

UNIVERSITY OF CALIFORNIA,
IRVINE

**Analysis and Optimization of Transaction Level Models for
Multi-Processor System-on-Chip Design**

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Electrical and Computer Engineering

by

Hans Gunar Schirner

Dissertation Committee:
Professor Rainer Dömer, Chair
Professor Daniel D. Gajski
Professor Pai Chou
Andreas Gerstlauer

2008

The dissertation of Hans Gunar Schirner
is approved and is acceptable in quality and form for
publication on microfilm and in digital formats:

Committee Chair

University of California, Irvine
2008

To my family.

Contents

List of Figures	viii
List of Tables	x
List of Acronyms	xi
Acknowledgments	xv
Curriculum Vitae	xvi
Abstract of the Dissertation	xxi
1 Introduction	1
1.1 System-Level Design	3
1.1.1 Methodology	5
1.1.2 System-Level Design Languages	7
1.2 Abstract Models	9
1.2.1 Abstraction of Communication	9
1.2.2 Abstraction of Computation	13
1.2.3 Basic Models in System-level Design	16
1.2.4 TLM Trade-off	18
1.3 Dissertation Goals	20
1.4 Dissertation Overview	21
1.5 Related Work	21
1.5.1 Languages for System-Level Design	21
1.5.2 Abstraction and Analysis of Communication	22
1.5.3 Abstraction and Analysis of Computation	25
2 Transaction Level Modeling	28
2.1 Introduction	28
2.1.1 TLM Trade-off	28
2.1.2 Problem Statement	29

2.1.3	Overview	30
2.2	Transaction Level Modeling	30
2.2.1	Transaction Level Model (TLM)	32
2.2.2	Arbitrated Transaction Level Model (ATLM)	32
2.2.3	Bus Functional Model (BFM)	32
2.2.4	Comparison with other TLM Abstractions	33
2.3	Metrics and Measurement Setup	33
2.3.1	Performance	33
2.3.2	Accuracy	34
2.4	AMBA	38
2.4.1	Introduction	38
2.4.2	Models	39
2.4.3	Performance Analysis	40
2.4.4	Accuracy Analysis	42
2.4.5	Summary for the AMBA AHB	46
2.5	CAN	47
2.5.1	Introduction	48
2.5.2	Models	49
2.5.3	Performance Analysis	50
2.5.4	Accuracy Analysis	51
2.5.5	Summary for the CAN	53
2.6	ColdFire Master Bus	56
2.6.1	Introduction	56
2.6.2	Models	57
2.6.3	Performance Analysis	57
2.6.4	Accuracy Analysis	58
2.6.5	Summary for the ColdFire Master Bus	59
2.7	Generalization	60
2.7.1	Performance	60
2.7.2	Accuracy	61
2.7.3	TLM Trade-Off	62
2.8	Summary Transaction Level Modeling	63
3	Result Oriented Modeling (ROM)	65
3.1	Introduction	65
3.1.1	Scope of Work	66
3.1.2	Overview	67
3.2	Result Oriented Modeling	67
3.2.1	Black Box Concept	67
3.2.2	Corrective Measures	67
3.2.3	Example	68
3.2.4	Analogy	69

3.3	Communication Modeling using ROM	70
3.3.1	AMBA AHB - Traditional Modeling	70
3.3.2	AMBA AHB - Result Oriented Modeling	73
3.3.3	CAN - Traditional Modeling	77
3.3.4	CAN - Result Oriented Modeling	78
3.4	ROM Algorithm	80
3.5	Experimental Results	82
3.5.1	Accuracy	83
3.5.2	Performance	85
3.5.3	Prediction Updates	89
3.6	Generalization	91
3.6.1	Escaping the TLM Trade-Off	91
3.6.2	Complexity Considerations	92
3.6.3	Limitations	92
3.7	Summary Result Oriented Modeling	93
4	Abstract Processor Modeling	95
4.1	Introduction	95
4.1.1	Problem Definition	96
4.1.2	Overview	98
4.2	Context: Our MPSoC Development Approach	98
4.3	Abstract Processor Modeling	100
4.3.1	Application	101
4.3.2	Task Scheduling (OS Kernel)	102
4.3.3	Firmware (External Communication)	104
4.3.4	Processor Transaction Level Model	105
4.3.5	Processor Bus Functional Model (BFM)	107
4.3.6	ISS-based Cosimulation Model	108
4.3.7	Model Benefits	109
4.4	Experimental Results	110
4.4.1	Setup	111
4.4.2	Performance Analysis	114
4.4.3	Accuracy Analysis	117
4.4.4	Trade-off for System Simulation	118
4.5	Summary Abstract Processor Modeling	120
5	Summary and Conclusions	122
5.1	Summary	122
5.1.1	TLM Analysis	123
5.1.2	Optimized Abstract Modeling Technique	124
5.1.3	Abstract Processor Modeling	124
5.2	Contributions	125

5.2.1	Analysis of Transaction Level Models for Communication	125
5.2.2	Optimized Abstract Modeling Technique	126
5.2.3	Abstract Processor Modeling	126
5.3	Closing Remarks	126
Bibliography		128

List of Figures

1.1	Productivity gap.	3
1.2	Abstraction levels in SoC design.	5
1.3	Software execution stack.	15
1.4	Abstraction layers of communication.	17
1.5	Transaction Level Modeling Trade-Off.	19
2.1	Transaction Level Modeling Trade-Off.	29
2.2	Model classes and their granularity.	31
2.3	Single master setup for performance measurements.	34
2.4	Cumulative and individual transfer time.	35
2.5	Bus contention.	37
2.6	Dual master setup for accuracy measurements.	37
2.7	AMBA bus architecture.	39
2.8	AMBA AHB operation modes.	39
2.9	Performance for the AMBA AHB models.	41
2.10	Individual timing accuracy of locked transfers for the AMBA AHB models.	42
2.11	Cumulative timing accuracy of locked transfers for the AMBA AHB.	43
2.12	Cumulative timing accuracy for unlocked transfers for the AMBA AHB.	44
2.13	Histogram of normalized transaction duration.	45
2.14	AMBA AHB TLM trade-off.	46
2.15	CAN data frame.	48
2.16	Performance of the CAN models.	50
2.17	Individual timing accuracy for the CAN models.	52
2.18	Cumulative timing accuracy for the CAN models.	54
2.19	CAN TLM trade-off.	55
2.20	ColdFire Master Bus with two masters	56
2.21	Performance of the ColdFire Master bus models.	57
2.22	Individual timing accuracy for the ColdFire Master bus models.	58
2.23	ColdFire Master bus TLM trade-off.	59
2.24	TLM trade-off summary.	63

3.1	Transaction Level Modeling Trade-Off.	66
3.2	Generic ROM concept.	68
3.3	ROM predicting an airplane arrival time.	69
3.4	Layer-based Bus Modeling.	71
3.5	Arbitration check points when transferring two 8-beat bursts.	72
3.6	Arbitration check points in ROM.	73
3.7	Preemption in BFM, TLM, ROM.	75
3.8	Contention in ATLM, TLM and ROM	79
3.9	Multi-node setup.	83
3.10	Accuracy of the AMBA AHB models.	84
3.11	Accuracy of the CAN models.	84
3.12	Transfer time using AMBA models.	86
3.13	Transfer time using CAN models.	87
3.14	Exponentially decreasing number of prediction updates.	89
3.15	Histogram of number of prediction updates.	90
3.16	ROM beats the TLM Trade-Off.	92
4.1	Trade-off in system simulation.	96
4.2	Generic MPSoC target architecture.	97
4.3	Software development framework.	99
4.4	Application model and external communication.	101
4.5	Timing back annotation.	102
4.6	Task model.	103
4.7	Abstract scheduler switching between tasks.	103
4.8	Firmware model.	104
4.9	Example of inserted driver code for synchronization.	105
4.10	Processor Transaction Level Model.	106
4.11	Hardware interrupt scheduling.	107
4.12	Processor Bus Functional Model.	108
4.13	Bus trace in BFM.	108
4.14	Bus Functional Model with ISS.	109
4.15	Example cellphone architecture.	111
4.16	Simulation time for SW-only systems.	115
4.17	Simulation time for HW/SW Systems.	116
4.18	Accuracy of HW/SW systems.	118
4.19	System performance and accuracy.	119

List of Tables

1.1	Communication layers.	11
2.1	Performance comparison of AMBA AHB models.	41
2.2	AMBA AHB model selection.	47
2.3	Summary of features captured in the CAN models.	50
2.4	Performance comparison for transferring 16 bytes using CAN models.	51
2.5	CAN model selection.	55
2.6	Performance comparison of ColdFire Master bus models.	58
2.7	Speedup over bus functional model.	61
2.8	Average individual timing error for the low priority master.	62
3.1	Preemption complexity comparison.	76
3.2	Wait complexity with disturbance.	80
3.3	System simulation bandwidth [$\frac{MBytes}{sec}$].	88
4.1	Features and layers in abstract processor models.	109
4.2	Simulation performance of software-only systems.	115
4.3	Simulation performance of HW/SW systems.	116
4.4	Simulation accuracy of SW-only systems.	117
4.5	Simulation accuracy of HW/SW systems.	118
4.6	Trade-off for RAZR system simulation.	119

List of Acronyms

AHB Advanced High-performance Bus. System bus definition within the AMBA 2.0 specification. Defines a high-performance bus including pipelined access, bursts, split and retry operations.

AMBA Advanced Microprocessor Bus Architecture. Bus system defined by ARM Technologies for system-on-chip architectures.

APB Advanced Peripheral Bus. Peripheral bus definition within the AMBA 2.0 specification. The bus is used for low power peripheral devices, with a simple interface logic.

ASB Advanced System Bus. System bus definition within the AMBA 2.0 specification. Defines a high-performance bus including pipelined access and bursts.

ASIC Application Specific Integrated Circuit. An integrated circuit, chip, that is custom designed for a specific application, as supposed to a general-purpose chip like a microprocessor.

ATLM Arbitrated Transaction Level Model. A model of a system in which communication is described as transactions, abstract of pins and wires. In addition to what is provided by the TLM, it models arbitration on a bus transaction level.

Behavior An encapsulating entity, which describes computation and functionality in the form of an algorithm.

BFM Bus Functional Model. A pin-accurate and cycle-accurate model of a bus (see also PCAM).

CAD Computer Aided Design. Design of systems or products assisted by computer technology, i.e. by use of software tools.

CAN Controller Area Network. Serial communications protocol with a focus for automotive applications.

CE Communication Element. A system component that is part of the communication architecture for transmission of data between PEs, e.g. a transducer, an arbiter, or an interrupt controller.

Channel An encapsulating entity, which abstractly describes communication between two or more partners.

CLI Cycle Level Interface. Refers to ARMs definition of the AMBA bus, cycle level accurate for SystemC.

DFG Data Flow Graph. An abstract description of computation capturing operations (nodes) and their dependencies (operands).

DSP Digital Signal Processor. A specialized microprocessor for the manipulation of digital audio and video signals.

HCFSM Hierarchical Concurrent Finite State Machine. An extension of the FSM that explicitly expresses hierarchy and concurrency.

HDL Hardware Description Language. A language for describing and modeling blocks of hardware.

FPGA Field Programmable Gate Array. An integrated circuit composed of an array of configurable logic cells, each programmable to execute a simple function, surrounded by a periphery of I/O cells.

FSM Finite State Machine. A model of computation that captures an algorithm in states and rules for transitions between the states.

FSMD Finite State Machine with Datapath. Abstract model of computation describing the states and state transitions of an algorithm like a FSM and the computation within a state using a DFG.

HAL Hardware Abstraction Layer. An implementation of a software API providing common access to a hardware platform independent of the actual implementation.

HW Hardware. The tangible part of a computer system that is physically implemented.

ISA Instruction Set Architecture. A description of the programmer visible portion of a processor, describes the boundary between hardware and software, typically in terms of instructions and registers.

ISO International Organization for Standardization

ISS Instruction Set Simulator. Simulates execution of software on a processor at the ISA level.

IP Intellectual Property. A pre-designed system component.

MAC Media Access Control. Layer within the OSI layering scheme.

MoC Model of Computation. A meta model that defines syntax and semantic to formally describe any computation, usually for the purpose of analysis.

MPSoC Multi-Processor System-on-Chip. A highly integrated device implementing a complete computer system with multiple processors on a single chip.

OS Operating System. Software entity that manages and controls access to the hardware of a computer system. It usually provides scheduling, synchronization and communication primitives.

OSI Open Systems Interconnection. A communication architecture model, described in seven layers, developed by the ISO for the interconnection of data communication systems.

PE Processing Element. A system component that provides computation capabilities, e.g. a custom hardware or generic processor.

PCAM Pin-accurate and Cycle-Accurate Model. An abstract model that accurately captures all pins (wires) and is cycle timing accurate.

PSM Program State Machine. A powerful model of computation that allows programming language constructs to be included in leaf nodes of a HCFSM.

RTL Register Transfer Level. Description of hardware at the level of digital data paths, the data transfer and its storage.

RTOS Real-Time Operating System. An operating system that responds to an external event within a predictable time.

SCE SoC Environment. A set of tools for the automated, computer-aided design of SoC and computer systems.

ROM Result Oriented Modeling. An approach for fast and abstract modeling of a process with limited visibility to internal state changes.

SoC System-On-Chip. A highly integrated device implementing a complete computer system on a single chip.

SLDL System-Level Design Language. A language for describing a heterogeneous system consisting of hardware and software at a high level of abstraction.

TLM Transaction Level Model. A model of a system in which communication is described as transactions, abstract of pins and wires.

UML Unified Modeling Language. A standardized general-purpose modeling language which includes a graphical notation used to create an abstract model of a system, referred to as a UML model.

Acknowledgments

I want to thank those who have supported me during the process of the thesis work. First and foremost I want to thank my advisor, Prof. Rainer Dömer, for his guidance and support throughout the Ph.D. degree journey. His technical ideas, his organizational talents, and his focus on doing things right very much inspired me. Especially, I appreciate our constructive discussions, which supported me in identifying, isolating and solving problems. His positive and precise advice has tremendously helped me in reaching my goals in the program. I am also very grateful for his patience, which I utilized especially toward the end of my degree when trying to decide for the next career step.

I want to thank Prof. Daniel Gajski for serving on my committee. His critical, yet visionary comments and discussions very much enriched the research and work environment. In addition, I would also like to thank Prof. Pai Chou for serving on my committee and for his valuable comments on improving this thesis. I would like to thank Andreas Gerstlauer for his contribution of ideas, the good discussions and for his patience throughout the process.

This thesis work was influenced by the members of the SpecC/SCE group, through discussions and meetings. The people are who make the Center for Embedded Computer Systems an excellent research place. In particular, I would like to thank Junyu Peng and Dongwan Shin for their support of the architecture and communication refinement tools. I was very fortunate to have their support in many occasions while running my experiments.

Finally, I want to thank the Fashion Island in Newport Beach, CA for establishing the salad bar, which as it turns out is the initial seed that made all this possible.

Curriculum Vitae

Gunar Schirner

Education

- 2008 **Ph.D.**, Electrical and Computer Engineering,
University of California, Irvine
- 2005 **M.S.**, Electrical and Computer Engineering
University of California, Irvine
- 1998 **Dipl.-Ing. (Berufsakademie)**, Technische Informatik,
Berlin, Germany

Experience

- 2004-2008 **Graduate Research Assistant**
Center for Embedded Computer Systems,
University of California, Irvine
- 2006-2007 **Pedagogical Fellow**
University of California, Irvine
- 2005-2007 **Teaching Assistant**
Henry Samueli School of Engineering,
University of California, Irvine
- 2003-2004 **Graduate Research Assistant**
Distributed Object Computing Laboratory,
University of California, Irvine
- 2000-2003 **Software Development Engineer III**
Alcatel USA,
Petaluma, CA
- 1998-2000 **Engineer for Software Development and System Planning**
Alcatel SEL AG,
Berlin, Germany
- 1995-1998 **Work Study**
Alcatel SEL AG,
Berlin, Germany

Publications

- J3.** Gunar Schirner, Andreas Gerstlauer, Rainer Dömer, “Fast and Accurate Processor Models for efficient MPSoC Design,” in *IEEE Transactions on CAD of Integrated Circuits and Systems(TCAD)*, under submission.
- J2.** Gunar Schirner, Rainer Dömer, “Result Oriented Modeling, a Novel Technique For Fast and Accurate TLM,” in *IEEE Transactions on CAD of Integrated Circuits and Systems(TCAD)*, vol. 26, no. 9, pp. 1688-1699, Sept. 2007.

- J1.** “Quantitative Analysis of the Speed/Accuracy Trade-off in Transaction Level Modeling,” in *ACM Transactions on Embedded Computing Systems (TECS)*, accepted for publication August 23, 2007.

Conference Papers

- C9.** Gunar Schirner, Rainer Dömer, “Introducing Preemptive Scheduling in Abstract RTOS Models using Result Oriented Modeling,” *Design Automation and Test in Europe (DATE)*, March 2008.
- C8.** Gunar Schirner, Andreas Gerstlauer, and Rainer Dömer. “Automatic Generation of Hardware dependent Software for MPSoCs from Abstract System Specifications“. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, Seoul, Korea, January 2008.
- C7.** Gunar Schirner, Gautam Sachdeva, Andreas Gerstlauer, and Rainer Dömer. “Embedded Software Development in an System-Level Design Flow: Case study for an ARM Processor“. In *Proceedings of the International Embedded Systems Symposium*, Irvine, CA, June 2007.
- C6.** Gunar Schirner, Andreas Gerstlauer, and Rainer Doemer. “Abstract, Multifaceted Modeling of Embedded Processors for System Level Design“. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, Yokohama, Japan, January 2007.
- C5.** Gunar Schirner and Rainer Dömer. “Fast and Accurate Transaction Level Models using Result Oriented Modeling“. In *Proceedings of the International Conference on Computer Aided Design (ICCAD)*, San Jose, CA, November 2006.
- C4.** Gunar Schirner and Rainer Dömer. “Accurate yet Fast Modeling of Real-Time Communication“ In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Seoul, Korea, October 2006.

- C3.** Gunar Schirner and Rainer 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.
- C2.** Gunar Schirner and Rainer 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.
- C1.** Gunar Schirner, Trevor Harmon, and Ray Klefstad. “Late Demarshalling: A Technique for Efficient Multi-language Middleware for Embedded Systems“. In *Proceedings of the International symposium on Distributed Objects and Applications (DOA)*, Larnaca, Cyprus, October 2004.

Technical Reports

- TR6.** Andreas Gerstlauer, Gunar Schirner, Dongwan Shin, Junyu Peng, Rainer Dömer, Danjel Gajski, “System-On-Chip Component Models“, UC Irvine, Technical Report CECS-TR-06-10, May 2006.
- TR5.** Gunar Schirner, Gautam Sachdeva, Andreas Gerstlauer, and Rainer Dömer. “Modeling, Simulation and Synthesis in an Embedded Software Design Flow for an ARM Processor“. Technical Report CECS-TR-06-06, Center for Embedded Computer Systems, University of California, Irvine, April 2006.
- TR4.** Andreas Gerstlauer, Gunar Schirner, Dongwan Shin, and Junyu Peng. “Necessary and Sufficient Functionality and Parameters for SoC Communication“. Technical Report CECS-TR-06-01, Center for Embedded Computer Systems, University of California, Irvine, May 2006.
- TR3.** Gunar Schirner and Rainer Dömer, “Using Result Oriented Modeling for Fast yet Accurate TLMs“. Technical Report CECS-TR-05-05, Center for Embedded Computer Systems, University of California, Irvine, May 2005.

- TR2.** Gunar Schirner and Rainer 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.
- TR1.** Pramod Chandraiah, Hans Gunar Schirner, Nirupama Srinivas, and Rainer Dömer, “*System-On Chip Modeling and Design: A Case Study on MP3 Decoder*“. Technical Report CECS-TR-04-17, Center for Embedded Computer Systems, University of California, Irvine, June 2004.

Abstract of the Dissertation

Analysis and Optimization of Transaction Level Models for Multi-Processor System-on-Chip Design

by

Hans Gunar Schirner

Doctor of Philosophy in Electrical and Computer Engineering

University of California, Irvine, 2008

Professor Rainer Dömer, Chair

The increasing complexity of modern embedded systems and systems-on-chip poses great challenges to the design process. An exploding number of alternatives has to be considered during the design process. Additionally, the amount of software with tight coupling to underlying hardware increases in current designs, adding another complexity dimension.

System-Level Design addresses these challenges by using a unified approach for hardware and software design. Raising the level of abstraction, system-level design uses fewer, abstract models of hardware and software for system analysis, exploration, simulation, and implementation. Well-defined and efficient models are crucial for reliable design space exploration. In particular, fast yet accurate models are needed to reduce the design time and improve the end product. In this dissertation, we address the modeling of Multi-Processor System-on-Chip (MPSoC) with Transaction Level Models (TLM) for two essential system elements, communication busses and software processors.

