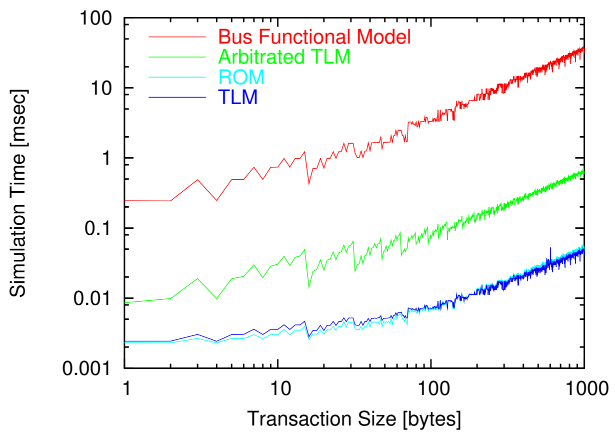


Figure 17: ROM performance comparison.

Figure 17 shows the time to simulate the low priority master over an increasing message size while the

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

high priority master is running at the same time. It reveals that both ROM and TLM are equally fast, three orders of magnitude faster than the BFM, and one order of magnitude faster than the ATLM.

Knowing that ROM is equally fast as TLM, we need to validate whether ROM meets the target of 100% timing accuracy. We use the same setup and metrics as in Section 2.2.2. Our analysis confirmes that ROM for the AMBA AHB is 100% accurate regardless of the bus-contention. In that respect it is identical to the BFM. We omit the actual measurement graph for space reasons, since it is identical to Figure 6, with the exception of a second fully accurate model on top of the x-axis, ROM.

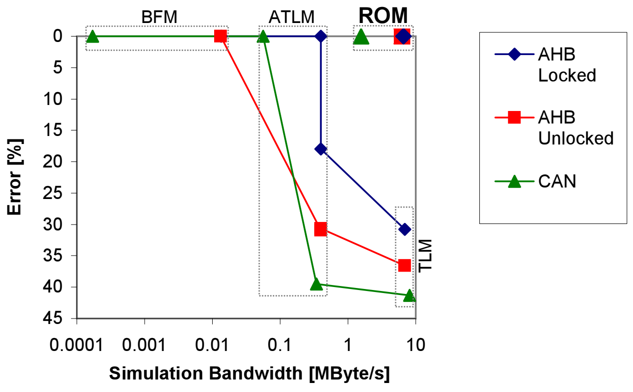


Figure 18: ROM beats the TLM Trade-Off.

We have also implemented the ROM approach for the Controller Area Network (CAN). The results of this second example are similarly exciting. Figure 18 combines our results for both the AMBA AHB and the CAN modeling. It shows on the x-axis the bandwidth of the low priority master in a two master setup with 50% utilization of the bus. The y-axis denotes the average error in transfer duration for the low priority master.

Our experimental results demonstrate the tremendous benefits of ROM. While the traditional TLM suffers from a significant speed/accuracy trade-off, as Figure 18 shows for the models in the ATLM and the TLM category, only ROM delivers highest speed *and* 100% accuracy at the same time, lying in the hot-spot top right area of the trade-off graph.

These exciting initial results strongly support the proper direction of our research, allowing high expectations for ROM applied to computation. Our goal is to reach 100% accurate communication *and* computation at highest speeds. Achieving that goal, or even only approaching it, will significantly increase our abilities in designing embedded systems more efficiently. This will allow us to explore a larger design space more rapidly, leading to ”better” embedded systems in the end.

4 Current Status and Future Work

Our work on TLM for computation, as outlined in the previous sections, is at this point still in it’s infancy. In fact, this work is still in the “idea” state. The main idea of applying the concepts of TLM to computation is clear, and we have shown some very promising initial results, but most work still lies ahead. Much more research is necessary in order to really understand the concept, to put it to work for real-world applications, to back it up with actual and accurate experimental results, and, last but not least, to deploy it such that it benefits the actual design process of embedded systems.

Looking back (see Section 2), we have based this research on a solid foundation, namely the work on SpecC featuring the separation of concerns (see Section 2.1), the systematic analysis of TLM of communication (see Section 2.2), the automation of modeling through re-coding (see Section 2.3), and finally the work on communication synthesis (see Section 2.4).

Looking forward, much research and development work needs to be performed to bring this idea to real fruition. The following is a rough attempt at listing the tasks ahead.

- We need to start with the identification of model features suitable for abstraction, and the definition of promising abstraction levels. Results of these tasks can then be used to define proper modeling techniques.