We contribute in three aspects. First, we systematically analyze communication models and quantify the speed/accuracy trade-off in TLM. We provide a classification of abstraction levels based on model granularity. In traditional models, each abstraction level improves the simulation speed by several orders of magnitude, however at a significant loss of accuracy. Second, we propose a novel modeling technique, Result Oriented Modeling (ROM), which removes the inaccuracy drawback of TLM, yet yields nearly the same speed. Third, we propose a fast alternative to traditional instruction set simulation, using a versatile processor model that shows speed gains of three orders of magnitude with only a few percent of error in accuracy.

Overall, our work guides the system developer in choosing the proper model features and provides efficient techniques to model them. It also supports the designer in model selection, analysis and implementation. As a result, our system modeling research will influence the design of digital embedded systems, resulting in better and less expensive end products while reducing the time-to-market.

Chapter 1

Introduction

Embedded systems play an important role in our everyday life. They are omnipresent in our environment, in virtually all application domains. To name a few, they process media data in consumer electronics, increase the safety and stability of automotive systems, control medical devices, and automate industrial processes. With the technological advances, an increasing number of products is based on embedded systems, which become pervasive and ubiquitous. Embedded systems by far outnumber classical workstation type computer systems. According to Netrino [8], only 2% of all manufactured processors in the year 2005 were used in workstations. The remaining 8.8 billion processors have been integrated into embedded systems. In the future, we can expect even more processors to be integrated into our everyday devices.

Embedded systems are integrated into a larger physical system or product in order to provide a few specific applications. They are constrained by external input and output. Following the definition in [63], the main reason for buying a product based on an embedded system is not the computational functionality by itself, but the overall product's external functionality. With the integration, many product challenges extend to the design of embedded systems. Many systems are mobile, thus battery operated, and require a power efficient implementation. At the same time, strict performance constraints demand high computational power, as for example in a portable media player decoding high-definition video. Additionally, embedded systems are often very complex, with tightly coupled Hardware (HW) and Software (SW), which for example controls a dynamic phys-

ical environment. In a modern car, for example, many Electronic Control Units (ECUs) control different aspects of a vehicle, such as fuel injection, electronic stability program and exhaust management. Already in the year 2004 [97] reported 50 to 80 ECUs for an upper class vehicle. These control systems are deeply integrated into the overall product and tightly coupled with the physical environment. With our reliance on products using embedded systems, many non-functional product requirements extend to the embedded system itself, such as dependability and real-time constraints. Meeting these requirements poses significant challenges on the design process.

In contrast to general purpose computing, the application and the operational environment of an embedded system are already known at design time. This results in a significant advantage, allowing to design a customized and optimized platform for a given product. The customization in turn may increase performance, allow for extra functionality, and/or meeting a tighter power budget. High volume applications may be implemented with a custom designed Application Specific Integrated Circuit (ASIC). Applications in a lower production volume, or systems demanding reconfigurable hardware can be realized using Field Programmable Gate Array (FPGA) technology. Modern manufacturing capabilities offer a high integration density, which enables combining multiple processors, together with customized hardware accelerators, communication hierarchy, I/O devices and drivers onto a single chip – a Multi-Processor System-on-Chip (MPSoC). A MPSoC basically contains a complete embedded system. This thesis addresses the modeling of complex MPSoCs in order to aid the design process.

The design complexity of modern MPSoC is exploding due to the market demand for more, increasingly complex features, the implementation flexibility and the high integration densities that allow to implement those complex features, and the pressure for shortening the time-to-market. To address the customer needs, and to remain competitive, the market demands an increasing number of increasingly more complex features. As one metric, the International Technology Roadmap for Semiconductors (ITRS) [99] quantifies the number of features for portable or consumer electronics doubling every two years. Technological improvements enable implementing more complex systems by allowing to integrate an increasing number of transistors onto a single chip. In its 2007 report, the ITRS [99] predicts 1.5 billion transistors to be integrated by 2009. Although the designs dramatically

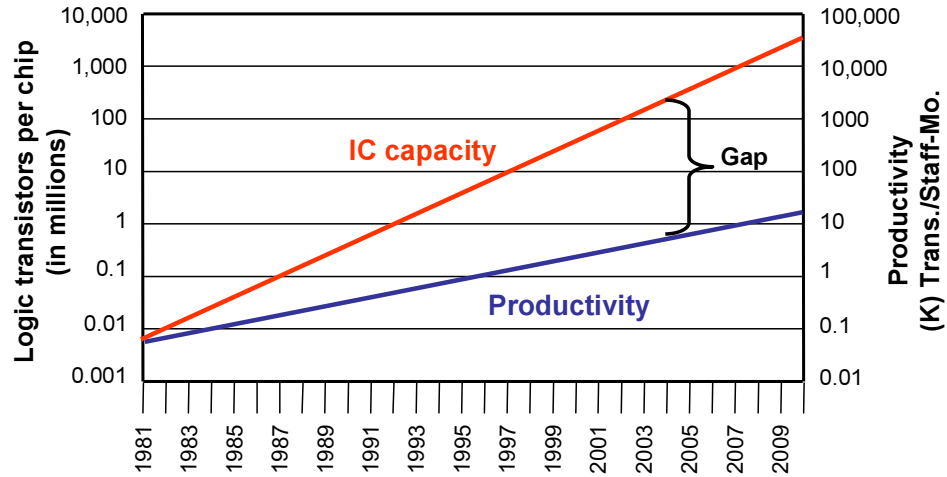


Figure 1.1: Productivity gap (courtesy [41]).

increase in complexity, the market still demands reducing the time-to-market to timely yield competitive products.

These conflicting demands lead to a significant productivity gap in the semiconductor industry, as reemphasized by ITRS [98] (2004). Figure 1.1 illustrates the productivity gap. It shows that over the years more transistors can be integrated onto a single chip than designed within the shortening time-to-market. Therefore, new approaches are needed to dramatically increase design productivity and to close the productivity gap. One such approach is utilizing hierarchy and designing at a higher level of abstraction, which enables constructing larger and more complex systems.

1.1 System-Level Design

The competitive market and the technological advances require a significant improvement in productivity when designing increasingly more complex embedded systems in a shorter amount of time. System-Level Design addresses these challenges by using a holistic approach. Instead of designing individual components separately, a complete embedded system is designed at once. Such a system under design typically contains one or more processors, custom or standardized hardware components, which accelerate computation or perform specialized functions (such as I/O), and a communication hierarchy connecting

the individual components. A system often also contains sensors and actuators to interact with the outside physical environment. Those actuators and sensors are mostly standardized components. The main focus of the system-level design rests on the digital portion. An essential aspect of system-level design is the hardware / software co-design, where both aspects of the system are jointly designed – concurrently at the same time.

Using a system-level approach offers many advantages. With a system-level view, the embedded system design starts early with a specific algorithmic system description independent of a particular hardware-software split. Jointly designing both aspects has the potential for more efficient designs, allowing for early, global optimizations across multiple layers. Furthermore, system-level design aims for a guided automatic generation of the target implementation and thereby dramatically increasing productivity. In particular, generating the communication interface between hardware and software has the potential to bridge the gap traditionally present between different organizations that are separately responsible for either HW or SW.

System-level design distinguishes three orthogonalized aspects: behavior description, structural mapping, and implementation. HW/SW co-design utilizes a system description in an implementation and platform agnostic format. For example, the behavior is described in algorithmic form and explicitly captures dependencies, instead of using implementation-detail, such as a Register Transfer Level (RTL) representation. Again, with the implementation independent format, a free mapping of behaviors to a platform structure becomes possible. In a subsequent more detailed process, the platform structures can be implemented, for example by using a set of standardized processors and custom acceleration hardware. The implementation optimization then is similar to traditional design processes.

An implementation-independent format naturally leads to abstraction, since specific low-level details have to be omitted. In system-level design, a system is hence captured as an abstract model that expresses the main properties, however hides implementation-level details. Using abstract models is the key to an efficient modeling process. Already in 2004, the ITRS [98] listed higher-level abstraction and specification as the first promising solution for tackling the system complexity. The same focus was more recently also highlighted by [81].

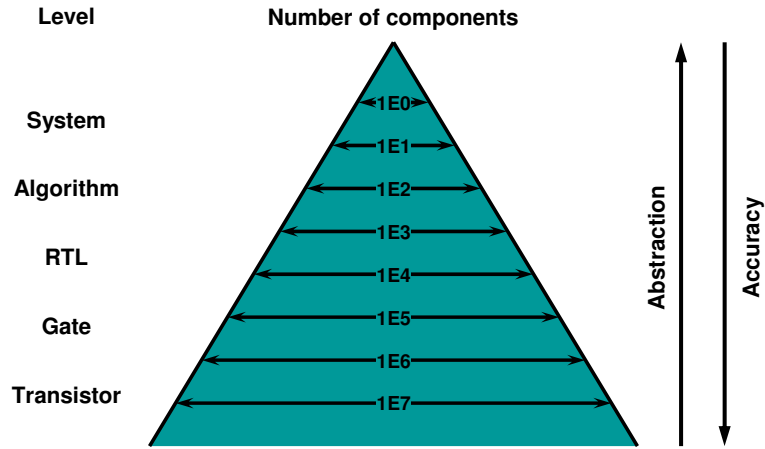


Figure 1.2: Abstraction levels in SoC design (source [32]).

With a higher level of abstraction a system can be composed out of fewer, yet more complex components using the concept of hierarchy. Figure 1.2 illustrates the relation between abstraction level and number of components. An embedded system that is initially composed out of tens of millions of transistors may only require tens of thousands of RTL components. These in turn may be represented by multiple tens of algorithms. Reducing the number of components to deal with at the same time, eases maintaining a system-level overview. However, with each abstraction level an increasing amount of implementation detail is hidden, which reduces the accuracy of the model. Ideally, system-level design allows describing a complete system solely as a composition of algorithms, so that the designer can focus on a purely functional system overview.

1.1.1 Methodology

Computer Aided Design (CAD) tools are utilized to establish an efficient design process. Such tools typically require adhering to a fixed procedure from specification to implementation, called a design *methodology*.

In a *top-down* methodology, a system is initially described at the highest abstraction level. The specification is then step-wise refined down to an actual implementation. With each refinement step, more implementation detail is added to the system description. Potentially after each refinement step, an analysis step investigates the effects of the implemented decisions.

In a *bottom-up* methodology, on the other hand, the design starts with simple basic blocks, called components. Then, more complex components are hierarchically composed out of these simple components. The process is iterative, and the previously defined complex components become the basic block for the new cycle. The process repeats until the complete system is composed. A bottom-up methodology is also referred to as component-based design.

A combination of both methodologies, a *meet-in-the-middle* methodology, may achieve the highest productivity. Then, a system design starts with a high level description, and is refined until predefined components (Intellectual Property (IP) components) can be instantiated out of a catalog.

The following paragraphs outline the process of a top-down design flow [29] to illustrate the decisions for refining an abstract specification down to an implementation.

In a top-down methodology, the SoC design starts with the specification model, which is a purely functional model – free of any implementation details. The functionality is algorithmically captured and encapsulated in behaviors. Behaviors communicate through abstract typed communication channels. The model is untimed and establishes only a causal ordering. The specification model allows a functional validation of the description. Once finished, it becomes a golden model, serving as a reference during the design cycle.

In the first refinement step, architecture information is added. For that Processing Elements (PEs) are inserted into the system and the behaviors composing the specification are mapped to them. PEs are programmable components, such as generic processor cores or Digital Signal Processors (DSPs), or non-programmable elements, such as customized hardware accelerators. PE parameters, such as clock frequency, are selected to adjust to the application demands. Based on embedded timing information of the PEs, an early runtime performance estimation gives initial feedback about the design decisions.

A next step in the refinement chain deals with defining scheduling decisions for PEs that host multiple behaviors. This refinement allows the designer to select suitable scheduling mechanisms, ranging from off-line static scheduling to priority based dynamic scheduling. In case of dynamic scheduling, behaviors are mapped to tasks for management by an operating system. This refinement step is essential especially for programmable PEs, which typically host many behaviors.

Communication decisions are captured in the following step. They define the communication hierarchy, the selection of busses and protocols. Now, the abstract communication channels, which have been introduced in the specification model, are mapped to physical busses and protocols. Detailed information about each utilized protocol is added, defining timing and structure. The resulting model includes specific instructions for the particular bus implementation, like the access logic for a bus master or bus slave.

The synthesis step concludes the design flow, addressing both HW and SW. Hardware synthesis generates RTL code for each custom hardware PE with the prerequisite of RTL component allocation, their functional mapping and scheduling. The hardware synthesis produces a cycle accurate description of each hardware PE. The synthesis step also includes software generation to implement the desired behavior using programmable processors. Here, specific implementation code is generated that performs internal communication, external communication with hardware components and potentially executes on top of a standard operating system. The output of the software generation is a cycle accurate model of each software-processing element, i.e. a target binary. The target binary can be simulated using an Instruction Set Simulator (ISS), or alternatively executed on the target processor. Combining the outputs of both synthesis parts yields an implementation model, containing a cycle-accurate description of the whole system.

1.1.2 System-Level Design Languages

In order to allow automated processing, abstract models have to be captured in a formal, machine analyzable language. Specific languages, so called System-Level Design Languages (SLDLs), have been developed or adapted for their use in system-level design. Common to all SLDLs is their ability to abstractly describe a system specification, covering hardware and software aspects. Ideally, a SLDL spans many abstraction levels so that it can be used throughout the design process, from an early abstract specification down to some implementation-level detail. The following paragraphs outline some SLDLs and their origins.

The Unified Modeling Language (UML) [71], which originated in software engineering, is a standardized visual specification language for object modeling that allows captur-

ing abstract system specifications. It offers a graphical input and representation of a large set of Models of Computation (MoCs) to flexibly express the system characteristics. Well defined subsets of UML are synthesizable into an implementation [62]. In addition, UML has been customized by the System Modeling Language (SysML) [70] to meet the needs of systems engineering. SysML is a UML profile and additionally introduces new concepts to support system-level design.

Matlab is a mathematical environment, which is used for algorithm development, and provides flexible simulation capabilities and a wide range of tools for visualizing results. Simulink extends Matlab to a multi-domain simulation environment with a graphical interface for model-based design. It offers both continuous time and discrete time models, as well as a wide range of predefined component blocks. Matlab/Simulink [64] is often used in control theory and digital signal processing.

Other approaches extend a Hardware Description Language (HDL). One example is SystemVerilog [103], which extends the widely used HDL Verilog to cater to system-level design. It embodies additional support for software concepts, such as an object-oriented programming model, and allows calling to and from C/C++ via its direct programming interface. Especially the latter significantly eases integration with software modules.

Finally, another set of languages emerged from standard sequential programming languages, such as C/C++. SystemC [42, 72] uses the object oriented features of the C++ language and is implemented as a library extension. Therefore, SystemC can be compiled with a standard C++ compiler. It provides C++ libraries to express and capture system-level aspects, such as concurrency and synchronization, as well as hardware aspects. SystemC is widely used and accepted in the industry and academia.

SpecC [29, 32] is based on a language extension approach and introduces new keywords to ANSI-C. Subsequently, it relies on a specialized compiler and simulation engine [68, 26, 114]. With SpecC being a language extension, the resulting SLDL is more concise and easier to learn than library extension based approaches [108]. The experimental work of this thesis has been performed using the SpecC language. The concepts however, are equally applicable to other SLDLs, such as SystemC, as well. Please see [29] for a detailed description of the SpecC and a comparison with other languages.

1.2 Abstract Models

By using a SLDL, a complete system, again with hardware and software, can be captured as an abstract model. An abstract model serves as a blueprint and reference for the implementation. Typically, an abstract model is executable, and simulates the system in a discrete event simulation [7]. In a discrete event simulation the system operation is represented as a chronological sequence of events. Each event occurs at an instant in time, updates the system state, and potentially increases the logical time by a discrete quantum.

Abstract models simulate multiple orders of magnitude faster than an implementation-level model (i.e. RTL). Increasing simulation performance is a key for simulating more complex systems and enables the designer to explore additional architectural alternatives in a given time period. An abstract model serves as a versatile platform for simulation-based validation, performance analysis, debugging and development. At the same time, the higher abstraction level allows the designer to focus on essential aspects of system design, without the burden of capturing all implementation details. This significantly reduces the modeling effort, since the number of components exponentially increases with each step toward implementation (see Figure 1.2). Therefore, using abstract models leads to a more efficient design process. However, abstracting implementation details, generally results in a reduced accuracy of the model, for example with respect to simulated timing. Therefore, it is important to find a suitable abstraction level, that yields fast simulation results while still providing sufficiently accurate results.

In general, a system is composed out of computation blocks that are connected by communication elements. The next two sections separately address abstraction of communication and computation.

1.2.1 Abstraction of Communication

Traditionally, communication has been abstractly described using distributed models of computation, such as Petri Nets [75], Kahn Process Networks (KPN) [51], and Synchronous Data Flow (SDF) [58]. Each of these models has an own set of well defined communication semantics, allowing for a detailed analysis of system communication (e.g.

for testing the scheduleability, or for determining buffer sizing). However, these models only provide very restrictive communication mechanisms.

For abstract communication modeling in the context of system-level design, transaction level modeling has been proposed [42]. Transaction level modeling abstracts communication in a system to whole transactions. It abstracts away low-level details about pins, wires and waveforms [17], and instead uses function call abstractions that provide the communication functionality. Although transaction level modeling has been widely accepted to abstract communication, the actual abstraction levels remain under debate.

1.2.1.1 OSI-based Abstraction

A generic view on possible abstraction levels can be derived from a traditional communication stack. For general network based communication, the International Organization for Standardization (ISO) provides a conceptual model organizing communication tasks and features. ISO defines in [50] the Open Systems Interconnection (OSI), a layer-based reference model. Each layer in this model has a well defined set of responsibilities, and provides services to the layer on top, hiding some implementation detail. By that principle, a layer higher in the stack can be seen as being more abstract than a lower layer. Thus, the OSI layering scheme can provide insight about possible abstraction levels. Table 1.1 enumerates the OSI layers with their main responsibilities.

Table 1.1 shows an overview of the layer separation, it also indicates where a particular layer is implemented and shows a representative code example for an invocation of each layer. The following list describes each layer in more detail. A more detailed description can be found in [31, chapter 5].

Application Layer. The application layer is the top most layer and implements the computational behavior of the system. The designer defines its basic content during the specification and the layer is gradually implemented throughout the development process. This application layer defines the system behavior and describes how the user data is processed in the system.

Presentation Layer. The presentation layer provides named channels, for the transfer of user typed data. User typed data (e.g. a data structure) is converted (marshalled)

Layer	Interface semantics	Functionality	Impl.	OSI
Application	N/A	• Computation	Application	7
Presentation	PE-to-PE, typed, named messages • <code>v1.send(struct myData)</code>	• Data formatting	Application	6
Session	PE-to-PE, untyped, named messages • <code>v1.send(void*, unsigned len)</code>	• Synchronization • Multiplexing	OS kernel	5
Transport	PE-to-PE streams of untyped messages • <code>strm1.send(void*, unsigned len)</code>	• Packeting • Flow control • Error correction	OS kernel	4
Network	PE-to-PE streams of packets • <code>strm1.send(struct Packet)</code>	• Routing	OS kernel	3
Link	Station-to-station logical links • <code>link1.send(void*, unsigned len)</code>	• Station typing • Synchronization	Driver	2b
Stream	Station-to-station control and data streams • <code>ctrl1.receive()</code> • <code>data1.write(void*, unsigned len)</code>	• Multiplexing • Addressing	Driver	2b
Media Access	Shared medium byte streams • <code>bus.write(int addr, void*, unsigned len)</code>	• Data slicing • Arbitration	HAL	2a
Protocol	Unregulated word/frame media transmission • <code>bus.writeWord(bit[] addr, bit[] data)</code>	• Protocol timing	Hardware	2a
Physical	Pins, wires • <code>A.drive(0)</code> • <code>D.sample()</code>	• Driving, sampling	Interconnect	1

Table 1.1: Communication layers (source [31]).

by the presentation layer into a sequence of bytes providing a system-wide common representation, which e.g. is independent of a PE's endianness. A transmission using the presentation layer is reliable, and can be synchronous or asynchronous.

Session Layer. The session layer typically is the interface between the software application and the Operating System (OS). It provides synchronous and asynchronous transport of untyped blocks of bytes. This layer provides services for end-to-end synchronization. In case the lower layer does not provide synchronous access itself, end-to-end synchronization is implemented here. Session layer channels are used for identification of individual software entities. Multiple message blocks may be

multiplexed into an untyped message stream within the transmitting stack. In such a case, the receiving stack will demultiplex the untyped message stream into message blocks.

Transport Layer. The transport layer provides reliable transmission of untyped streams between PEs in the system. A channel between two PEs acts as a pipe that carries the streams of the layers above. Generally, the transmission characteristics are asynchronous. The transport layer implements end-to-end flow control, as well as segmentation and reassembly, to split up the streams into smaller packets.

Network Layer. The network layer provides services to establish end-to-end paths, which connect two PEs, by routing packets through a set of point-to-point links, which connect adjacent stations along the route. The end-to-end paths carry packet streams from the layers above. The network layer completes the operating system kernel implementation for high-level end-to-end communication. For the routing of packets, the network layer provides separation of packets from different end-to-end paths going through the same station.

Link Layer. The link layer controls the link establishment between two directly connected (adjacent) stations and provides data exchange of uninterpreted packets of bytes. The link layer is the highest layer for a peripheral driver inside the operating system kernel. It defines the type of station (e.g. master / slave) and supports synchronization primitives by splitting each logical link into a separate data and control stream.

Stream Layer. The stream layer provides services for transporting control and data messages between stations. It implements addressing of streams to merge multiple separate data/control streams over a single shared medium. Data messages are uninterpreted blocks of bytes. The control message format, on the other hand, is heavily implementation dependent (e.g. interrupt handling, polling). The transfer services are generally asynchronous and unreliable. However, the effective reliability depends on synchronization on higher levels (e.g. through implementation of flow control).

Media Access Layer. The media access layer provides services to transfer an arbitrary length, contiguous block of bytes over the selected media. It hides the specific implementation details of the transmission medium. The media access layer is the lowest layer providing a medium independent access. In addition, the media access layer implements data slicing: an incoming data transfer request, called the user transaction, is split into individual bus transactions depending on the underlying medium.

Protocol Layer. The protocol layer provides transmission capabilities for individual bus transactions - words, shorts, bytes and defined lengths of blocks. This layer also performs arbitration for each bus transaction.

Physical Layer. The physical layer implements a bus cycle access to the physical wires. It performs sampling and driving of individual bus wires. Separate interfaces are provided for accessing the data, address and control portion of the bus. The physical layer also provides all implementation necessary for the bus connection scheme, i.e. in case of the Advanced High-performance Bus (AHB) the interconnection network consisting of multiplexers. Furthermore the physical implementation of arbitration is included.

In summary, the OSI layers offer a possible approach for abstraction from the physical implementation. With each layer, an increasing amount of implementation detail is hidden. While the physical layer deals with wire accesses and clock cycles, the protocol layer already provides services for transport of bus transactions independent of the clock cycle detail. The implementation-specific characteristics of the bus are hidden above by the media access layer, since it provides a point-to-point communication of arbitrary sized messages. Further up in the stack, above the network layer, even the hierarchy of the communication infrastructure is hidden by the provided end-to-end links, which connect two PEs regardless of the number of stations in between.

1.2.2 Abstraction of Computation

Traditionally, computation modeling was approached with specifically tailored MoCs, with the main focus on a static analysis of the system behavior. A common basis for many

MoCs is a Finite State Machine (FSM) representation, which expresses an algorithm as a set of states and rules for transitioning from one state to another. FSMs are typically used for control applications. A Data Flow Graph (DFG), on the other hand, focuses more on computation than control. A DFG is formally an acyclic directed graph, where each node in the graph represents an operation, and each arc between nodes represents a dependency (i.e. operands for the operation). Combining the FSM and DFG concepts yields the Finite State Machine with Datapath (FSMD). A FSMD can express both control and computation; it captures states (nodes) and transitions between states, while each state contains a DFG describing the computation executed in that particular state. The FSMD is a model typically used in behavioral synthesis. It translates to a controller and a datapath.

A further extension of the state machine concept, the Hierarchical Concurrent Finite State Machine (HCFSM), adds concurrency and hierarchy building. Each state in a HCFSM may consist of sub-states. Additionally, multiple states may execute in parallel. One representation of HCFSM is State Charts [43].

Common for all of the above MoCs is their focus on describing computation with a focus on analysis. For this purpose, each MoC provides well defined, yet restrictive execution semantics. As a result, capturing a larger, more complex system with a state machine approach leads to an explosion in the state space, which makes handling these models difficult. To allow more complex states, the Program State Machine (PSM) [105] allows programming language constructs being used as a state description. A PSM is a hierarchical concurrent FSMD, where the leaf states contain program statements. It is a very powerful computational model, which allows for a concise system description. On the other hand, the powerful computational model significantly complicates analysis, which has shifted the focus from a static analysis toward a simulation-based analysis. The PSM is used in the SpecC SLDL and is present in other SLDLs as well.

Software simulation has traditionally been performed using Instruction Set Simulators (ISSs). An ISS simulates the Instruction Set Architecture (ISA) of a processor, interpreting the instructions of a binary stream. It provides functional-accurate simulation and simulates the processor's micro architecture to provide timing-accurate simulation on a host platform at a very fine granularity. ISS-based approaches are widely used in academia [9, 109] and in industry [3, 107, 24].

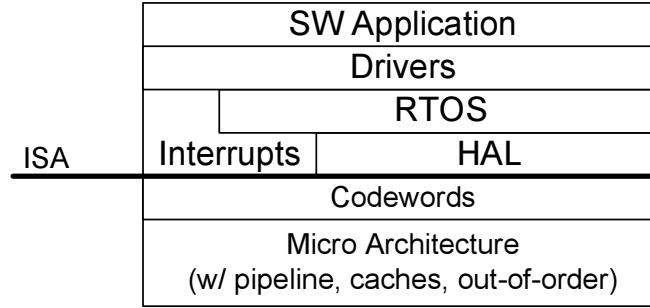


Figure 1.3: Software execution stack.

However, interpreting ISSs simulate very slowly, especially when multiple instances are integrated into a MPSoC system simulation. Furthermore, the final software binary is needed for an ISS-based simulation. Hence, it requires a detailed implementation of all software components, as outlined in Figure 1.3.

In particular, an ISS-based simulation requires the final implementation of the Hardware Abstraction Layer (HAL), interrupts, Real-Time Operating System (RTOS), and drivers to execute a software application. The *HAL* abstracts most of the hardware specific details of the processor. To name a few, it implements a low-level bus access, provides an API to access the processor registers and offers basic context switching capabilities. The *RTOS* implementation on top of the HAL provides real-time multi-tasking capabilities as well as communication and synchronization primitives for communication within the processor. *Interrupts* and *Drivers* provide services for synchronization and communication with external devices, such as hardware accelerators.

The effort for creating a detailed implementation of all the above described software components limits design space exploration. Therefore, software execution has to be abstracted above the ISA-level, hiding some of the implementation detail to achieve an efficient abstract system modeling.

One possible abstraction above the target ISA utilizes a host-compiled RTOS, such as the commercial RTOS simulator VxWorks Simulator [49] (previously known as VxSim). Both, the application and the RTOS are compiled to execute on top of the simulation host. The host-compiled RTOS provides the full RTOS API to the simulated application. Communication with external components, however, has to be manually emulated (e.g. through a socket based communication). Similar academic approaches include [47].

An even higher abstraction employs an abstract model of the system, including an abstract RTOS implemented on top of a SLDL. By abstracting the RTOS a higher simulation speed can be achieved, however the resulting model is less accurate (e.g. in terms of observable features). It is clear, that similar to the abstraction of communication, different abstractions are feasible for modeling computation. The level of abstraction then determines the observable features, the accuracy of the model (e.g. in terms of timing accuracy, or accuracy in terms of power estimation) and also influences the simulation performance.

1.2.3 Basic Models in System-level Design

By combining an abstract description of communication and computation, a complete system can be abstractly captured. Many models with fine nuances in abstraction are possible (e.g. when using the ISO OSI communication layering scheme as a guidance). For a practical application however, it is useful to restrict to fewer models for a more concise system design. We propose three basic models for capturing systems: a high-level *Specification Model*, a performance-expressing *Transaction Level Model* and a detailed *Pin-Accurate, Cycle-Accurate Model*. These three models are visualized in Figure 1.4. It shows two applications mapped to individual PEs, which communicate with each other through a communication stack.

Specification Model. The specification model is the most abstract model. At this abstraction level, the applications directly communicate through abstract channels and none of the other OSI layers is implemented. The specification model is the starting point in a top-down design flow. It describes the algorithms of the system and their dependencies in an untimed and platform-agnostic form using a SLDL. Important for a flexible and analyzable input specification is the separation of computation and communication, which allows automatically refining the communication and mapping of computation to separate PEs.

In the application layer, the system functionality is described as algorithms that have been split into multiple parallel / sequential processes. Communication between applications is performed using typed channels on the application layer. These channels

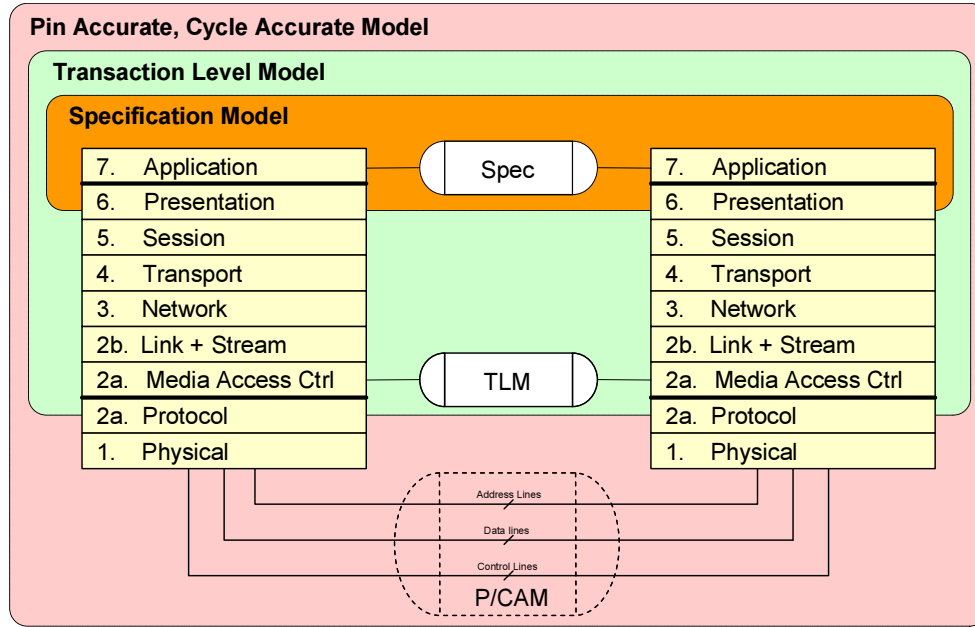


Figure 1.4: Abstraction layers of communication.

provide high-level communication semantics for synchronization and storage. Examples of channels include synchronous blocking channels (double handshake), asynchronous buffered channels (e.g. FIFO, queue) and synchronization only channels (e.g. mutex, semaphore, barrier channel). The high-level channels are very similar to communication primitives offered by a classical RTOS, in addition however, they provide typed communication (e.g. transfer of complex data structures).

Transaction Level Model. The Transaction Level Model (TLM) implements part of the communication stack to reveal performance implications of the implementation choices. It is used by the platform designer (and the application designer) to validate system functionality and for analyzing the system performance.

The TLM refines communication between PEs over multiple layers of the reference model. In the visualized example, each virtual PE implements the communication stack down to the Media Access Control (MAC) layer and the stacks are connected by an abstract transaction level model of the communication medium.

To reveal implication of communication architectural choices, the TLM resolves communication down to the level of point-to-point communication as introduced by

the Link layer. The remaining layers are abstracted within the TLM channel that connects the two stacks. Since the TLM in this example is implemented at the MAC level, the TLM transports contiguous blocks of bytes while reflecting the characteristics of abstracted communication medium (e.g. with respect of timing). The level at which the TLM abstracts communication is flexible. Depending on the desired detail level, observable features, and simulation speed the number of abstracted layers within the TLM can be varied.

The TLM serves as an analysis platform for the design space exploration, to estimate the system performance. It also is an platform to further refine and develop software and hardware.

Pin- and Cycle-Accurate Model. The most detailed model of the system is the Pin-accurate and Cycle-Accurate Model (PCAM) (also referred to as Bus Functional Model (BFM)). The PCAM implements all layers of the communication stack. The two communication stacks are connected by abstract wires, which accurately reflect the connectivity of the implemented communication platform. Communication partners exchange data and synchronization using the explicitly modeled wires in a cycle-accurate manner. With the high detail level, the PCAM serves as a detailed analysis platform, for example for observing detailed communication statistics. Also, the PCAM offers waveform-level detail, which allows integrating existing RTL Intellectual Property (IP) and furthermore eases comparison with real hardware. The detail level of a PCAM serves as a final validation before handover to the system synthesis.

1.2.4 TLM Trade-off

As indicated before, the level at which to implement a TLM is a design choice. With a high abstraction, the simulation speed increases, however this typically leads also to a loss of accuracy. In general, TLMs pose a trade-off between an improvement in simulation speed and a loss in accuracy. This trade-off is present for both abstracting communication as well as computation. The trade-off is visualized in in Figure 1.5.

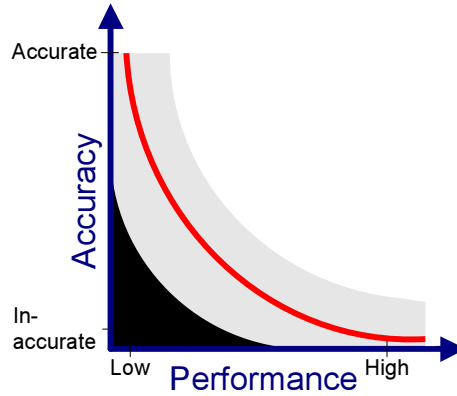


Figure 1.5: Transaction Level Modeling Trade-Off.

The TLM trade-off deals with weighing the detail level of a model, hence its accuracy, against the achievable simulation speed. To illustrate the extremes, an abstract model that is very close to the implementation, would reveal most implementation detail. Hence, such a model would yield a high accuracy. However, with the large detail level, such a model would reach a low simulation performance (low simulation speed). A very abstract model on the other hand, would achieve the opposite. Most of the implementation details are abstracted away, which typically leads to a fast simulation, however produces inaccurate results.

The trade-off essentially allows models at different degrees of accuracy and speed that range between these two extremes. However, having both high speed and high accuracy at the same time is typically not possible. The gray area of the diagram indicates models that follow the TLM trade-off. In contrast, models in the dark area, which are slow and inaccurate, are existent, however are practically not relevant. On the other hand, models that are both fast and accurate, which would be placed in the white area in top right of the diagram, are highly desirable but typically not achievable.

Although abstract modeling in form of TLM has been generally accepted as one solution to tackle the complexity in SoC design, this TLM trade-off however, has not been examined in detail. The TLM trade-off is a main aspect of this dissertation. Hence, the TLM trade-off will be addressed from several perspectives in separate chapters.

1.3 Dissertation Goals

With the dramatic increase of complexity of modern MPSoC designs, abstract models become crucial for an efficient system-level design. Fast simulating system models, which are still sufficiently accurate, are needed for system analysis, development and validation.

Well defined abstraction levels are crucial for the success and acceptance of system-level design. For an efficient design process, concise models are necessary that are expressive enough to exhibit important features, yet offer excellent simulation speed to allow an extensive design space exploration and a fast turn around time. Additionally, clearly defined abstraction levels and modeling styles are crucial for the interoperability between models of different vendors.

This dissertation aims at addressing abstract modeling issues in the following aspects:

- Identify proper abstraction levels for communication and computation.
- Identify test setups and measurement metrics for quantitatively analyzing abstract models.
- Quantitatively analyze the TLM trade-off for representable model examples for the gain in performance and loss in (timing) accuracy.
- Guide the model designer in efficiently abstracting communication and computation.
- Guide the user of abstract system models in selection of suitable models for a given simulation purpose.
- Explore alternative abstract modeling techniques to increase both performance and accuracy at the same time.
- Define modeling techniques for abstracting computation above the ISA for a timed simulation of software execution.

1.4 Dissertation Overview

The remainder of this dissertation is organized as following. First, the relevant related work is introduced and categorized in Section 1.5. Then, Chapter 2 systematically analyzes and quantifies the speed/accuracy trade-off in TLM. To this end, it provides a classification of TLM abstraction levels based on model granularity and defines appropriate metrics and test setups to quantitatively measure and compare the performance and accuracy of such models. Chapter 3 proposes a novel modeling technique, called Result Oriented Modeling (ROM), which removes the inaccuracy drawback of TLM in many cases. Using ROM, simulation models yield nearly the same speed as their traditional TLM counterparts, yet are still 100% accurate in timing. Chapter 4 focuses on abstracting computation on a software processing element. It introduces our approach of abstract processor modeling in the context of multi-processor architectures. The chapter combines modeling of computation on processors with an abstract RTOS model and accurate interrupt handling into a versatile, multi-faceted processor model with several levels of features. Finally, Chapter 5 summarizes and concludes this dissertation.

1.5 Related Work

This section briefly describes relevant related work.

1.5.1 Languages for System-Level Design

System-level modeling has become an important research area that aims to improve the SoC design process and its productivity. Languages for capturing SoC models have been developed, which have emerged from very different backgrounds.

From the mathematical modeling background, Matlab/Simulink [64] has emerged which is often used in modeling control systems and digital signal processing solutions. It combines discrete timed and continuous time models, a large range of predefined blocks, together with a wide range of visualization tools of the base-product, Matlab. From the software engineering background, UML [71] and its customization SysML [70] have emerged.

They provide a graphical input and a graphical representation of different models of computation. SystemVerilog [103] is an example of a SLDL that is based on a hardware description language, which has been extended for system use and for the description of software aspects. Finally, many system languages are based on generic programming languages, such as C, C++, and Java. Examples of SLDLs based on programming languages are SpecC [29], SystemC [42] and OpenJ [113]. These languages provide means to abstractly capture systems, but by themselves do not define modeling and abstraction approaches.

1.5.2 Abstraction and Analysis of Communication

We group abstraction and analysis of communication into three categories: (a) analytical approach, (b) trace-based approach, and (c) functional simulation approach.

1.5.2.1 Analytical Communication Performance Analysis

For an analytical approach, the system is described in a well defined distributed model of computation, such as Petri Nets [75], Kahn Process Networks (KPN) [51], and Synchronous Data Flow (SDF) [58]. Using well defined, yet restrictive, semantics allows to analytically reason about the system performance, and statically determine scheduling and configuration (e.g. queue sizes of a KPN implementation).

1.5.2.2 Trace-based Communication Performance Analysis

A trace-based approach separates a functional simulation from a simulation of the communication architecture. Communication activity (traces) are extracted during a functional simulation either with an abstract model or using reference hardware, and converted into architecture level communication primitives [60]. These traces are then later replayed on the communication architecture under design to optimize and configure the communication system. Hybrid approaches integrate trace generation within a functional simulation with the analysis and application of traces [55].

1.5.2.3 Analysis Based on Functional Simulation

Capturing and designing communication architectures using TLM [42] has received much attention. Cai and Gajski [17] provide an initial taxonomy of TLM. [80] define a

standard for transaction level modeling in SystemC. The main body of related work focuses on describing individual approaches to abstracting aspects of communication. Although they provide valuable guidance, none formally quantify the benefits and drawbacks of abstract communication modeling.

Sgroi et al. [100] address the SoC communication with a Network-on-Chip approach. Here, communication is partitioned into layers following the OSI structure. Software reuse is promoted with an increase of abstraction from the underlying communication. While this paper guides on the organization of communication, it does not directly address transaction level modeling.

Siegmund and Müller [101] describe with SystemC^{SV} an extension to SystemC and propose SoC modeling at three different levels of abstraction: physical description at RTL, a more abstract model for individual messages, and a most abstract model utilizing transactions. The abstraction levels used in this dissertation are similar to what Siegmund and Müller describe. The paper focuses on the interface description allowing a multi-level simulation. However, it does not address abstract modeling of multi-master busses.

Brem and Müller [14] describes how the CAN bus is modeled using the above mentioned extension SystemC^{SV}. The work also shows the three abstraction levels, but does not give any experimental results on performance or accuracy.

In [20] Caldari et al. describe the results of capturing the AMBA rev. 2.0 bus standard in SystemC. The bus system has been modeled at two levels of abstraction, first a bus-functional model at RTL, and second a model at transaction level simulating individual bus transactions. The described state machine based TLM reaches a speedup of 100 over the RTL model. Our abstraction approach described Chapter 2, however, reaches a higher speedup (three orders of magnitude over the BFM for the AMBA AHB) by avoiding explicit internal states.

Coppola et al. [23] also propose abstract communication modeling. They present the IPSIM framework and show its efficient simulation. While the paper delivers a general overview of the SoC refinement and introduces their intra-module interface, it does not supply details of the bus modeling itself as we will present in Chapter 2.

Gerstlauer et al. describe in [36] a layered approach and propose models that implement an increasing number of ISO OSI layers [50]. [36] presents how to arrange communication

and the granularity levels of simulation. However, it does not provide insight on the bus specific modeling.

Haverinen et al. [45] describe in a white paper three TLMs with increasing abstraction for the OCP-IP protocol. Only their most detailed TL-1 is cycle accurate. They do not show an accuracy analysis for the more abstract models.

Abstract communication is also used in Ptolemy as presented in [56] and [46] with an extension of dynamic switching between abstraction levels. A common point is the loss in accuracy with abstraction, which the work in this thesis eliminates.

Ghenassia describes in [39] transaction level modeling from an industry perspective, stating what is current and practical for industry applications. This work also supports the general trade-off between abstraction and accuracy.

Pasricha et al. [73] describe an approach using transaction-based abstraction. The paper introduces the concept of a model that is cycle count accurate at transaction boundaries (CCATB). It takes advantage of the limited observability of a transaction to increase simulation performance. However, only a very limited speedup of 55% over the bus functional model is achieved. Their approach models individual bus transactions and uses an active thread for the bus simulation. Our optimized abstract modeling technique, ROM, which we describe in Chapter 3, also utilizes limiting the observability within a transaction to gain simulation performance. Our ROM approach, however, is conceptually different. We raise the abstraction to user transactions (potentially spanning multiple bus transactions) and avoid a dedicated thread. Consequently, ROM achieves a higher speedup of up to 4 orders of magnitude. In other words, while Pasricha et al. use an extra thread, in our approach master and slave communicate directly through a shared channel without the need of a separate thread.