- Next, actual example models will be required. Modeling, describing and validating these models using a SLDL will require a significant amount of time, based on past experience [23, 35, 80]. As soon as models become available, suitable metrics and measurements need to be defined such that inherent trade-offs can be analyzed in detail (compare Section 2.2.2).

- Then, the concepts developed before can be applied toward synthesis. Well-designed models are expected to be easier synthesizable and will lead to better implementations.
- The next step can then focus on defining, building, and testing a suitable synthesis flow.
- Finally, to demonstrate the success of the proposed approach, we need to finally develop and build a prototype system of a real-world application such as an MP3 player or video codec.

Eventually, in order to generate an impact in the real world, technology transfer to industry is essential. This should begin as soon as successful results become available and show sufficient maturity.

5 Summary and Conclusion

In this work, we have introduced and outlined our research on transaction level modeling (TLM) of computation. We have shown that TLM for communication enables simulation speeds of up to four orders of magnitude faster than pin-accurate bus functional models. Usually, however, this performance gain comes at the price of low accuracy.

This work now applies the successful techniques of TLM to the computation aspects in embedded systems. This has not been done before. Moreover, the exciting advantages experienced with TLM for communication strongly indicate a similar impact when applied for computation.

The initial results of TLM for computation outlined in this document support this high expectation. Our analysis has shown that the use of a novel modeling approach, called result-oriented modeling (ROM), enables the same high speed simulation as traditional TLM, yet achieves 100% accuracy in timing. This eliminates the speed/accuracy trade-off, allowing to utilize both advantages in a model at the same time.

While traditional work largely has focused on analysis, refinement and synthesis tasks starting from a given system model, this project addresses the creation and optimization of input models for effective use in existing design processes, including accurate estimation, rapid design space exploration, and efficient synthesis and implementation. The results of this work are directly applicable to established design flows in the industry.

Just like the quality of an architectural blue-print determines the quality of the resulting building, the model of an embedded system is the key to its successful implementation. Thus, the better modeling techniques proposed in this work will directly improve the technical systems around us and the quality of life for everyone in our society.

Acknowledgement

I would like to thank Pramod Chandraiah and Gunar Schirner for helping to prepare this document.

References

[1] Accellera, C/C++ Class Library Standardization Working Group. *RTL Semantics*, February 2001. Draft Specification, Version 0.8.

[2] Advanced RISC Machines Ltd. (ARM). SoC Developer with MaxSim Technology. http://www.arm.com/products/DevTools/MaxSim.html.

[3] ARM. *AMBA Specification (Rev. 2.0), ARM IHI 0011A*. Advanced RISC Machines Ltd. (ARM), 1999.

[4] I. Bacivarov, S. Yoo, and A. Jerraya. Timed HW-SW Cosimulation Using Native Execution of OS and Application SW. In *International High-Level Design Validation and Test workshop*, Cannes, France, October 2002.

[5] I. Bacivarov, S. Yoo, and A. Jerraya. Fast and Accurate Timed Execution of High Level Embedded Software using HW/SW Interface Simulation Model. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, Yokohama, Japan, January 2004.

[6] F. Balarin, H. Hsieh, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, and Y. Watanabe.

Metropolis: An Integrated Environment for Electronic System Design. *IEEE Computer*, 36(4), April 2003.

[7] J. R. Bammi, W. Kruijtzer, L. Lavagno, E. Harcourt, and M. T. Lazarescu. Software performance estimation strategies in a system-level design tool. In *Proceedings of International Workshop on Hardware/Software Codesign (CODES)*, San Diego, CA, 2000.

[8] L. Benini, D. Bertozzi, A. Bogoliolo, F. Menichelli, and M. Olivieri. MPARM: Exploring the Multi-Processor SoC Design Space with SystemC. *Journal of VLSI Signal Processing*, 41(2):169–184, 2005.

[9] L. Benini and G. D. Micheli. Networks on chips: A new SoC paradigm. *IEEE Computer*, January 2002.

[10] I. Bolsens, H. D. Man, B. Lin, K. V. Rompay, S. Vercauteren, and D. Verkest. Hardware/Software co-design of the digital telecommunication systems. *Proceedings of the IEEE*, March 1997.

[11] Bosch. *CAN Specification*. Robert Bosch GmbH, 2.0 edition, 1991. http://www.can.bosch.com/.

[12] A. Bouchhima, I. Bacivarov, W. Yousseff, M. Bonaciu, and A. Jerraya. Using Abstract CPU Subsystem Simulation Model for High Level HW/SW Architecture Exploration. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, Shanghai, China, January 2005.

[13] G. Braun, A. Nohl, A. Hoffmann, O. Schliebusch, R. Leupers, and H. Meyr. A universal technique for fast and flexible instruction-set architecture simulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(12):1625–1639, 2004.