Timed abstract simulation has also been incorporated into commercial products. For example, the discrete event simulation engine in the VCC environment [57], supports several delay models (e.g. explicitly distributed by the designer, or by an automatic back annotation approach). VCC models preemption for software tasks and bus accesses by use of *suspend()* and *resume()* messages to the simulation task, which are taken into account when a task executes a *delay()* function. With that, VCC uses explicit test points (i.e. the *delay()* call) to account for preemptions as a traditional TLM. While [57] mostly focuses on

the simulation framework, our work introduces a modeling technique (that actually can be implemented using the VCC framework). Our modeling technique ROM (see Chapter 3) minimizes the interactions with the simulation engine to handle preemptive bus modeling¹.

Real-time communication has been analyzed in previous work. [104], for example, analyzes different CAN controllers. System level modeling of real-time communication is not explored as much. [106] describes abstract real-time communication for a class of LAN protocols by modeling LAN characteristics.

Real-time analysis, e.g. Audsley et al. [6], often captures a task's execution time as the task's computation delay (could be referred to as base time) and the sum of all external influences, such as preemptions, shared resource accesses, or communication. We use a similar terminology to describe the ROM approach. However, while the real-time analysis relies on an off-line static calculation, our approach computes the effects of any disturbances dynamically at run-time.

1.5.3 Abstraction and Analysis of Computation

For the purpose of organization, computation abstraction can be grouped into (a) analytical approaches, (b) simulation at the ISA-level, and (c) simulation above the ISA-level.

1.5.3.1 Analytical Computation Performance Analysis

Traditionally, computation abstraction and analysis has often been approached by using well defined MoCs. A MoC provides well defined execution semantics, however is restrictive to allow an analytical reasoning about the system behavior without requiring a functional simulation. Example MoCs are, Finite State Machine (FSM), which sets the foundation for many other MoCs capturing states and transitions, FSMD, which combines FSM and DFG, and HCFSM, which extends further by adding concurrency and hierarchy. The main application of these MoC is a static analysis for reasoning about the system performance. The work in this thesis, however, focuses on the functional timed simulation.

¹ Note that our approach does *not* require any change to existing discrete event simulation engines, such as SystemC [42] and SpecC [29]. ROM only uses the standard *wait-for-time* interface.

1.5.3.2 Simulation at the Instruction Set Architecture Level

Instruction set simulators (ISS) traditionally are important for simulating software execution. ISS examples are Shade [22], the SimpleScalar tool set [16, 61], or SimIT ARM [78]. Significant research effort has been invested to improve ISS performance [59, 79, 115, 112, 13, 19], mainly by using compiled instruction set simulation. For a host-compiled ISS the target binary is translated off-line into a host executable form, eliminating the costly interpretation of the binary stream at runtime. Disadvantages of compiled ISS typically include the large memory requirement, and the inability to handle runtime alterations of the binary. Nohl et al. [69] present just-in-time cache compiled simulation, a hybrid between compiled ISS and interpretive ISS, which maintains the performance of a compiled solution while reducing the memory constraints.

A system simulation of both software and the surrounding hardware is traditionally obtained by an ISS based co-simulation, integrating one or more ISSs into an abstract system model. Commercial solutions are provided by many vendors, such as ARM's SoC Designer with MaxSim Technology [3], VaST Systems' virtual system prototyping tools [107] and CoWare's [24] Virtual Platform Designer. In addition, ISS based co-simulation is used in many academic projects, such as the MPARM platform [9] or the ISS-based NoC simulator [109].

1.5.3.3 Simulation above the Instruction Set Architecture Level

Abstracting computation beyond the level of the ISA, as an essential element of the system level exploration, on the other hand, has been introduced only recently.

One approach for abstraction, which is still relatively close to the implementation, is a host-compiled RTOS approach. In such a system, a processor is abstractly represented by the RTOS that executes on top of the processor. For simulation, the target RTOS is ported to execute natively on the simulation host. It is called host-compiled, since it uses the compiler of the simulation host. One commercial example is the VxWorks Simulator (previously called VxSim) by WindRiver [102]. It uses a host-compiled version of the VxWorks RTOS that executes as a process on the simulation host (e.g. Solaris, Linux). A system simulation of multiple processors is achieved by running multiple instances of the

VxWorks Simulator, each in an own process. Customized socket communication connects instances of the VxWorks Simulator on the simulation host. This host-compiled RTOS approach has the advantage that the full RTOS API is available to the application. However, the execution timing provided by the VxWorks simulator is coupled to the simulation host (i.e. no targeted progress of time is provided), therefore it will not offer any feedback of the expected target performance.

Honda et al. [47] describe a RTOS centric cosimulator, which is mainly oriented toward the software designer. Here as well, a host compiled RTOS is used. External communication is modeled through the RTOS and external hardware is simulated. However, this solution does not include targeted execution time simulation.

A host-compiled RTOS approach requires the software to use the RTOS API, which already ties the model to a RTOS-specific implementation. While this approach provides detailed information, higher-level models are desirable to be utilized in earlier stages of the design process.

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. Their approach includes multiple processors on a higher level of abstraction. In contrast, our processor modeling described in Chapter 4 provides a finer grained model with the resulting feature observability advantages at similar simulation performance levels. Kempf et al. [52] introduce their Virtual Processing Unit for analysis of task mapping and scheduling effects using a quantitative model. They do not, however, include any processor specific features, such as interrupts.

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.

Chapter 2

Transaction Level Modeling

2.1 Introduction

Designing embedded systems and System-On-Chip (SoC) becomes increasingly challenging. The design space to be explored grows with increasing complexity, while at the same time shorter product life cycles require a shorter time-to-market. Addressing this gap has been the aim of system-level research. As one main approach, abstract models have been introduced to tackle the design complexity. For one, abstract models exhibit tremendous gains in simulation speed, allowing fast validation and extensive design space exploration.

For communication in particular, Transaction Level Modeling (TLM) has been proposed [42]. 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 [91].

2.1.1 TLM Trade-off

In general, TLMs pose a trade-off between an improvement in simulation speed and a loss in accuracy, as illustrated in Figure 2.1. The trade-off essentially allows models at different degrees of accuracy and speed. However, having both high speed and high

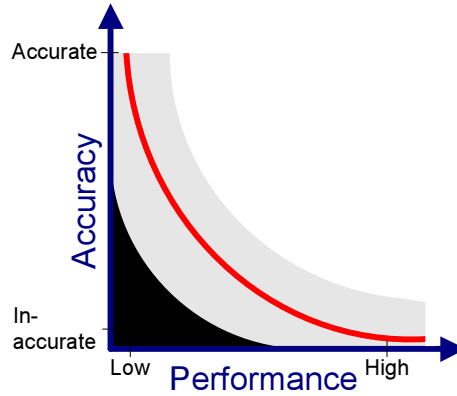


Figure 2.1: Transaction Level Modeling Trade-Off (repeated Figure 1.5).

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. 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.

2.1.2 Problem Statement

For this chapter, we define the problem as follows: a quantitative analysis of the trade-off between simulation performance and accuracy is needed. Definitions of generic metrics and a test framework, applicable to a range of communication protocols, are required to perform the analysis. The immediate analysis goal is to confirm the existence of the TLM trade-off. As a broader goal, we also expect conclusions on (a) classification of abstraction levels, (b) guidelines for model designers in efficiently abstracting communication, and (c) guidance of communication model users in the selection of a suitable model for a given design task.

2.1.3 Overview

In this chapter, we 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. For our analysis, we define in Section 2.3 proper metrics and test setups for measuring the performance improvement and the accuracy loss. For each bus system, we use models at different abstraction levels as defined in Section 2.2. In particular, we use two classes of TLMs (ATLM and TLM), and compare them against a fully accurate BFM as a reference.

Our measurements are based on examples from three different bus categories. In Section 2.4, we analyze the AHB of AMBA [4], as a representative of parallel on-chip bus systems with centralized arbitration and multiplexed interconnection scheme. AMBA is a widely used and industry-accepted standard for on-chip bus systems. For the second category of off-chip serial busses with distributed arbitration, we investigate the Controller Area Network (CAN) [11] in Section 2.5. This bus system dominates in automotive applications. Third, we analyze in Section 2.6 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 [66] that is used by the popular ColdFire MCF5206 processor. Based on the results of this range of examples, we will then generalize our TLM analysis in Section 2.7 and derive general conclusions in Section 2.8.

2.2 Transaction Level Modeling

TLM allows a wide variety of modeling styles and abstractions. It is not clearly defined in the literature, as also stated by [17] in their approach of structuring the models.

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 2.2 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

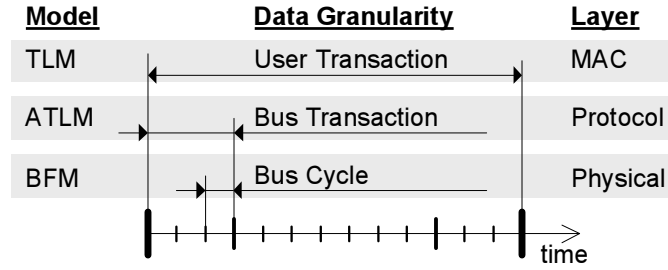


Figure 2.2: Model classes and their granularity.

primitives, such as a word transfer. Each bus transaction is usually processed in several *bus cycles*, which represent the finest granularity in our modeling.

Our classes of granularity follow the layers defined in the ISO OSI reference model [50]: the media access control (MAC), the protocol sublayer, and the physical layer. Each considered layer handles data and arbitration at its own granularity.

The *media access layer* provides a transmission service for a user transaction, which is a contiguous block of bytes. This layer divides the arbitrarily sized user transaction into smaller bus transactions observing the bus addressing rules, and transfers these byte blocks using the protocol layer. The *protocol layer* transfers data as bus transactions, which are bus primitives (e.g. bytes, words, or 4 word burst). It in turn uses the *physical layer*, which provides services to sample and drive individual bus wires at bus cycle granularity.

For dealing with the description complexity, we chose a layered architecture for our bus models similar to [36]. Using a system level modeling approach, each layer can be implemented as a separate channel using a system level design language (SLDL)¹. Then, different models can be easily created by composing the channels hierarchically.

According to our classes of granularity, we consider models at three different abstraction levels; the *Transaction Level Model* (TLM) that models user transactions, the *Arbitrated Transaction Level Model* (ATLM) at bus transaction granularity, and the bus cycle accurate *Bus Functional Model* (BFM).

¹ We have used SpecC [29] as the SLDL of choice, we could have used SystemC just as well.

2.2.1 Transaction Level Model (TLM)

The TLM² is the most abstract model; it only implements the media access layer. The user data, handled at the user transaction granularity, is transferred regardless of its size in one chunk using a single *memcpy*. Timing is simulated by a single *wait-for-time* statement, covering the whole user transaction. Arbitration is not modeled. Instead, concurrent access is resolved using a semaphore once per user transaction. The semaphore-based contention resolution depends on the simulation environment and is independent of the arbitration of the actual bus protocol.

2.2.2 Arbitrated Transaction Level Model (ATLM)

The ATLM simulates the bus access with bus transaction granularity (e.g. AHB bus primitives). It uses the MAC layer to split user transactions into bus transactions and implements an own abstracted protocol layer. The ATLM accurately models the bus arbitration for each bus transaction. We implement the arbitration and the bus without an own flow of execution to maximize simulation performance. Although this model correctly models arbitration, it is not pin-accurate and not cycle-accurate in all cases.

2.2.3 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 and covers all timing and functional properties of the bus definition. It handles arbitration per bus transaction and has the capability to take arbitration decisions on a bus cycle granularity. This model may include additional hardware, such as an arbiter, to correctly implement the bus standard.

² In the general sense we consider TLM to be a class of models and our bus models are a part of this class. To be consistent with our previous publications however, we have kept TLM also as a name of our most abstract model.

2.2.4 Comparison with other TLM Abstractions

In order to relate our TLM abstraction levels to other abstraction schemes, we will briefly compare our models to the OCP-IP and SystemC TLM standards.

In comparison to the OCP-IP TLM definitions [45], our models range between TL0 and TL3. Our BFM correlates most closely to TL0, since it models all bus wires, implements active bus components if applicable (e.g. multiplexers of AHB) and is based on an explicit clock. Our ATLM compares to TL2, since it uses bus transactions and is cycle-approximate in the general case³. Our TLM bus model lies between TL2 and TL3. It is more abstract than TL2, since it is based on user transactions (messages in the OCP-IP). On the other hand, it provides cycle-approximate timing, hence it is more detailed than TL3.

A comparison with the SystemC TLM standard [80] is more difficult, because the version 1.0 does not define the SystemC abstraction levels in detail. Our BFM matches the Cycle Callable (CC) model, since it is cycle-accurate and bit-accurate (CABA). In addition to the CC properties, our BFM is pin-accurate (PA) as well. Both our ATLM and the TLM could be called a Programmers View + Time (PVT) model, since both provide cycle-approximate (CX) timing.

2.3 Metrics and Measurement Setup

Given the model classification, we will now describe our metrics and setup for analyzing our models. We focus on two aspects. First, we look at the simulation *performance*, since a performance gain is the main premise of TLM. Second, we evaluate the *timing accuracy*.

2.3.1 Performance

Our metric for the performance is the simulation bandwidth. The simulation bandwidth (MBytes/s) quantifies the amount of data transferred in simulation when running for a second of real-time on a given simulation host. This metric can be compared to quantifying

³ The ATLM can be cycle-accurate for protocols without preemption, as we will show later.

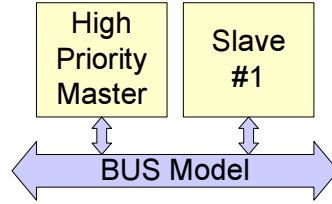


Figure 2.3: Single master setup for performance measurements.

the simulation speed of an ISS, which is typically expressed in MCycles/s. To contrast, the simulation bandwidth is independent of the simulated bandwidth, which would refer to the performance of the simulated communication system (for a unit of simulated time).

We have measured the simulation performance of each model in a minimal scenario with one master and one slave (Figure 2.3). A transaction with random content is performed repeatedly, without any delay in between. We have measured the simulation time (also referred to as real time or wall clock time) over a set of repeated iterations (e.g. 5000) and computed the simulation bandwidth. In order to estimate the scalability, we repeat this test for an increasing size of the user transaction.

Besides the data transfer and the controlling loop, the test masters and slaves do not execute any other code. This minimizes their impact on the overall simulation time and allows us to examine solely the performance of the bus models. All tests have been executed using the SpecC V 2.2.0 discrete event simulator using Quick Threads on a Pentium 4 (2.8 GHz) PC running Red Hat Enterprise WS 3.

2.3.2 Accuracy

Accuracy can be segregated into many aspects. For the purpose of this dissertation, we will distinguish three aspects of accuracy: functionality, represented feature detail, and timing⁴. For our analysis, we will fix the first, vary the second, and measure the third aspect.

The first aspect of functional accuracy requires that the data transmitted by the sender reaches correctly the receiver. Since this is a necessary requirement for a functional simu-

⁴ We omit one dimension of model accuracy, the "coding quality", due to the difficulty of a realistic measurement. In order to keep this factor as constant as possible, all models have been implemented with the same optimization effort by the author.

lation, all our models are functionally accurate. The second aspect is the represented detail level, which states how many features of the bus are actually present in the model. We vary this aspect in our model under test. The captured features range from accurately modeled pins and waveforms in the BFM up to a single *memcpy* and *wait-for-time* statement in the TLM⁵. The third aspect is the accuracy in timing which we measure in our tests. To compare the models, we use the timing of user transactions.

2.3.2.1 Timing Accuracy Metrics

The relevant measurements for expressing the accuracy of a model depend on the prediction goal of the simulation. We have identified two goals: first, the prediction of the application *latency* due to an individual bus access, and second, the prediction of the application *finish time*. For these goals, we use two metrics. First, by analyzing the *individual* duration of a user transaction, the application latency due to a bus access can be inferred. Second, the *cumulative* duration, which is the sum of all user transactions, shows how much the application finish time is delayed by the performed communication. For our tests, we prefer the cumulative transfer time over the actual finish time, since the latter includes the constant computation time between transactions (simulated by a delay), which is independent of the utilized bus model.

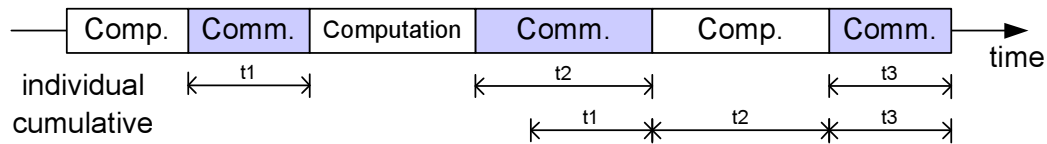


Figure 2.4: Cumulative and individual transfer time.

Figure 2.4 shows an example of a bus node's activity over time which progresses along the x-axis. Any process alternatively performs computation or communicates externally, the latter of which we are interested in. For the individual duration, we analyze the transfers t_1 , t_2 and t_3 separately. Our cumulative analysis is based on the sum of these three transfers.

⁵ A detailed table of represented features is given later in Section 2.5 for the CAN bus.

For the purpose of analyzing the accuracy based on the individual and cumulative duration, we define for this dissertation the *duration error* as the percentage error over the bus standard⁶:

$$\begin{aligned} d_{std} &: \text{duration as per standard} \\ d_{test} &: \text{duration in model under test} \\ error_i &= 100 * \frac{|d_{test} - d_{std}|}{d_{std}} \end{aligned} \quad (2.1)$$

Given this error definition, a timing accurate model exhibits 0% error. Note also that a particular model may have an error of more than 100% (i.e. the model under test shows more than twice the simulated time).

2.3.2.2 Bus Contention Metrics

Our models differ in the granularity of data and arbitration handling. We therefore expect a significant correlation between bus contention and accuracy. Thus, we use contention as an input to our measurements.

For our tests, we examine the status of a user transaction to determine the bus contention. This makes the contention definition independent of the actual bus implementation and can be applied to different bus systems. For our analysis, we define the bus contention as the percentage overlap between user transactions:

$$contention = 100 * \frac{\text{bus cycles with two or more active user transactions}}{\text{bus cycles with at least one active user transaction}} \quad (2.2)$$

Figure 2.5 depicts example contentions of 0%, 25% and 50%. The low priority transaction starts delayed due to an ongoing high priority transaction, which is considered bus contention in our definition.

Note that the actual bus contention is also dependent of the timing accuracy of the model under test. A simulation of an abstract model, which is not timing accurate, may yield a different bus contention, than an accurate model. To compensate for this interdependency, we first measure actual bus contention for particular traffic pattern using an accurate model

⁶For practical purposes, we use a two step approach. We first implement and validate the BFM according to the bus standard. Then, we use the BFM to obtain the reference timing.

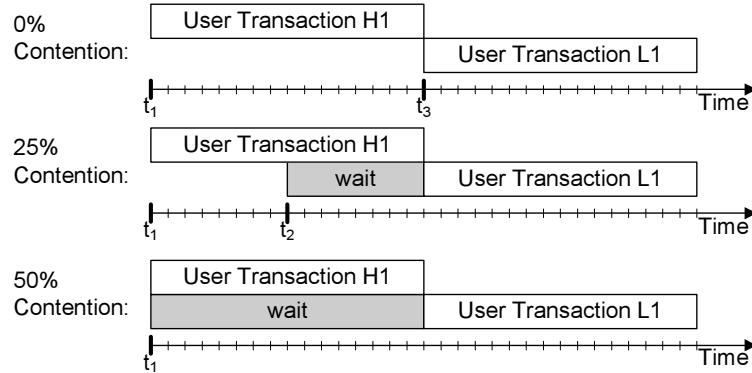


Figure 2.5: Bus contention.

(i.e. the BFM). Later, we use the identical traffic pattern for analyzing the abstract model and correlate the results to the earlier measured (accurate) contention.

2.3.2.3 Accuracy Measurement Setup

The actual timing accuracy highly depends on the test setup and the communication patterns of the involved applications. Therefore, we define a generic test setup and a procedure that covers a range of applications, so that the designer can derive the expected accuracy for her/his particular setup.

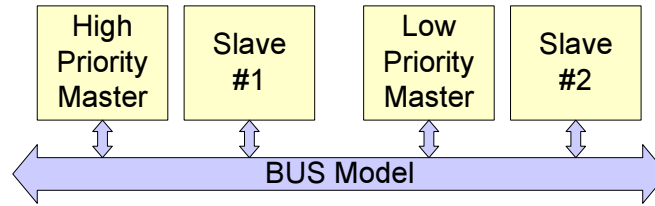


Figure 2.6: Dual master setup for accuracy measurements.