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

[15] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. Ptolemy: A framework for simulating and prototyping heterogeneous systems. *International Journal of Computer Simulation*, 4(2):155–182, April 1994.

[16] D. Burger and T. M. Austin. The simplescalar tool set, version 2.0. *ACM SIGARCH Computer Architecture News*, 25(3):13–25, 1997.

[17] L. Cai and D. Gajski. Transaction Level Modeling: An Overview. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, Newport Beach, CA, October 2003.

[18] L. Cai, A. Gerstlauer, and D. D. Gajski. Retargetable profiling for rapid, early system-level design space exploration. In *Proceedings of the Design Automation Conference (DAC)*, San Diego, CA, June 2004.

[19] H. W. Cain, K. M. Lepak, and M. H. Lipasti. A dynamic binary translation approach to architectural simulation. *ACM SIGARCH Computer Architecture News*, 29(1):27–36, 2001.

[20] 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 *Proceedings of the Design, Automation and Test in Europe (DATE) Conference*, Munich, Germany, March 2003.

[21] W. O. Cesario, A. Baghdadi, L. Gauthier, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A. A. Jerraya, and M. Diaz-Nava. Component-baed design approach for multicore SoCs. In *Proceedings of the Design Automation Conference (DAC)*, pages 789–794, June 2002.

[22] W. O. Cesário, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A. A. Jerraya, L. Gauthier, and M. Diaz-Nava. Multiprocessor SoC platforms: A component-based design approach. *IEEE Design and Test of Computers*, 19(6), November/December 2002.

[23] P. Chandraiah and R. Dömer. Specification and design of an MP3 audio decoder. Technical Report CECS-TR-05-04, Center for Embedded Computer Systems, University of California, Irvine, May 2005.

[24] B. Cmelik and D. Keppel. Shade: a fast instruction-set simulator for execution profiling. In *In Proceedings of the ACM SIGMETRICS conference on Measurement and modeling of computer systems*, Nashville, Tennessee, United States, 1994.

[25] M. Coppola, S. Curaba, M. Grammatikakis, and G. Maruccia. IPSIM: SystemC 3.0 enhancements for communication refinement. In *Proceedings of the Design, Automation and Test in Europe (DATE) Conference*, Munich, Germany, March 2003.

[26] CoWare. Virtual Platform Designer. www.coware.com.

[27] R. Dömer, A. Gerstlauer, and D. Gajski. *SpecC Language Reference Manual, Version 2.0.* SpecC Technology Open Consortium, http://www.specc.org, December 2002.

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

[29] M. Gasteier and M. Glesner. Bus-Based Communication Synthesis on System-Level. In *Proceedings of the International Symposium on System Synthesis*, San Diego, CA, USA, November 1996.

[30] M. Gasteier, M. Münch, and M. Glesner. Generation of interconnect topologies for communication synthesis. In *Proceedings of the Design, Automation and Test in Europe (DATE) Conference*, pages 36–42, March 1998.

[31] A. Gerstlauer, R. Dömer, J. Peng, and D. D. Gajski. *System Design: A Practical Guide with SpecC*. Kluwer Academic Publishers, 2001.

[32] A. Gerstlauer, D. Shin, R. Doemer, and D. Gajski. System-Level Communication Modeling for Network-on-Chip Synthesis. In *Asia and South Pacific Design Automation Conference*, Shanghai, China, January 2005.

[33] A. Gerstlauer, D. Shin, J. Peng, R. Dömer, and D. Gajski. Systematic, Layer-based Generation of System-on-Chip Bus Networks. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, submitted, May 2006.

[34] A. Gerstlauer, S. Zhao, D. D. Gajski, and A. M. Horak. Design of a GSM vocoder using SpecC methodology. Technical Report ICS-TR-99-11, Information and Computer Science, University of California, Irvine, March 1999.

[35] A. Gerstlauer, S. Zhao, D. D. Gajski, and A. M. Horak. SpecC system-level design methodology applied to the design of a GSM vocoder. In *Proceedings of the Workshop of Synthesis and System Integration of Mixed Information Technologies*, Kyoto, Japan, April 2000.

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

[37] A. Haverinen, M. Leclercq, N. Weyrich, and D. Wingard. SystemC based SoC Communication Modeling for the OCP Protocol, October 2002. http://www.ocpip.org.

[38] K. Hines and G. Borriello. Optimizing Communication in Embedded System Co-simulation. In *Proceedings of Codes/CACHE*, Braunschweig, Germany, March 1997.