We use a generic test setup with two masters and two slaves connected to the same bus, as shown in Figure 2.6. Each master transfers a predefined set of 5000 user transactions to a slave target. Each transaction varies linear randomly in the base address, size (1 to 100 bytes), content⁷, and the delay to the next transaction. This delay simulates the application's computation. It ranges from a zero delay to a maximum delay value that depends on the

⁷ Please note that for some protocols, e.g. the CAN protocol, the content of the transaction can influence the transaction duration.

desired contention (see discussion below). The list of all transactions is created before the actual test by using a random number generator with a fixed start seed.

During the simulation, the start time and the duration is recorded for each individual user transaction and each master. The test is repeated for each implemented bus model. Since each model transfers the same set of user transactions, the results are comparable and can be analyzed together.

We have repeated the described test for different levels of bus contention. Since the bus contention cannot be controlled directly, we have varied instead the maximum delay between user transactions for each test run. This results in a varying bus utilization per master. Since the two masters share the same bus in our setup, the utilization correlates to the amount of bus contention. We have measured the resulting amount of bus contention according to our previous definition using the BFM, by checking for each clock cycle whether one or two user transactions are active. After we have determined the maximum delay that produces a desired contention, we use the created transaction set for measuring the abstract models.

2.4 AMBA

The first bus system is an example for an on-chip bus system with a centralized arbitration scheme [88]. We use the Advanced Microprocessor Bus Architecture (AMBA) for our analysis.

2.4.1 Introduction

AMBA defined by ARM [4] is a widely used and industry accepted standard for an on-chip bus system. The AMBA standard contains a group of busses, which are used hierarchically, as shown in Figure 2.4.1. We focus on the Advanced High-performance Bus (AHB), a system bus designed for connecting high-speed components including ARM processors.

The AHB is a multi-master bus that operates on a single clock edge and employs a centralized arbitration scheme. High performance is achieved by a pipelined operation that

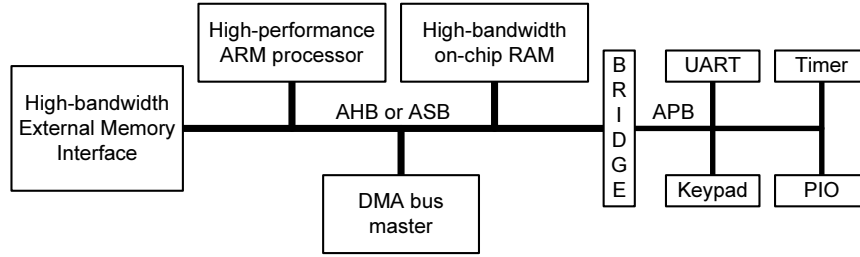


Figure 2.7: AMBA bus architecture (source [4]).

overlaps address and data phases, as well as by the usage of burst transfers. Split and retry transfers allow the slave to free the bus if the requested data is temporarily unavailable. The AHB uses a multiplexed interconnection scheme to avoid tri-state drivers.

2.4.1.1 Locked versus Unlocked Transfers

The AHB allows two operation modes, *locked* and *unlocked* transfers. These differ in whether preemption of a burst transfer is allowed. An example is shown in Figure 2.8.

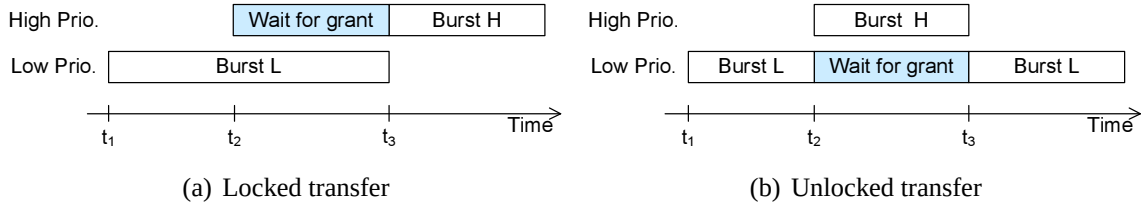


Figure 2.8: AMBA AHB operation modes.

In the locked mode in Figure 2.8(a), an ongoing burst cannot be preempted by a higher priority bus transaction. The high priority transaction released at t_2 is delayed until the current low priority bus transaction finishes at t_3 . In the unlocked mode in Figure 2.8(b), the high priority transaction preempts the ongoing low priority bus transaction and gains bus access already at t_2 . Since the operation mode influences the granularity of the arbitration decision, we will analyze these two operation modes separately.

2.4.2 Models

For modeling the AMBA AHB, we have applied the granularity-based approach described in Section 2.2.

Transaction Level Model (TLM) The TLM is the most abstract model that operates on the granularity of user transactions. It resolves contention independent of the priority arbitration using a semaphore once per user transaction, as described in Section 2.2.1. It provides basic support for bursts.

Arbitrated Transaction Level Model (ATLM) The ATLM operates on the bus transaction granularity, such as a StoreWord, StoreByte, LoadBurst. It adds support for priority-based arbitration and reflects the effects of pipelined operation. We have implemented two variants of the ATLM that differ in the time frame to collect arbitration requests. On an idle bus, the ATLM (a) collects requests for one clock cycle before making a decision, as required by the standard. The ATLM (b), on the other hand, makes the decision immediately after receiving the first request. Both variants behave identical when the bus is busy: requests are collected while a bus transaction is active, and the highest priority master continues after that.

Bus Functional Model (BFM) The BFM is the bus-cycle accurate and pin-accurate bus model. In order to correctly model the bus architecture, we have implemented several active components, such as multiplexers, an arbiter and an address decoder. On top of the features offered by the ATLM, the BFM supports unlocked (preemptable) transfers. At this point, our model does not support split and retry operations.

We have functionally validated all described models. We have also validated the cycle count timing of the BFM against the standard for all bus primitives, and have compared the waveforms with examples in [4, 5]. Finally, we have ensured that all abstract models show the correct timing in the single master setup.

2.4.3 Performance Analysis

Our performance measurements (Figure 2.9) confirm the TLM expectations: the simulation speed increases significantly with abstraction. The performance raises with each TLM abstraction by two orders of magnitude. However, no significant performance difference exists between the variants within the ATLMs. The additional abstraction of the

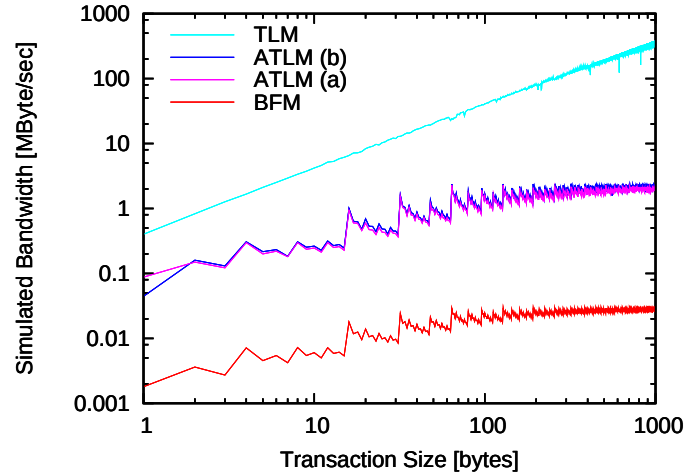


Figure 2.9: Performance for the AMBA AHB models.

(b) variant does not yield a significant speed improvement. Table 2.1 compares the performance in detail for transferring 512 bytes.

The TLM executes the fastest among the analyzed models. Its simulation bandwidth increases linearly with the transaction size, since a constant number of operations is executed for each transfer (one *memcpy* and one *wait-for-time*). 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. To give an example, transferring 3 bytes utilizes 2 bus transactions, a byte transfer and a short transfer, whereas 4 bytes are transferred in a single bus transaction, a word. Since each bus transaction is explicitly modeled, the simulation performance, and hence the simulation bandwidth, correlates with the number of bus transactions, and thus produces the saw tooth shape. 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).

	BFM	ATLM (a)	ATLM (b)	TLM
Simulation Time [ms]	16.75	0.2137	0.206	0.00246
Sim. Bandwidth [MByte/s]	0.03	2.29	2.37	198
Rel. Speedup over BFM	1	78	81	6802

Table 2.1: Performance comparison for transferring 512 bytes using AMBA AHB models.

2.4.4 Accuracy Analysis

The performance analysis for the AHB has shown the impressive speed benefits of TLM. Now we will evaluate the accuracy reduction, that the designer has to accept for achieving the higher simulation speed. As described before, we analyze *locked* and *unlocked transfers* separately.

2.4.4.1 Analysis for Locked Transfers

With locked transfers, a burst cannot be preempted by a higher priority master. Figure 2.10 shows the average timing error over a range of bus contention for the high priority master. 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 BFM and the ATLM (a) perform accurately over the whole range of bus contention (their graphs lie on top of the x-axis). Since the test setup uses only locked transfers, a burst is not preemptable and therefore no arbitration test is needed within a bus transaction. The features abstracted away in the ATLM (a) are not exercised. The ATLM (b), which makes an immediate decision and does not collect further requests, shows an inaccuracy of up to 18%. It may mispredict bus access when the two masters attempt a bus access within the same clock cycle.

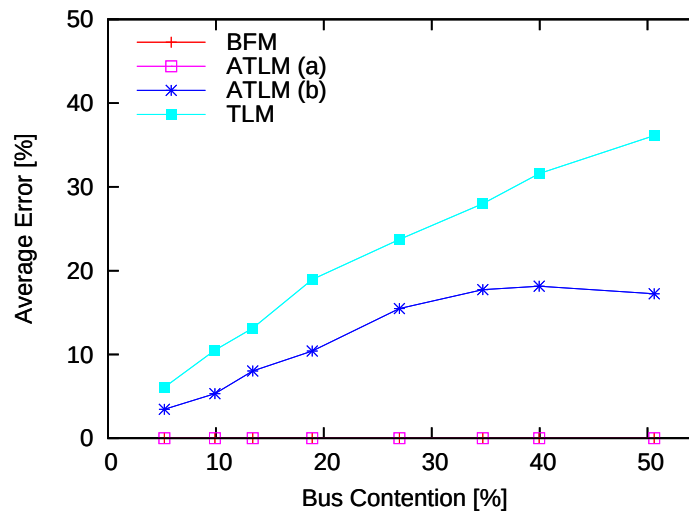


Figure 2.10: Individual timing accuracy of locked transfers for the AMBA AHB models.

On the other hand, the ATLM (b) is accurate, when the additional bus request arrives during an active bus transaction. Therefore, the error rate plateaus for bus contentions higher than 35%.

The TLM, which handles contention resolution on user transaction level with a semaphore, performs the least accurate due to the coarse grained decision independent of the master's priority. Its inaccuracy amounts up to 35%.

The measurements show very similar results for the low priority master. Hence, its graph is omitted for brevity, but can be found in [84].

To predict the delay of an application due to bus communication, we have also analyzed the same experimental data in terms of cumulative transfer time. Figure 2.11 shows the error in the cumulative duration for the high priority master using locked transfers⁸.

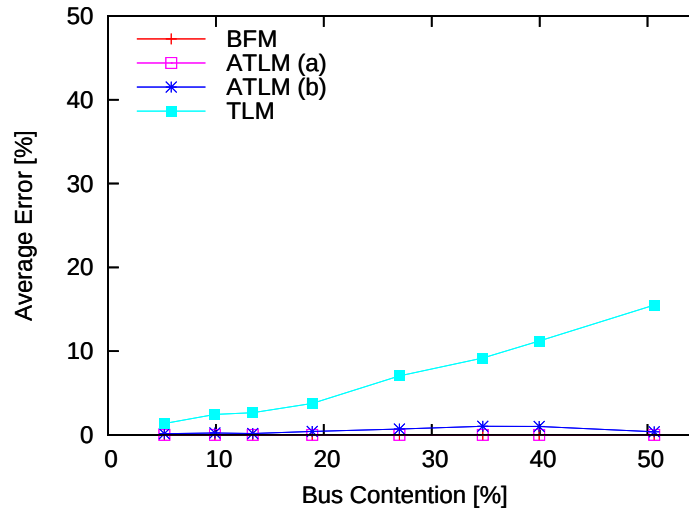


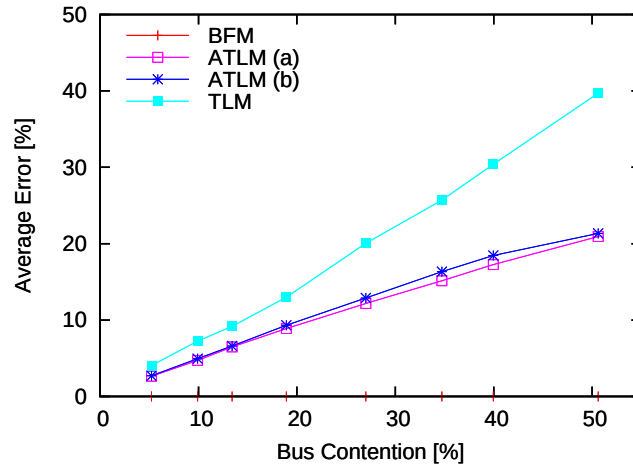
Figure 2.11: Cumulative timing accuracy of locked transfers for the AMBA AHB.

As in the previous graph, the lines for both BFM and ATLM (a) lie on top of the x-axis. The graph reveals that the mispredictions made by the less accurate ATLM (b) do average out. Both ATLM variants are good predictions for the application finish time. Only the mispredictions of the TLM do not average out, its error increases linearly to 15% for 50% bus contention.

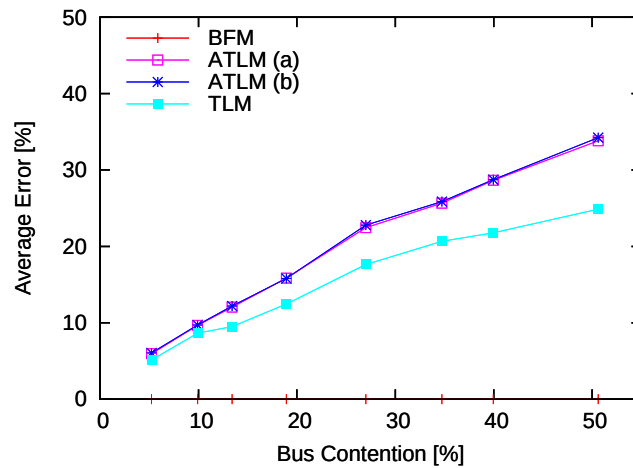
⁸ The graph for the low priority master is omitted, due to its similarity.

2.4.4.2 Analysis for Unlocked Transfers

We have repeated the same experiment for unlocked transfers. A burst may be pre-empted in this operation mode by a higher priority bus master, to be resumed later. As before, we have analyzed the accuracy for both: the individual transfer duration and the cumulative transfer time. Both analysis aspects yield similar results, therefore only one - the accuracy based on cumulative transfer time - is shown in Figure 2.12. Since the results differ by priority, the graphs for both the high priority master and the low priority master are shown.



(a) High priority master



(b) Low priority master

Figure 2.12: Cumulative timing accuracy for unlocked transfers for the AMBA AHB.

Figure 2.12 shows that only the BFM yields accurate results. With unlocked transfers, an arbitration decision is also necessary within a bus transaction. Therefore the ATLM (a) does now show an error of up to 35% for the low priority master. It handles arbitration only at the bus transaction granularity. The difference between the variants of the ATLM becomes insignificant. The ATLMs perform similar for the high and the low priority master.

The TLM can only give a very rough timing estimate. It exhibits a linear increasing error of up to 45% for the high priority master; the test requires arbitration handling for each bus cycle. The error is less for the lower priority master.

2.4.4.3 Error Distribution

Up to now our analysis has focused on the average of errors, which is applicable when predicting the overall system performance. Additionally, the error distribution may be of interest, to judge the range of errors to be expected during simulation. Figure 2.13 shows a histogram of the transaction durations normalized to the duration in the BFM. A value of 1 indicates an accurate prediction, with a value of 2 the abstract model simulates a transaction with twice the time of the BFM.

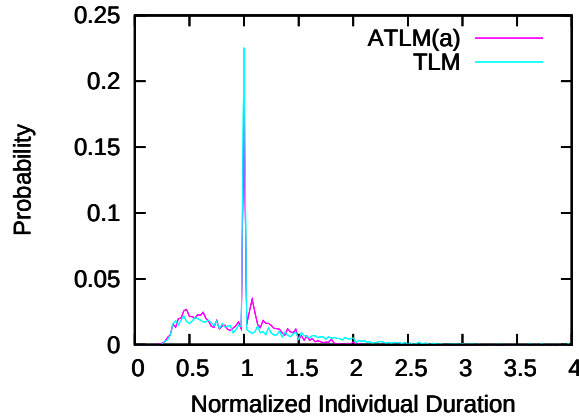


Figure 2.13: Histogram of normalized transaction duration.

For simplicity, we only compare the ATLM (a) and the TLM. Both are timing accurate for about 20% of the user transactions. The normalized duration of the ATLM range from 0.25 to 2, the results of the TLM are more wide spread, ranging from 0.25 up to 4. Thus, the ATLM has a tighter error bound.

2.4.5 Summary for the AMBA AHB

Figure 2.14 summarizes our analysis for the AMBA AHB and depicts the actual TLM trade-off. It shows the detailed relation between performance and accuracy. The x-axis denotes the performance in simulation bandwidth for transferring a 100 byte user transaction. The y-axis denotes the accuracy as the average error in individual timing at 40% bus contention for the low priority master.

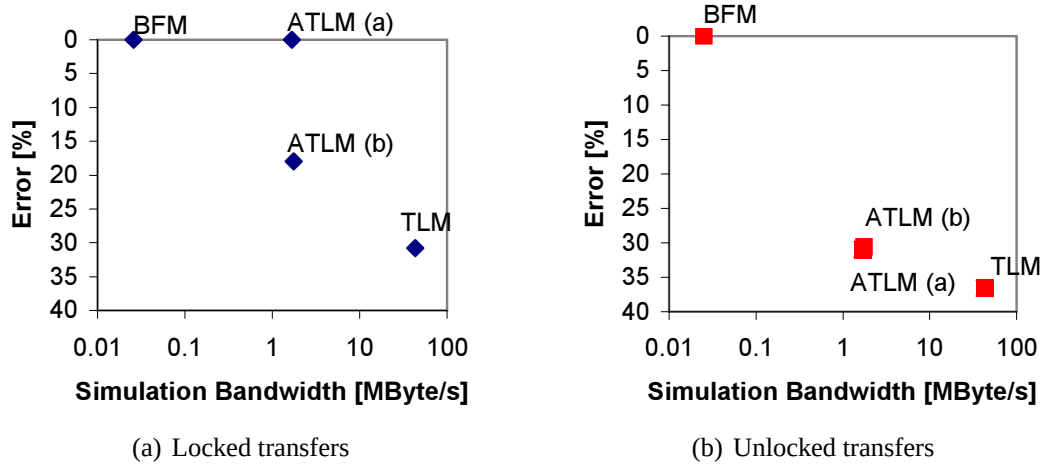


Figure 2.14: AMBA AHB TLM trade-off.

Comparing to Figure 2.1, the Figures 2.14 show the expected trade-off curve, with the exception of the ATLM (a) in Figure 2.14(a), which shows medium bandwidth at 0% error due to the locked transfer mode. Note that in Figure 2.14(b) the ATLM (a) does not exhibit this exception, lying directly over the ATLM (b).

In summary, our measurements of the AMBA AHB confirm the expectation of a significant speedup with abstraction. The ATLMS are two orders of magnitude faster than the BFM and simulate with more than 1 MByte/s. Decreasing the granularity further to user transactions, as done by the TLM, increases the speed for another two orders of magnitude.

Both variants of the ATLM show the same performance. This indicates that staying within the same granularity level, in this case bus transactions, usually leads to similar performance.

The increasing abstraction, however, also leads to a decreased timing accuracy. The BFM is the slowest model, but always delivers accurate results. The medium speed ATLM

model, is accurate for the locked transfers, but is inaccurate for the unlocked transfers (30% error). The fastest model, the TLM, also is most inaccurate (30% error for locked and 37% for unlocked transfers).

The ATLM (b) variant, which performs a reduced arbitration, is not efficient. Although it is not faster than the ATLM(a) it does produce inaccurate results even for the locked transfers (17% error).

Based on our performance and accuracy analysis, Table 2.2 lists the fastest model that yields acceptable results for a given situation and simulation focus.

Environment Condition	Model	Speedup
• single master bus • no bus contention	TLM	10^4
• only locked transfers • unlocked transfers, low contention	ATLM	10^2
• unlocked transfers, high contention	BFM	10^0

Table 2.2: AMBA AHB model selection.

All our AHB models are accurate in case of zero bus contention. Therefore, the TLM should be chosen in an architecture with a single master, or when no bus contention is expected. Then, it delivers accurate results the fastest. A system that only uses locked transfers can be accurately simulated by the ATLM. Its results are also acceptable for unlocked transfers in systems with a low bus contention. However, when simulating unlocked transfers under high bus contention only the BFM yields accurate results.

We have modeled the AMBA AHB as a representative for the category of on-chip parallel busses with centralized arbitration. We expect that our results are a good indicator for the behavior of models of other busses in this category, e.g. the IBM CoreConnect Processor Local Bus [48].

2.5 CAN

Our second bus example [83] falls into the category of an off-chip serial bus with decentralized arbitration. The Controller Area Network (CAN) is a serial communication protocol introduced by the Robert Bosch GmbH [11], that was designed for automotive applications.

2.5.1 Introduction

CAN is a serial multi-master 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 or not to process a frame. The frame identifier also serves as a priority. If multiple senders simultaneously attempt a transmission, the collision free CS-MA/CA arbitration will guarantee that the highest priority frame will succeed undisturbed.

After transmitting the start of frame bit (see Figure 2.15), the frame identifier is transmitted with the most significant bit first as a sequence of recessive (1) and dominant (0) bus states. During transmission, each sender compares the sent and received signal. If a sender has sent a recessive bit but detects a dominant bit, it will back off from transmission because another sender must have started a higher priority frame.

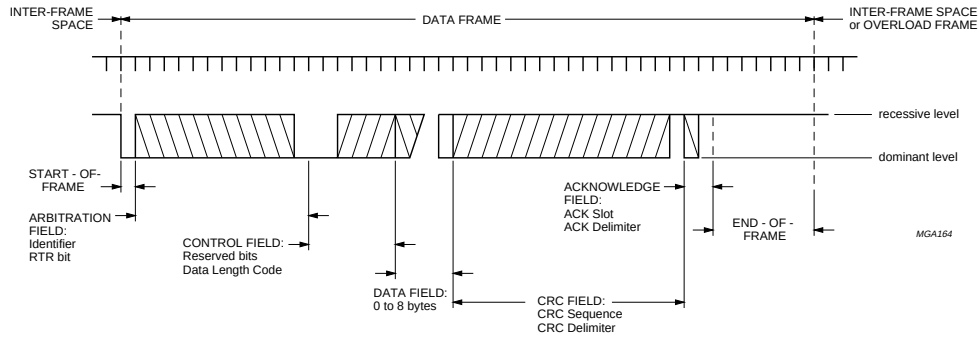


Figure 2.15: CAN data frame (source [76]).

In order to allow detection of corrupted data, each CAN frame includes a 15-bit Cyclic Redundancy Check (CRC). In case of a CRC mismatch, a retransmission of the frame is triggered. The protocol also defines elaborate error detection and error confinement rules for protection against faulty bus nodes.

The CAN serial protocol operates without a centralized clock. Each bus 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.

For modeling a CAN bus, 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

2.5.2 Models

We have applied our granularity-based abstraction to model the CAN. For each model, we have selected a subset of the above listed features.

Transaction Level Model The most abstract model is identical to the TLM of the AHB. Contention resolution is implemented on the user transaction granularity by using a semaphore (which ignores the frame identifier).

Arbitrated Transaction Level Model The ATLM simulates arbitration accurately for each bus transaction (CAN frame) based on the frame identifier. It collects all requests during the start of frame, and proceeds with the highest priority frame.

Again, we have defined two variants for the ATLM model. The ATLM (a) performs a bitwise inspection of the frame in order to calculate the CRC and handle stuff bits: a stuff bit is inserted/removed each time 5 bits of equal polarity are found. Note that due to the bit stuffing, the physical frame length depends on the frame content.

The ATLM (b) does neither calculate the CRC nor does it handle stuff bits. It avoids the costly bit inspection and is expected to execute faster than the ATLM (a), however at the cost of accuracy.

Bus Functional Model The BFM 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 nodes clock is synchronized to the bit stream according to [11] and [44].

Table 2.3 summarizes the features included in each model ("yes" indicates a modeled feature, "no" indicates a feature abstracted away).

Feature	BFM	ATLM (a)	ATLM (b)	TLM
Serial transmission, bit sync	yes	no	no	no
Error detection, confinement	yes	no	no	no
CRC calculation, bit stuffing	yes	yes	no	no
CSMA/CA arbitration	yes	yes	yes	no

Table 2.3: Summary of features captured in the CAN models.

2.5.3 Performance Analysis

As with the AMBA AHB, we analyze the CAN models in terms of performance and accuracy. The results of our performance measurements in terms of simulation bandwidth over an increasing user transaction size are shown in Figure 2.16. In addition, Table 2.4 compares the performance of the models in detail for a user transaction of 16 bytes.

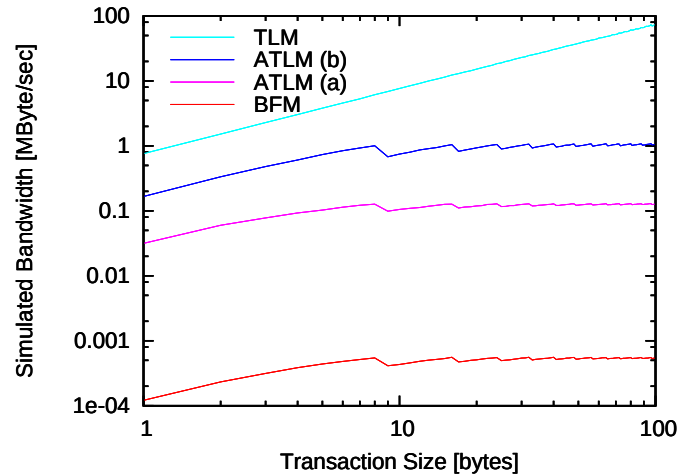


Figure 2.16: Performance of the CAN models.

Our performance measurements of the CAN show a similar dramatic increase in simulation speed as seen for the AHB. The TLM shows the highest simulation bandwidth, which increases linearly due to the constant number of operations. It achieves 12 MBytes/s when using 16 byte user transactions.

The ATLM (b) is the next slower model (12x). It does not model bit stuffing and CRC. Since the ATLM models individual bus transactions, a step is noticeable in the graph for each 8 bytes - an additional CAN message is needed for transferring the user data. The

execution time increases linearly with the amount of bus transactions, reaching a bandwidth of 1 MByte/s.

The ATLM (a) performs 8 times slower than the ATLM (b), since it inspects every bit of the message for the bit stuffing and the CRC calculation.

The BFM is another two orders of magnitude slower than the ATLM (a), due to the additional computational effort for the fine grained serial transmission and the bit synchronization. In addition, the structure of the implementation reduces the performance. For each bus node two extra threads of execution are required, one for the bit stream processor and one for the bit timing logic.

	BFM	ATLM (a)	ATLM (b)	TLM
Simulation Time [ms]	27.4	0.12	0.015	0.0012
Sim. Bandwidth [MByte/sec]	0.0006	0.127	1.05	12.3
Speedup over BFM	1	228	1879	22124
Rel. Speedup over previous	-	228	8	12

Table 2.4: Performance comparison for transferring 16 bytes using CAN models.

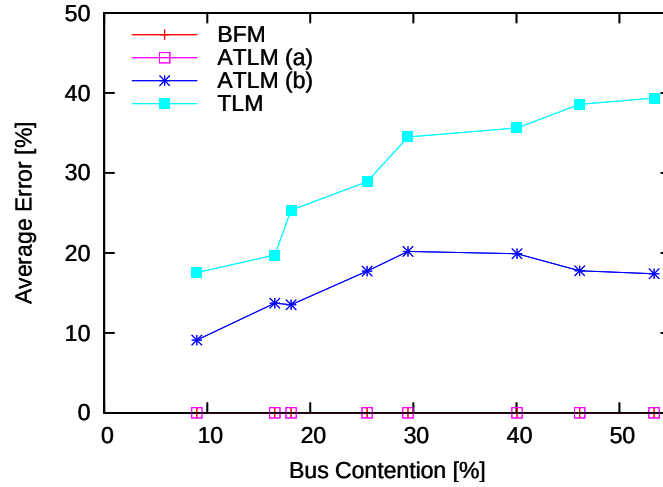
2.5.4 Accuracy Analysis

We use our generic dual master setup (Figure 2.6) for measuring the CAN model accuracy. However, CAN does not have a master/slave distinction, so we have modified our setup to two nodes acting as senders and two nodes acting as receivers. We let the user transactions vary in message id, length and content of the transaction (1 - 16 bytes), and in the delay between two transactions. Each sender uses an exclusive range of message identifiers. One will send messages with high priority identifiers (0-511), the other emits messages with low priority (identifiers 512-1023).

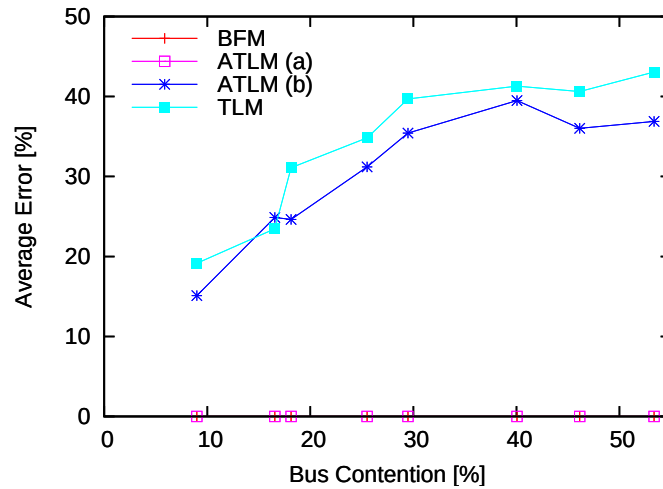
2.5.4.1 Analysis Based on Individual Transfer Duration

The first set of resulting graphs in Figure 2.17 show the average individual timing error over an increasing bus contention separately for high and low priority messages.

Figure 2.17(a) shows that the ATLM (a), which includes bit stuffing and CRC calculation, performs as accurate as the BFM (both graphs lie on top of the x-axis). This result has



(a) High priority messages.



(b) Low priority messages.

Figure 2.17: Individual timing accuracy for the CAN models.

to be interpreted with perspective to the restrictions of the test, which are: no propagation delay between sending and receiving on the CAN bus, all delays between user transactions are multiple of the CAN bit time and the test starts aligned to the bit clock of the first sender. With this setting (reasonable for a simulation environment only) all bus accesses are performed aligned to the CAN bit clock, and no sub-cycle information is needed. In this situation, the additional capabilities of the BFM, i.e. bit synchronization, are not exercised and both the BFM and the ATLM (a) are accurate.

The ATLM (b), due to the missing bit stuffing and CRC, performs inaccurately. For

messages in the high priority range, the inaccuracy starts with 10% for low contention situations and, after linearly rising to 20% inaccuracy, plateaus at 30% contention. Without the bit stuffing modeling, an individual message transfer is - depending on its content - shorter than in the bus functional model. Therefore, the arbitration interaction between the two senders differs. With an increasing contention the user transactions of the low priority band increasingly influence the high priority transactions. However an earlier started low priority transaction, which may consist of multiple CAN frames, can delay a later started high priority user transaction only for up to one frame. A second started CAN frame of the low priority transaction will lose arbitration, which leads to the plateau in inaccuracy at 30%. Looking at the same scenario with reversed priorities, this limitation does not apply. A low priority user transaction may be delayed for a full high priority transaction consisting of many CAN frames. Hence, the timing error of the ATLM (b) increases without a plateau for the low priority user transactions (Figure 2.17(b)) with increasing contention.

The TLM yields uniform results for both the high and low priority sender. For both cases, the inaccuracy increases with the bus contention. As expected, the TLM suffers the most in loss of accuracy (40% inaccuracy at 45% contention).

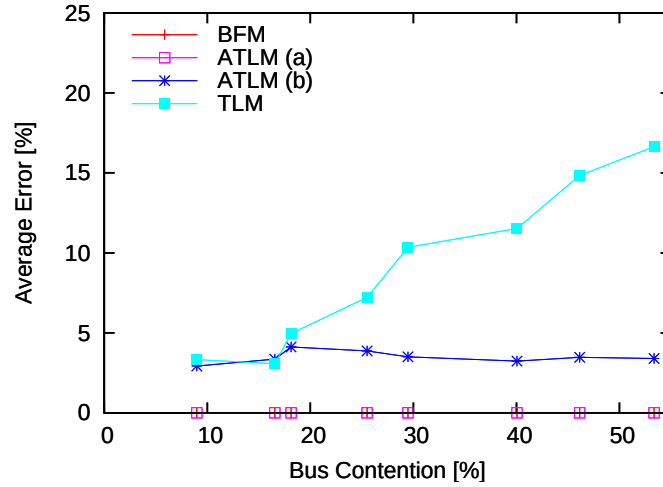
2.5.4.2 Analysis Based on Cumulative Transfer Duration

Figure 2.18(a) and Figure 2.18(b) show the accuracy results based on cumulative transfer time.

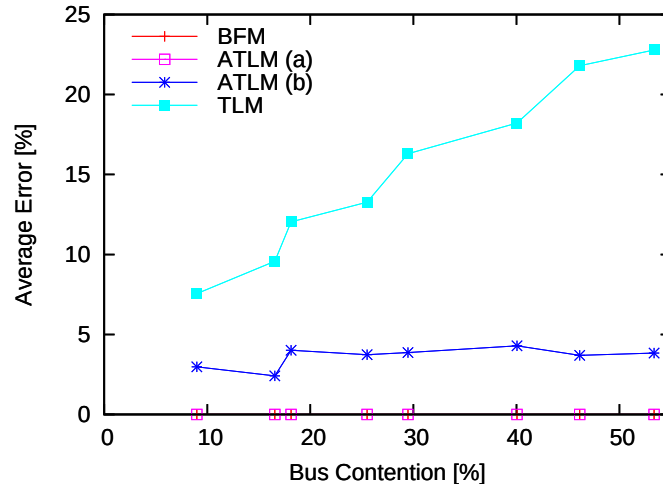
The analysis reveals that mispredictions of the ATLM (b) for individual CAN frames average out during the test, since the model correctly captures arbitration. Regardless of priority, a constant error of about 4% is measured. This can be attributed to not modeling the bit stuffing which in average adds 4% bits. The TLM, with its coarse grain priority-independent contention resolution, shows for both priority ranges a linear increasing error.

2.5.5 Summary for the CAN

Based on our analysis, Table 2.5 lists the fastest model that yields acceptable results for a given situation and simulation focus.



(a) high priority messages.



(b) low priority messages

Figure 2.18: Cumulative timing accuracy for the CAN models.

The TLM can only be used in very early stages of the design. Its accuracy, for individual and cumulative transfer time, degrades drastically with increasing bus contention. The ATLM (b) is still fast. It is applicable in scenarios where the main focus is on the application finish time. However, this model is not suitable for predicting an individual transfer delay, since the duration-based analysis has not shown acceptable results.

The ATLM (a), which includes bit stuffing and CRC calculation, has shown 100% accuracy given the test restrictions (e.g. no propagation delay). It is the fastest model that accurately predicts the delay of an individual transfer in all contention situations. The BFM

Environment Condition	Applicable Model
• No overlap between masters bus access • Early stage in design	TLM
• Main focus on application finish time	ATLM (b)
• Main focus on individual transfer delay	ATLM (a)
• Synthesizable • Using propagation delay	BFM

Table 2.5: CAN model selection.

is necessary as a synthesizable model, or in case the simulation includes propagation delay on the simulated CAN bus.

Figure 2.19 depicts the TLM trade-off for the CAN using the average individual timing error of the lower priority node. The accurate models BFM and ATLM (a) are slow with a bandwidth of less than 1 MByte/s. The faster models yield inaccurate results, the ATLM (b) with 40% error. The fastest model, the TLM with close to 100 MByte/s, is also the most inaccurate with 43% error.

We classify the CAN as an example for an off-chip serial bus with decentralized arbitration. Its results can be used as an indicator for modeling other serial bus systems, such as I²C [77] and Ethernet.

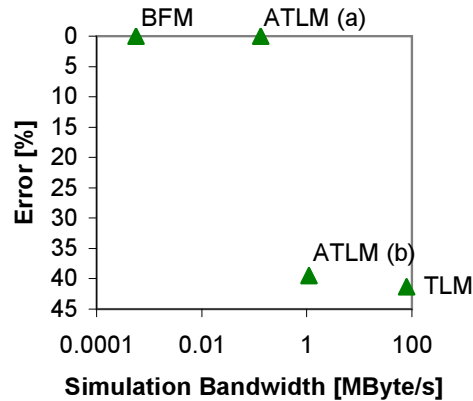


Figure 2.19: CAN TLM trade-off.

2.6 ColdFire Master Bus

Our third bus example, the ColdFire Master Bus (Version 2.0) [66], falls into the category of custom CPU busses. Typically, those busses are simpler than general standard busses, since they have been designed for the specific requirements of one embedded processor.

2.6.1 Introduction

Motorola (now Freescale Semiconductor Inc.) has introduced the ColdFire Master Bus for its popular microprocessor ColdFire MCF5206. It is a 32 bit tristate bus that uses basic transfers (byte, short, word), as well as a 4 beat burst to fill or store a cache line. It has to be noted that the line access (like a 4-beat burst) cannot be preempted. Therefore, there is no distinction between locked and unlocked transfers, as we have seen for the AMBA AHB.

The ColdFire Master Bus allows the usage of two arbitration schemes. One, the three wire mode, involves an external arbiter and allows a true multi-master operation. The second option, the two wire mode, connects two masters that directly hand over the bus without the need of an external arbiter, as shown in Figure 2.6.1. We have selected the operation mode with two masters to complement the centralized arbitration scheme used for the AMBA AHB.

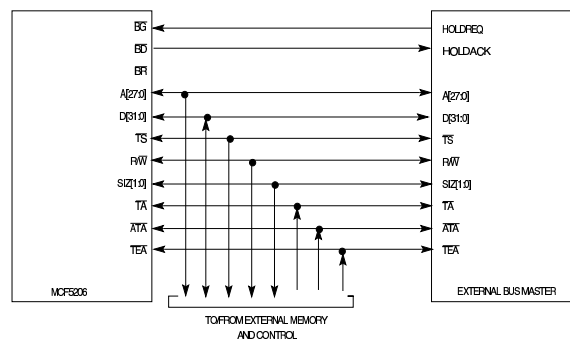


Figure 2.20: ColdFire Master Bus with two masters (source [66]).

2.6.2 Models

Our modeling of the ColdFire Master bus follows our scheme of abstracting data and arbitration, as described in Section 2.2. Again, we have implemented three major models: the BFM that implements a bus cycle access, the ATLM with a granularity of bus primitives, and the user transaction based TLM. We have not implemented any variants of these models.

2.6.3 Performance Analysis

We have measured the performance of the ColdFire Master bus in the setup with one master and one slave as described in Section 2.3.1. Figure 2.21 shows the simulation bandwidth.

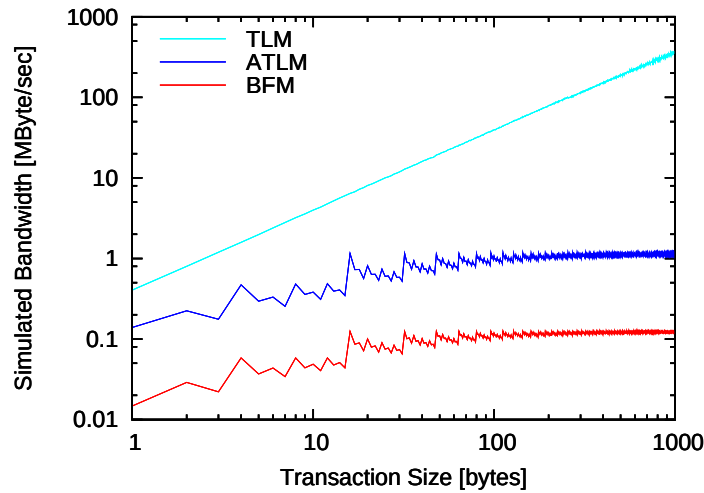


Figure 2.21: Performance of the ColdFire Master bus models.

Again, the measurements confirm the performance increase with abstraction. Also, the characteristic saw tooth shape, as seen already for the AHB, repeats. Due to the split of user transactions into bus transactions, the number of bus transactions does not increase linearly with an increase in size.

The BFM shows the lowest simulation bandwidth, since it models each individual bus cycle. The ATLM is one order of magnitude faster, due to the abstraction to bus transactions. The TLM is the fastest model. By simulating whole user transactions, it surpasses

the ATLM by two orders of magnitude. Table 2.6 lists detailed numerical results for transferring 512 bytes.

	BFM	ATLM	TLM
Simulation Time [ms]	3.93	0.423	0.0026
Sim. Bandwidth [MByte/s]	0.124	1.15	189.7
Speedup over BFM	1.0	9.3	1525.0
Rel. Speedup over previous	-	9.3	164.3

Table 2.6: Performance comparison for transferring 512 bytes using ColdFire Master bus models.

2.6.4 Accuracy Analysis

We have analyzed the timing accuracy in the setup with two concurrent masters as described in Section 2.3.2. As for the AHB, each master sends 5000 predefined user transactions that vary linear randomly in address, size (1 .. 100 bytes), and delay to the next transfer. Figure 2.22 depicts the accuracy of both masters over an increasing amount of bus contention.