[39] IEEE. *IEEE Standard VHDL Language Reference Manual, IEEE Std. 1076-1993*. IEEE, revision 1993 edition, 1993.

[40] IEEE. *Hardware Description Language Based on the Verilog Hardware Description Language, IEEE Std. 1364-1996*. IEEE, 1996.

[41] ISO. *Reference Model of Open System Interconnection (OSI)*. Internation Organization

for Standardization (ISO), second edition, 1994. ISO/IEC 7498 Standard.

[42] T. Kempf, M. Dörper, R. Leupers, G. Ascheid, H. Meyr, T. Kogel, and B. Vanthournout. A Modular Simulation Framework for Spatial and Temporal Task Mapping onto Multi-Processor SoC Platforms. In *Proceedings of the Design, Automation and Test in Europe (DATE) Conference*, Munich, Germany, March 2005.

[43] W. Klingauf, H. Gädke, and R. Günzel. TRAIN: A virtual transaction layer architecture for TLM-based HW/SW codesign of synthesizable MPSoC. In *Proceedings of the Design, Automation and Test in Europe (DATE) Conference*, March 2006.

[44] K. Lahiri, A. Raghunathan, and S. Dey. Efficient exploration of the SoC communication architecture design space. In *Proceedings of the International Conference on Computer Aided Design (ICCAD)*, pages 424–430, November 2000.

[45] K. Lahiri, A. Raghunathan, and S. Dey. System-level performance analysis for designing on-chip communication architectures. *IEEE Transactions on Computer-Aided Design of Intergrated Circuits and Systems (TCAD)*, 20(6):768–783, June 2001.

[46] K. Lahiri, A. Raghunathan, and S. Dey. System-Level Performance Analysis for Designing On-Chip Communication Architectures. In *IEEE Transactions on Computer-Aided Design of Intergrated Circuits and Systems (TCAD)*, volume 20, pages 768–783, June 2001.

[47] M. Lajolo, C. Passerone, and L. Lavagno. Scalable Techniques for System-level Co-Simulation and Co-Estimation. *IEE Proceedings - Computers and Digital Techniques*, 150(4):227–238, July 2003.

[48] M. Lajolo, C. Passerone, and L. Lavagno. Scalable Techniques for System-level Co-Simulation and Co-Estimation. In *IEE Proceedings–Computers and Digital Techniques*, volume 150, pages 227–238, July 2003.

[49] R. Leupers, J. Elste, and B. Landwehr. Generation of interpretive and compiled instruction set simulators. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan. 1999.

[50] Y.-T. S. Li, S. Malik, and A. Wolfe. Performance estimation of embedded software with instruction cache modeling. *ACM Transactions on Design Automation of Electronic Systems*, 4(3):257–279, 1999.

[51] D. Lyonnard, S. Yoo, A. Baghdadi, and A. A. Jerraya. Automatic generation of application-specific architectures for heterogeneous multiprocessor system-on-chip. In *Proceedings of the Design Automation Conference (DAC)*, Las Vegas, NV, June 2001.

[52] N. Manjikian. Multiprocessor enhancements of the simplescalar tool set. *ACM SIGARCH Computer Architecture News*, 29(1):8–15, 2001.

[53] Motorola. *MCF5206 ColdFire Integrated Microprocessor User's Manual*, 1997.

[54] Open SystemC Initiative, http://www.systemc.org. *Functional Specification for SystemC 2.0*, 2000.

[55] R. B. Ortega and G. Borriello. Communication synthesis for distributed embedded systems. In *Proceedings of the International Conference on Computer Aided Design (ICCAD)*, pages 437–444, November 1998.

[56] Open SystemC Initiative. http://www.systemc.org.

[57] 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.

[58] W. Qin and S. Malik. Flexible and formal modeling of microprocessors with application to retargetable simulation. In *Proceedings of the Design, Automation and Test in Europe (DATE) Conference*, Munich, Germany, March 2003.

[59] M. Reshadi, P. Mishra, and N. Dutt. Instruction set compiled simulation: a technique for fast and flexible instruction set simulation. In *Proceedings of the Design Automation Conference (DAC)*, Anaheim, USA, June 2003.

[60] A. Sarmento, W. Cesario, and A. A. Jerraya. Mixed-Level Cosimulation for Fine Gradual Refinement of Communication in SoC Design. In *Proceedings of the Design, Automation and Test in Europe (DATE) Conference*, Munich, Germany, March 2001.

[61] 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.

[62] G. Schirner and R. Dömer. System Level Modeling of an AMBA Bus. Technical Report CECS-TR-05-03, Center for Embedded Computer Systems, University of California, Irvine, March 2005.