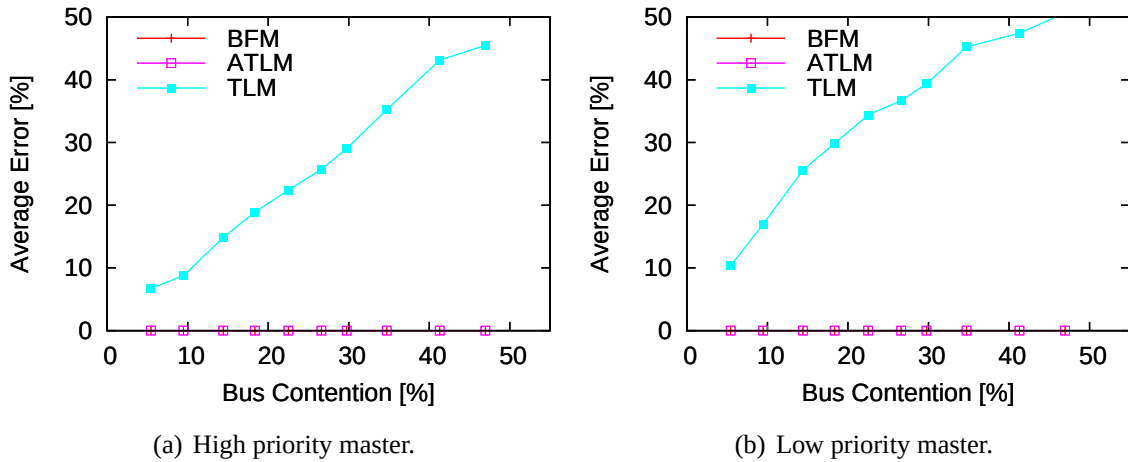


Figure 2.22: Individual timing accuracy for the ColdFire Master bus models.

As intended, the BFM complies with the standard and exhibits 0% error (its graph lies on top of the x-axis). The ATLM also reaches 100% accuracy. Since the ColdFire Master bus does not support preemption, the arbitration handling at the boundary of the

bus transaction is sufficient. In case the high priority master requests the bus during an ongoing burst of the low priority master, the high priority master has to wait until the burst completes and the low priority master releases the bus. The bus ownership changes only at the boundary of a bus transaction. Thus, the ATLM, which models this granularity, is accurate and its graph lies on top of the x-axis.

The TLM exhibits a linear increase in error with increasing bus contention, since it models the transfers at the coarser granularity of user transactions. At 45% bus contention, the TLM shows 54% error for the low priority master.

Analyzing the cumulative transfer times yields very similar results, so we omit these graphs. The TLM is the only model that exhibits an error, which reaches 35% at 45% bus contention.

2.6.5 Summary for the ColdFire Master Bus

With the Motorola ColdFire Master bus being simpler than the other protocols, its speedup with abstraction is not as dramatic. The most abstract TLM is 1525 times faster than the BFM. At the same time, the TLM is the only model that produces inaccurate results, with up to 54% error. The ATLM already simulates accurately, since the bus does not support burst preemption. However, the ATLM is only 9.3 times faster than the BFM.

Figure 2.23 shows the trade-off for the ColdFire Master bus. The accurate models BFM and ATLM are slow with a bandwidth below 1 MByte/s. The fast TLM, on the other hand, only yields inaccurate results with an average 47% error.

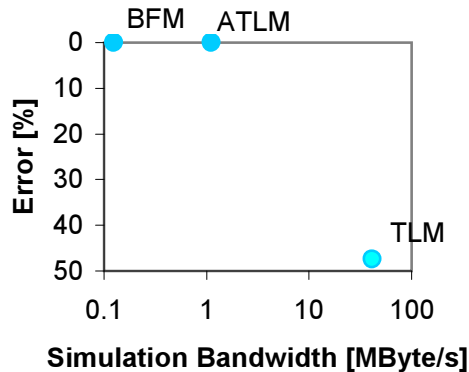


Figure 2.23: ColdFire Master bus TLM trade-off.

2.7 Generalization

Combining the performance and accuracy results of the three modeled bus systems, we will now generalize this data to give a broader perspective on the benefits and drawbacks of TLM. While, for an existing multitude of embedded bus architectures, an extrapolation based on three individual models is difficult in general, our data is a strong indication of generality.

As basis for our generalization, we have carefully selected three standard examples found in a majority of real-world embedded systems that exhibit very diverse communication properties. The AMBA AHB is a general-purpose on-chip pipelined parallel bus with central arbitration and a multiplexed interconnection scheme. In contrast, the CAN is a dedicated off-chip serial bus with distributed arbitration and a dominant-recessive interconnection. Yet a different candidate is the Motorola ColdFire Master Bus, a special-purpose embedded processor bus with custom arbitration and tri-state interconnection. Thus, these three chosen busses cover a wide variety of bus categories and can therefore serve as basis for a generalization with strong confidence.

Moreover, our granularity-based modeling approach is general and applicable to any layer-based bus model. Finally, we have observed the very same performance and accuracy patterns in our analysis for all three cases.

2.7.1 Performance

All our measurements confirm that TLM abstraction dramatically increases simulation performance (see Table 2.7). Decreasing granularity of user data handling and arbitration is a very efficient method of abstraction. For all models, we have measured a speedup of orders of magnitude with each step of coarser granularity. The cycle-accurate BFM performs slowly for all busses. The ATLTM, that simulates on a bus transaction granularity, is at least one order of magnitude faster. Further decreasing the granularity to user transactions, as done by the TLM, increases the speed by another two orders of magnitude.

On the other hand, variants at the same granularity level do not yield any significant speedup. We see this clearly in the ATLTM variants for the AMBA AHB. In other words,

Bus	ATLM (a)	ATLM (b)	TLM
AMBA AHB	78	81	6802
CAN	228	1879	22124
ColdFire	9.3	-	1525

Table 2.7: Speedup over bus functional model.

models within the same granularity category simulate at the same speed. It should be noted that we have observed the same fact also for variants at the TLM abstraction in previous work [88].

Only, if a feature requires a significant computation effort, then abstracting even at the same granularity level can improve performance. As an example, the ATLM (b) of the Controller Area Network (CAN) does not model bit inspection and simulates 8 times faster than its counterpart at the same level.

Our analysis also shows that the potential performance gain for a model depends on the complexity of the actual protocol. We have achieved the highest speedup of 22124x over the BFM for the complex CAN protocol. On the other hand, for the relative simple protocol of the custom CPU bus, the Motorola ColdFire Master bus, the TLM is only 1525 times faster.

2.7.2 Accuracy

Table 2.8 summarizes the results of our accuracy analysis. The accuracy achieved by a model depends on the modeling granularity and the actual granularity of the bus. By definition, each BFM is accurate, since it is the most fine-grained model. The ATLMs based on bus transactions are accurate if (and only if) the modeling granularity matches the actual granularity of the bus. The ATLM (a) for the AHB was accurate in the locked transfer mode as well as the ATLMs for the CAN and the ColdFire Master bus. Here, the actual bus protocol arbitrates once per bus transaction.

Inaccuracy has to be accepted when the model is more coarse grained. The ATLM (a) is inaccurate in the unlocked mode of the AMBA AHB, since bursts can be preempted and an arbitration decision within a bus transaction would be necessary. Furthermore, all our TLMs are inaccurate, due to the high abstraction.

Bus	BFM	ATLM (a)	ATLM (b)	TLM
AMBA AHB (locked)	0%	0%	18%	32%
AMBA AHB (unlocked)	0%	31%	31%	37%
CAN	0%	0%	39%	42%
ColdFire	0%	0%	-	47%

Table 2.8: Average individual timing error for the low priority master at 40% bus contention.

Abstracting away timing relevant features also leads to an inaccurate model, as shown for the CAN ATLM (b) that omits CRC calculation and bit stuffing. Also, the abbreviated arbitration of the ATLM (b) of the AMBA AHB yields an inaccurate model.

Finally, the actual measured inaccuracy of a model highly depends on the bus contention. The error of coarse grain arbitration modeling only takes effect if two masters access the bus simultaneously. In other words, all our models are accurate if there is no bus contention. Therefore, the most abstract model is the best choice if the architecture at hand only uses a single master, or if a very low bus contention can be expected. On the same note, modeling point-to-point dedicated links that inherently do not have any bus contention, can also be accurately modeled at the highest TLM abstraction.

2.7.3 TLM Trade-Off

Figure 2.24 summarizes our measurements and depicts the TLM trade-off for all our bus models. On the x-axis, Figure 2.24 denotes the simulation performance in terms of the simulation time for a 100 byte user transaction. The y-axis denotes the average error of the model for the low priority master at 40% bus contention.

Abstract modeling poses a trade-off, with either fast or accurate results. The accurate BFM's are slow with less than 0.2 MByte/s bandwidth. On the other hand, the fast TLM's with a bandwidth of up to 100 MByte/s produce an average error from 32% to 47%.

The ATLM's 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's generate an average error from 18% to 39%.

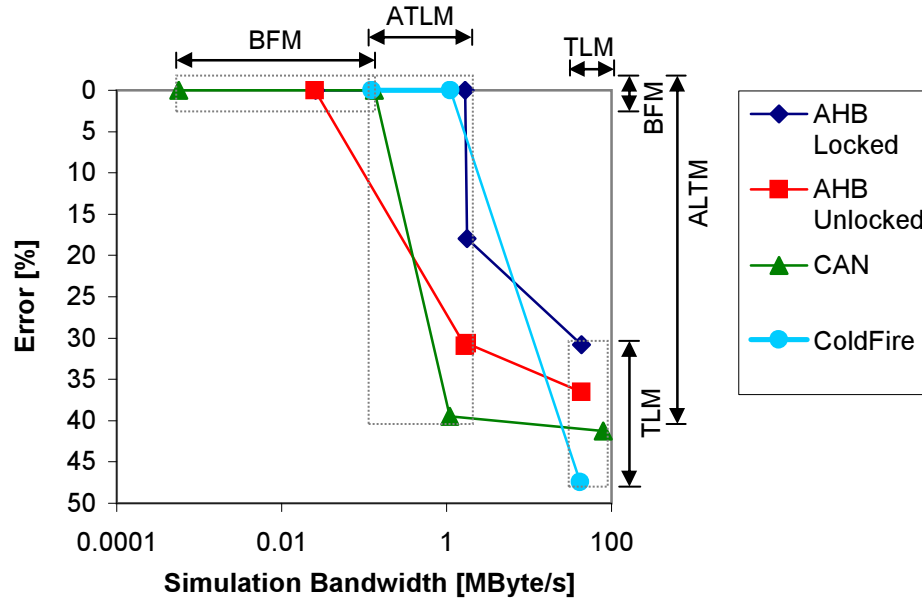


Figure 2.24: TLM trade-off summary.

2.8 Summary Transaction Level Modeling

Our contributions in this chapter (and in journal [91]) are fourfold. First, we have classified TLM according to the granularity of data and arbitration handling. Second, we have defined appropriate metrics and setups for a systematic quantitative analysis of TLM with respect to simulation performance and timing accuracy. Third, we have quantified the TLM trade-off for three diverse communication protocols. Fourth, we have generalized our observations from the analysis to guide communication model designers and model users.

In particular, we have applied our granularity-based abstraction to three common bus systems covering diverse communication protocols: first the AMBA AHB, as an on-chip parallel bus with centralized arbitration, second the CAN, as an off-chip decentralized serial bus system, and third the ColdFire Master bus, as a custom embedded processor bus. We have modeled, validated, and systematically analyzed each bus using our performance and accuracy metrics.

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 TLM abstraction.

However, TLM abstraction 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 TLM models 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.

The actual measured timing error highly depends on the bus contention. All our models are accurate in the absence of bus contention. Therefore, in the special case of a single master architecture, the most abstract TLM can be used without loss of accuracy.

In summary, this chapter contributes a systematic quantitative analysis of performance and accuracy in TLM, using a diverse set of major bus architecture standards, that confirms the TLM promise of high simulation speed. Our detailed analysis also identifies conditions for abstract and accurate models. As a result, we provide guidelines to the designer of communication models for efficiently abstracting communication protocols. The same guidelines also allow the system designer as a user of communication models to make an informed decision about the appropriate model for her/his design at hand.

Chapter 3

Result Oriented Modeling (ROM)

3.1 Introduction

Transaction Level Modeling (TLM) abstracts communication above the implementation detail level and uses timing annotations for simulating the time progress. TLMs execute dramatically faster (multiple orders of magnitude) than bit-accurate models. However, this benefit 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 we have analyzed in Chapter 2, accurate models achieved less than 0.2MBytes/s simulation bandwidth, while the TLMs exhibited up to 100MBytes/s, however at up to 47% error (see Figure 2.24).

The trade-off (we repeat its illustration in Figure 3.1) 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.

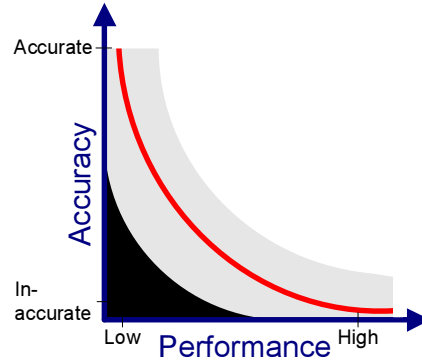


Figure 3.1: Transaction Level Modeling Trade-Off (repeated Figure 1.5).

3.1.1 Scope of Work

In this chapter, we introduce a novel modeling technique for TLM, called Result Oriented Modeling (ROM), which eliminates this TLM trade-off for a wide range of platforms [89]. For this work, we consider designs for which the response time of a node is statically known or computable. Examples of such platforms include cell phones, multimedia SoCs and automotive control systems [96]. Platforms for which a node's response time within a bus transaction cannot be predicted (e.g. random slave controlled wait cycles), are outside the scope of this dissertation.

ROM delivers simulation speeds similar to a TLM. At the same time, ROM retains 100% accuracy in timing for the considered range of platforms¹. As a result, the system designer is relieved from the traditional speed/accuracy trade-off and can focus on the design space exploration instead of model selection. Moreover, having a 100% accurate bus model at such a high simulation performance opens new applications of abstract communication modeling. For example, it allows for the first time, that a real-time system simulation can profit from the speed advantage of abstract communication modeling.

We will apply the ROM concept to the modeling of two busses from opposite ends of the spectrum with different characteristics. We have chosen the Advanced High-performance Bus (AHB) of AMBA ([4]), a parallel on-chip bus system with a centralized arbitration scheme, and the Controller Area Network (CAN) [11], a off-chip serial bus with distributed arbitration.

¹ ROM cannot guarantee timing accurate results for platforms outside of our scope, in which the node response time can not be calculated.

3.1.2 Overview

We first describe in Section 3.2 the general concepts of our novel ROM approach. Then, in Section 3.3, we explain in detail the application of ROM to communication modeling based on the two examples AMBA AHB and CAN. We validate our approach and show experimental results in Section 3.5. We then generalize the results in Section 3.6, and conclude the chapter in Section 3.7.

3.2 Result Oriented Modeling

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

3.2.1 Black Box Concept

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

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

3.2.2 Corrective Measures

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

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

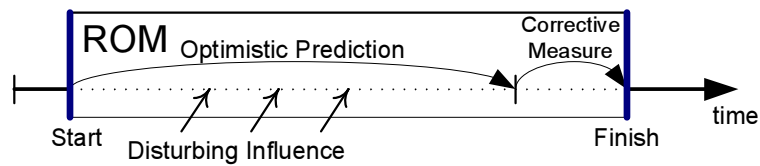


Figure 3.2: Generic ROM concept.

Generally speaking, the ROM approach can be characterized as shown in Figure 3.2 by the following items:

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

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

3.2.3 Example

Figure 3.3 illustrates the ROM approach using an example of predicting the arrival time of an airplane. The real process (a) exhibits continuous changes to the ground speed de-

pendent on the disturbing influence wind. The traditional abstract modeling approach TLM (b) approximates the result by incrementally calculating the ground 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 speed. Instead, it makes one initial optimistic prediction about the arrival time, and finally corrects its prediction retroactively for the average wind condition.

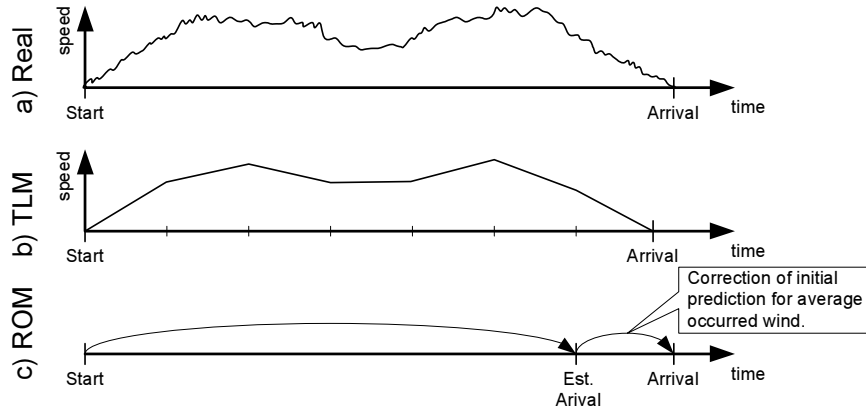


Figure 3.3: ROM predicting an airplane arrival time.

3.2.4 Analogy

As an analogy, our ROM approach can also be compared in a broader sense to optimistic timed co-simulation [110], a synchronization technique between two concurrent discrete event simulators. It allows each simulator to "run ahead" using internal events, before synchronizing with the other simulator. This minimizes the synchronization effort, in contrast to a lock-step synchronization approach. However, it requires a "roll-back" in case of early external events, which our ROM does not require. ROM uses an optimistic prediction approach, that estimates the earliest time to finish. Hence, a disturbing influence may only delay the time to finish, but cannot shorten it. Therefore, ROM does not require a "roll-back" mechanism and can be implemented on an unmodified discrete event simulator.

3.3 Communication Modeling using ROM

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

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

3.3.1 AMBA AHB - Traditional Modeling

The AMBA defined by ARM [4] is a widely used and industry accepted standard for an on-chip bus system. We focus on the Advanced High-performance Bus (AHB), a system bus designed for connecting high-speed components including ARM processors.

Following the ISO OSI reference model [50], we model the AHB using a layered architecture as described in Section 2.2. Important for this discussion is the granularity of data handling in each of the layers. The *media access layer* provides a transmission service for a contiguous block of bytes, called a *user transaction*. This layer divides the arbitrarily sized user transaction into smaller bus transactions observing the bus addressing rules, and transfers these byte blocks using the protocol layer. The *protocol layer* transfers data as *bus transactions*, which are bus primitives (e.g. bytes, words, or 4 word bursts). It uses the services of the *physical layer* which provides a *bus cycle* access to sample and drive individual bus wires.

Figure 3.4 (similar to Figure 2.2) 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 define three models which we will refer to as TLM, ATLM, and BFM.

Transaction Level Model (TLM). The TLM is the most abstract model, implementing only the media access layer. Data is handled at the user transaction granularity and

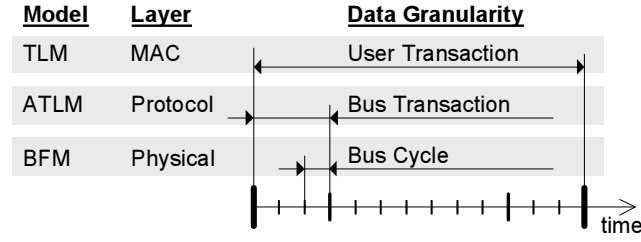


Figure 3.4: Layer-based Bus Modeling.

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. Arbitration is abstracted to a semaphore used once per user transaction.

ATLM. The ATLM models a bus access with AHB bus primitives 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 priority-based arbitration, however only once for each bus transaction. This model is not pin-accurate and not in all cases cycle-accurate.

BFM. The BFM is a cycle- and pin-accurate bus model. It implements all layers down to the physical layer and covers all timing and functional properties of the bus definition. The BFM handles arbitration per bus transaction and verifies the bus grant on each cycle of a burst. Additional active components, such as multiplexers, an arbiter and an address generator are needed to accurately model the bus architecture.

3.3.1.1 Limitations of layer-based models

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

Figure 3.5 shows the arbitration check points as check marks for the models. A user transaction of 16 words is transferred in two 8-beat bursts. The BFM performs full arbitra-

tion at the beginning of each bus transaction and also verifies the arbitration at each cycle. The ATLM checks arbitration only at the bus transaction boundaries. The TLM performs the least amount of checking. It arbitrates only at the beginning of a user transaction.

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

3.3.1.2 Static vs. Dynamic Delay Model

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

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

The dynamic delay model becomes apparent for a preemption of the transfer, as shown

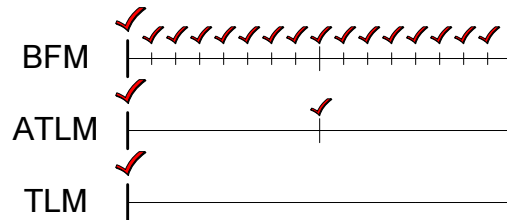


Figure 3.5: Arbitration check points when transferring two 8-beat bursts.

in Figure 3.6. Despite the fact that the model initially represented a transaction using a single *wait-for-time* statement, it can accommodate preemption on a cycle-by-cycle basis. For that, it records any disturbing influence during the delay time, recalculates later the transaction time using the updated system knowledge, and executes an additional *wait-for-time* statement. The process repeats in case a another preemption occurs in the updated time period, until no further preemptions are detected. As a consequence, the number of *wait-for-time* statements may vary when re-executing the same transaction. In the optimal case without preemption, ROM will execute a single *wait-for-time* statement, same as the TLM.