[63] G. Schirner and R. Dömer. Accurate yet fast modeling of real-time communication. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, Seoul, Korea, October 2006.

[64] G. Schirner and R. Dömer. Fast and Accurate Transaction Level Models using Result Oriented Modeling. In *Proceedings of the International Conference on Computer Aided Design (ICCAD)*, San Jose, California, November 2006.

[65] G. Schirner and R. Dömer. Quantitative Analysis of the Speed/Accuracy Trade-off in Transaction Level Modeling. *ACM Transactions on Embedded Computing Systems*, submitted, June 2006.

[66] G. Schirner and R. Dömer. Quantitative Analysis of Transaction Level Models for the AMBA Bus. In *Proceedings of the Design, Automation and Test in Europe (DATE) Conference*, Munich, Germany, March 2006.

[67] SEMATECH Inc. International technology roadmap for semiconductors (ITRS), 2004 update, design. http://www.itrs.net/, 2004.

[68] 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 *Proceedings of the Design Automation Conference*, Las Vegas, NV, USA, June 2001.

[69] D. Shin, A. Gerstlauer, R. Doemer, and D. D. Gajski. Automatic generation of communication architectures. In A. Rettberg, M. C. Zanella, and F. J. Rammig, editors, *From Specification to Embedded Systems Application*. Springer, August 2005.

[70] D. Shin, A. Gerstlauer, R. Dömer, and D. D. Gajski. Automatic network generation for system-on-chip communication design. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, September 2005.

[71] D. Shin, A. Gerstlauer, J. Peng, R. Dömer, and D. D. Gajski. Automatic generation of transaction-level models for rapid design space exploration. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, Seoul, Korea, October 2006.

[72] R. Siegmund and D. Müller. SystemC^{SV}: An extension of SystemC for mixed multi-level communication modeling and interface-based system design. In *Proceedings of the Design, Automation and Test in Europe (DATE) Conference*, Munich, Germany, March 2001.

[73] SpecC Technology Open Consortium. http://www.specc.org.

[74] S. Sutherland, S. Davidmann, P. Flake, and P. Moorby. *System Verilog for Design: A Guide to Using System Verilog for Hardware Design and Modeling*. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[75] K. van Rompaey, D. V. I. Bolsens, and H. D. Man. CoWare: A design environment for heterogeneous hardware/software systems. In *Proceedings of the European Design Automation Conference (Euro-DAC)*, Geneva, Switzerland, September 1996.

[76] VaST Systems. VaST tools and models for embedded system design. www.vastsystems.com.

[77] I. Viskic and R. Dömer. A flexible, syntax independent representation (SIR) for system level design models. In *Proceedings of EuroMicro Conference on Digital System Design*, Dubrovnik, Croatia, August 2006.

[78] A. Wieferink, M. Doerper, T. Kogel, R. Leupers, G. Ascheid, and H. Meyr. Early iss integration into network-on-chip designs. In *In Proceedings of International Workshop on Systems, Architectures, Modeling and Simulation (SAMOS)*, Samos, Greece, July 2004.

[79] T.-Y. Yen and W. Wolf. Communication synthesis for distributed embedded systems. In *Proceedings of the International Conference on Computer Aided Design (ICCAD)*, San Jose, CA, November 1995.

[80] H. Yin, H. Du, T.-C. Lee, and D. D. Gajski. Design of a JPEG encoder using SpecC methodology. Technical Report ICS-TR-00-23, Information and Computer Science, University of California, Irvine, July 2000.

[81] S. Yoo, G. Nicolescu, L. Gauthier, and A. Jerraya. Automatic Generation of Fast Timed Simulation Models for Operating Systems in SoC Design. In *Proceedings of the Design, Automation and Test in Europe (DATE) Conference*, Paris, France, March 2002.

[82] J. Zhu, R. Dömer, and D. D. Gajski. Syntax and semantics of the SpecC language. In *Proceedings of the International Symposium on System Synthesis*, Osaka, Japan, December 1997.

[83] J. Zhu and D. D. Gajski. A retargatable, ultra-fast instruction set simulator. In *Proceedings of the Design, Automation and Test in Europe (DATE) Conference*, Munich, Germany, March 1999.

[84] X. Zhu and S. Malik. A hierarchical modeling framework for on-chip communication architectures. In *Proceedings of the International Conference on Computer Aided Design (ICCAD)*, pages 663–670, November 2002.

[85] V. Zivojnvic, S. Tjiang, and H. Meyr. Compiled simulation of programmable dsp architectures. In *In Proceedings of the IEEE Workshop on VLSI Signal Processing*, Sakai, Japan, 1995.