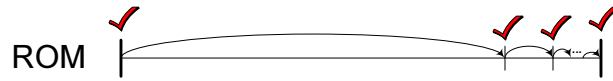


Figure 3.6: Arbitration check points in ROM.

3.3.2 AMBA AHB - Result Oriented Modeling

We will now apply the ROM approach to the AMBA AHB [87]. Unlike the traditional models, ROM does not exhibit the correlation between granularity and abstraction. It avoids unnecessary arbitration check points. As a result, it reaches the BFM accuracy of 100% and near TLM performance.

3.3.2.1 Assumptions as with TLM

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

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

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

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

As in TLM, the main idea 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 possible context switches.

3.3.2.2 Optimistic modeling

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

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

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

3.3.2.3 Preemption

To compare ROM against the layered models, we will analyze the case of a bus preemption in more detail, as shown in Figure 3.7.

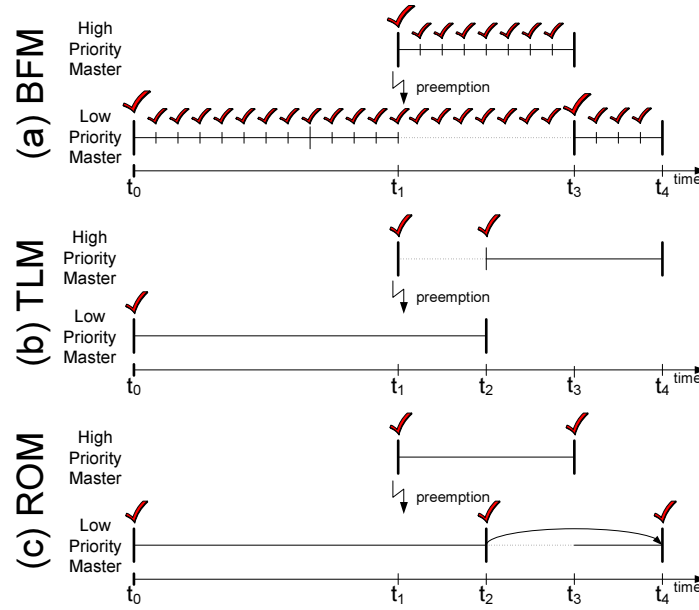


Figure 3.7: Preemption in BFM, TLM, ROM.

Figure 3.7 shows the bus activity of a system with two processing elements that are connected as masters to the AMBA AHB; one with high priority (referred to as high priority master), and one with low priority (referred to as low priority master). Each processing element has an own flow of execution, and communicates via the master interface through an abstract bus model; (a) BFM, (b) TLM, and (c) ROM. Depending on the functional simulation inside the processing element, transactions are issued to the bus.

In Figure 3.7, (a) BFM shows the reference bus activity. A burst transaction issued by the low priority master, which starts at t_0 , is preempted by the high priority master at t_1 . The transfer of the high priority master completes at t_3 . At that time, the preempted low priority transfer resumes and terminates finally at t_4 . In the simulation, both masters perform arbitration checks for every bus cycle. In total, the model executes 32 arbitration checks for both masters in this example.

In (b) TLM², the transaction of the low priority master is modeled with a single wait-for-time statement, starting at t_0 and finishing at t_2 . Therefore, the preemption triggered at t_1 can not be taken into account at that point, and is delayed instead until t_2 . As a result, the low priority transaction is not properly preempted and still ends at t_2 (not at t_4).

² For brevity, we will omit the similar ATLM case here.

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. On the other hand, only three arbitration checks are performed, promising a high simulation performance.

In (c) ROM, the inaccuracies of the TLM are corrected by 2 additional arbitration checks. ROM requires in total only 5 arbitration checks and 3 wait-for-time statements.

At t_0 , the low priority transfer is initially predicted to finish at t_2 , and hence the low priority master executes a wait-for-time delaying until then. At t_1 however, the high priority master preempts the ongoing transfer, predicts for its own transfer a finish time of t_3 , and records the disturbing influence for the low priority master. Note, the low priority master is not woken at this time (t_1) for participation in the arbitration, instead it remains sleeping on the initial wait-for-time statement.

At t_2 , the low priority finishes the initial wait-for-time statement, detects that it has been preempted at t_1 and subsequently recalculates its finish time. Taking the preemption into account, the updated finish time is now $t_3 - t_1$ time units later at t_4 . Hence, the low priority master executes a wait-for-time statement until t_4 .

The high priority master wakes up at t_3 , checks its own preemption records and finishes its transaction, since it was not preempted. At t_4 , the low priority master wakes up, verifies that no other preemption has occurred, and thus completes its transfer.

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

Table 3.1 compares the arbitration checks performed by the models. The number of checks in ROM is close to the TLM case and an order of magnitude lower than in the BFM.

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

Table 3.1: Preemption complexity comparison.

3.3.2.4 Pipelining

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

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

3.3.3 CAN - Traditional Modeling

The second bus example, CAN, is an off-chip serial bus with decentralized arbitration (see Section 2.5.1), very different from the earlier described AMBA AHB. In the following sections, we will touch briefly on the layer based models and describe the ROM for the CAN wherever it is different from the AMBA AHB implementation. The main focus will then rest on the analysis of the ROM benefits.

We have applied the same layer-based approach as in Section 3.3.1 for modeling the CAN, using the same three abstraction levels: the BFM, ATLM and the TLM. The most abstract model, the TLM, is identical to the AHB implementation and simulates on a user transaction granularity. The ATLM, simulating individual CAN frames, already achieves accurate arbitration based on the CAN frame identifier. It also performs a bitwise inspection of each frame for CRC calculation and stuff bit handling. The BFM is our most detailed model implementing all features of the specification. It serially sends and receives the data, and synchronizes each node's clock to the bit stream according to [11] and [44].

3.3.4 CAN - Result Oriented Modeling

The basic concepts of ROM for the CAN are identical to ROM of the AMBA bus. ROM abstracts to the level of a user transaction, combining multiple CAN frames using a single *memcpy* and minimizing the number of potentially costly *wait-for-time* statements. As described for the general case, the CAN ROM relies on the limited observability for rearranging internal events [86]. The application is unaware of splitting the user transaction into individual CAN frames and CSMA/CA arbitration.

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

In comparison to the AMBA AHB ROM implementation, additional computation effort is needed for accurately simulating CAN. CAN employs a bit stuffing algorithm to guarantee sufficient edges in the raw data stream for synchronizing the local clocks from the shared serial bus. Due to that bit stuffing rule, the number of physical bits needed for transmitting a CAN frame not only depends on user data length, but also depends on the actual content of the message. In addition, the bit stuffing rule also extends to the CRC field of a CAN frame. Therefore, ROM has to compute the CRC for each CAN frame, and has to perform a bit inspection of each frame, calculating the number of stuff bits that would be inserted. As we will see later, the additional effort for the bit inspection prevents ROM from outperforming TLM.

In order to guarantee accurate timing, ROM validates 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 transmission. Then, the initially predicted time will be too short. ROM detects this *disturbing influence* at the predicted end time of the low priority message. It then recalculates the predicted time and waits for the remaining portion. This process repeats until no further disturbing influence has been detected, thus the prediction has deemed correct. To minimize the computation effort for an update, ROM stores the physical length of each individual CAN frame during the initial prediction and thus avoids the costly bit inspection during an update.

To illustrate the ROM approach in contrast to the layered models, we analyze an exam-

ple with a disturbing influence as shown in Figure 3.8. A low priority *msg2* with 4 frames starts at t_0 . During transmission of the its second frame, at t_1 , a high priority *msg1* is released. This delays sending the third frame of *msg2*, thus delays completion of the overall *msg2* transmission. In the example, we depict *wait-for-time* statements as arcs, which gives a visual indication for estimating a model's performance.

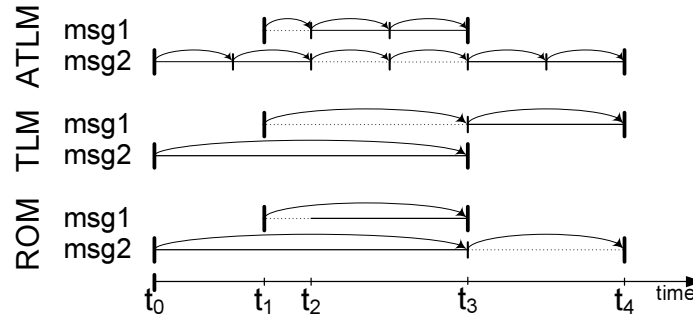


Figure 3.8: Contention in ATLM, TLM and ROM

In **ATLM**³, the transmission delay is correctly modeled. Since a CAN frame itself is not preemptable, modeling at the CAN frame granularity is sufficient (see also Section 2.5.4). The transfer of the higher priority message *msg1*, released at t_1 , is delayed until the end of the currently running CAN frame. Then, at t_2 , a new CAN frame starts and the first frame of the higher priority *msg1* wins the arbitration. The sender for *msg2*, on the other hand, loses arbitration for its 3rd frame at t_2 . It retries twice until t_3 , when *msg1* finishes. Then, the remaining two frames of *msg2* are transmitted until t_4 . In total, the sending nodes execute 9 *wait-for-time* statements.

In **TLM**, the example transfer is not accurately simulated due to the coarse granularity. Since a TLM models a whole message as one *wait-for-time*, the ongoing transfer of *msg2* cannot be interrupted and thus ends at t_3 (not at t_4). Only after that transfers of *msg1* starts and thus finishes too late at t_4 . Clearly, the abstract TLM is highly inaccurate in the finish times of both transfers, however it will execute fast with only 3 *wait-for-time* statements.

In **ROM**, the low priority *msg2* is initially predicted to finish at t_3 . This prediction assumes immediate availability of the bus, as no other transaction is scheduled at that time. At t_1 , the high priority *msg1* is released. Since the bus is currently occupied, it is predicted

³For brevity, we omit the BFM case here as the ATLM is already timing accurate. The BFM, with modeling the serial transmission, executes 11,888 *wait-for-time* statements.

to start after the currently transmitting frame of *msg2*, at t_2 . *Msg2* is predicted to finish at t_3 . At that time (t_3), the node sending the high priority *msg1* wakes up and terminates since no disturbing influence has occurred. At the same time, the node sending the low priority *msg2* wakes up and detects the disturbing influence of *msg1*. Thus, it then updates the predicted finish time for *msg2* to finish at t_4 , which is $t_3 - t_2$ time units later, and subsequently waits for the remaining time. When the sender of the low priority message wakes up again at t_4 , it detects no other disturbing influence and completes its transfer. Overall, ROM is able to correctly simulate the example with only 3 *wait-for-time* statements.

Table 3.2 compares the *wait-for-time* statements performed by the CAN models. ROM waits as often as the TLM. They both wait four orders of magnitude less frequently than the BFM, which simulates individual clock cycles on the serial bus.

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

Table 3.2: Wait complexity with disturbance.

3.4 ROM Algorithm

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

Algorithm 1 shows the characteristic elements of ROM: the initial prediction of the transaction time (*waitTime*, line 1), waiting for the predicted time (line 4) and updating the prediction for any disturbing influence (line 5). *GetDelayDueToPreemptions()* examines past disturbing influence and returns the accumulated delay of preempting transactions. The loop repeats the waiting until no further preemptions have occurred. After no further preemptions have been detected, the transaction finishes by copying the user data (line 7).

A global *List*, sorted by priority and transaction start time, keeps track of the scheduled transactions. Each transaction is added to that *List* at the beginning (line 2) and removed

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

```

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

```

upon completion (line 8). Using this *List*, the schedule of bus for each clock cycle can be reconstructed.

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

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

GetDelayDueToPreemptions(), which is called in Algorithm 1, line 5, is similar in principle to *PredictWaitTime()*. *GetDelayDueToPreemptions()* examines, once a transaction is

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

```

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

```

marked as preempted, the list of preempting transactions and recalculates the bus schedule, taking into account all currently scheduled transactions. It determines start and duration of the remaining portion of the preempted transfer. Based on this information, it then calculates the additional delay due to the occurred preemptions.

3.5 Experimental Results

To validate the benefits of our proposed ROM approach, we have implemented for both busses all four models using the SpecC [29] SLDL. Please note that we implemented ROM without any modification to the simulation engine, using only regular *wait-for-time* statements and events.

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

3.5.1 Accuracy

For analyzing the accuracy of our models, we use the metrics and setup described in Section 2.3.2. In a multi node setup such as depicted in Figure 3.9, each node sends a set of 5000 predefined user transactions that vary in user transaction id (each from its own range), size, content and delay between two transfers. The same set of transactions is transferred by each model. We repeat each test for different amounts of bus contention. During the test, we record the duration for each individual transfer and compare that later against the cycle-accurate BFM.

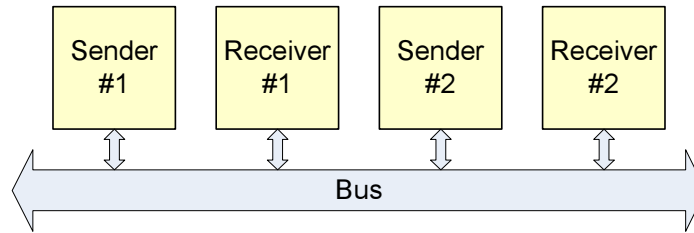


Figure 3.9: Multi-node setup.

3.5.1.1 AMBA AHB

We have measured the accuracy of the AMBA AHB models in a two master setup. Figure 3.10 shows the average error in transaction duration for the high priority master 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). In contrast, the TLM and ATLM show significant error rates, linear increasing with growing bus contention. At 45% contention, the TLM reaches 45% error, making any system timing analysis based on TLM questionable. Please see Section 2.4.4 for a more detailed analysis and discussion about the timing error of the layer-based AMBA AHB models.

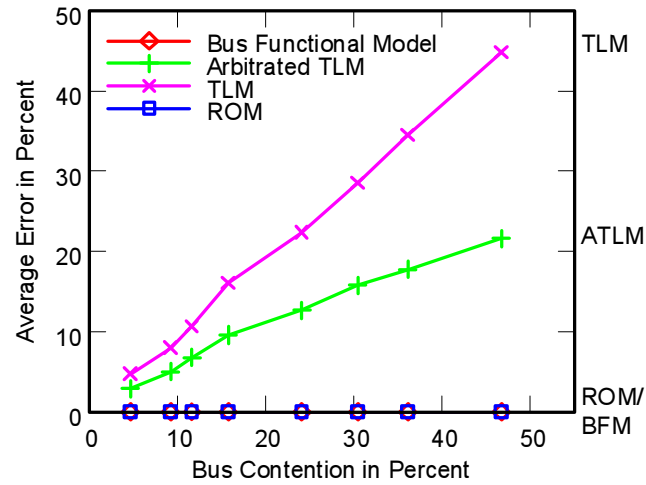


Figure 3.10: Accuracy of the AMBA AHB models.

3.5.1.2 CAN

We use a test setup with 8 nodes⁴ connected to the CAN bus. Four nodes act as senders and four nodes as receivers. Each sender uses an exclusive range of message identifiers (in effect, a different priority). Figure 3.11 shows the average error in transaction duration for the node sending the lowest priority messages over a varying degree of bus contention.

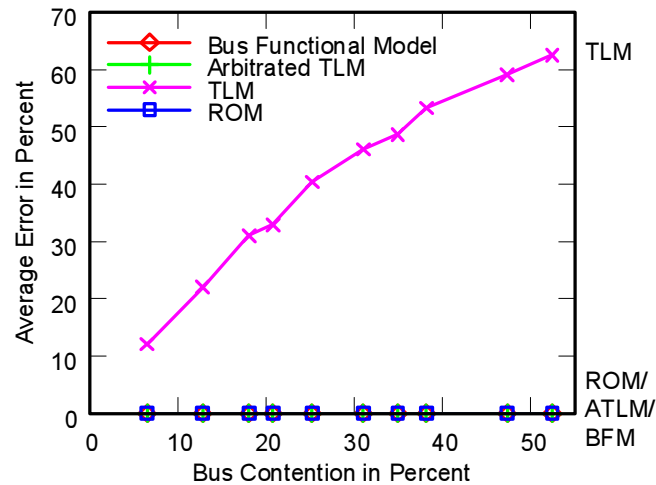


Figure 3.11: Accuracy of the CAN models.

For the CAN as well, ROM shows 0% error for all measurements. Its graph lies right on top of the x axis (same as the BFM and the ATLM). Note that in the CAN protocol

⁴ Other tests show similar results.

a CAN frame cannot be preempted, therefore the ATLM is already accurate in this case, since arbitrating once per frame matches the protocol definition. In contrast, the TLM shows significant error rates, linear increasing with growing bus contention, passing 60% error at 50% contention. This inaccuracy hinders using TLM in a context requiring accurate simulation of communication, such as in a real-time system. Please see Section 2.5.4 for a more detailed analysis and discussion about the timing error of the layer-based CAN models.

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

3.5.2 Performance

We have measured the simulation performance also in a multi-node setup. One group of nodes, with higher priority, accesses the bus in a regular interval. The higher priority transactions produce a base utilization of the bus system, but leave a defined bus bandwidth unused. At the same time, the lowest priority node issues transactions of increasing size without any delay in between. Hence, the low priority transactions compete with high priority transactions, and are preempted or interspersed by higher priority transactions.

To measure the simulation performance, we have simulated the whole system. We have measured the time for simulating the lowest priority transaction while the whole system is running. Hence, the measured time includes higher priority transactions, that are transmitted in between.

Note that the actual traffic depends on the timing accuracy of the model under test, as preemptions may or may not be accurately modeled. The high priority nodes access the bus in constant time intervals. To illustrate the problem, assume that the low priority transaction would be modeled with a too long time. Hence, this longer low priority transaction would potentially be preempted more often in that model than in reality, yielding a non-comparable test scenario.

To ensure a fair comparison between the models regardless of their timing accuracy, additional control is required to guarantee that each model transfers the same amount of

data. For that, we use an adaptive test setup. It first measures for each test scenario the number of transactions with a known accurate model, the BFM. Then, while analyzing the model under test, the setup ensures that the same number of transactions is executed. With this additional control, all models transfer the same data amount regardless of the timing accuracy, and a fair comparison can be established.

All tests are executed on a Linux system, using a Pentium 4 PC at 2.8 GHz.

3.5.2.1 AMBA AHB

We use an architecture with two masters and two slaves for analyzing the AMBA AHB. The high priority master generates an equally distributed base load of 33% on the bus by sending 8-beat burst transactions in a predefined time interval. Figure 3.12 shows the time to simulate the low priority master's transactions over an increasing message size while the high priority master is running at the same time.

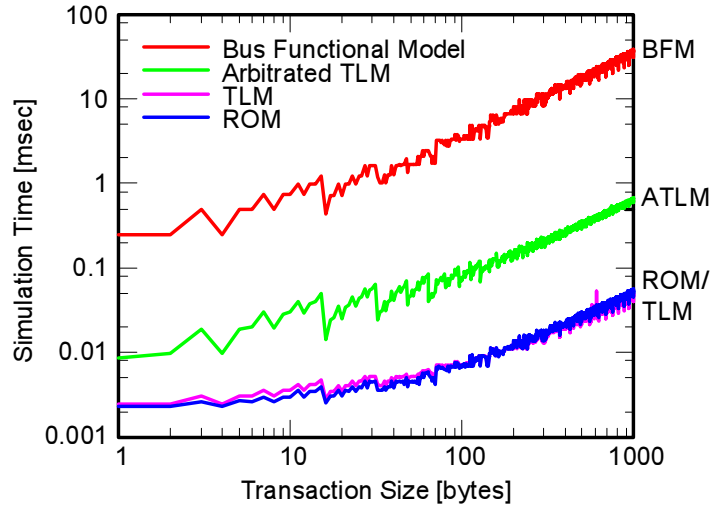


Figure 3.12: Transfer time using AMBA models.

Figure 3.12 reveals the tremendous performance benefit of ROM. Both ROM and TLM are equally fast, three orders of magnitude faster than the BFM, and one order of magnitude faster than the ATLM. Both the BFM and the ATLM show a characteristic saw tooth

shape, due to the non-linear split into bus transactions, equivalent to what we reported in Section 2.4.3.

It is interesting to note, that in this test setup also the TLM and the ROM show a saw tooth shape, although much less pronounced. Again, this is caused by the non-linear split of user transactions into bus transactions. Although no additional effort is needed for simulating an additional bus transaction within the same user transaction, as both the TLM and ROM do not explicitly model bus transactions, the simulated time for a whole user transaction increases with the additional bus transaction. The increased simulated time results in a higher probability of preemption. With an additional preemption, more higher priority transactions have to be simulated, which increases the simulation effort and therefore causes a longer simulation time.

3.5.2.2 CAN

We use a setup with 4 sending nodes. Three nodes send messages of higher priority with a size of 16 bytes each. These messages are sent in constant time intervals. The timing is setup as such, that the bus load produced by all three higher priority senders, occupies 50% of the bus bandwidth. As before, we measure the simulation time for lowest priority sender. Figure 3.13 shows the simulation time for transmitting a low priority transaction over an increasing size, running concurrently with the high priority senders.

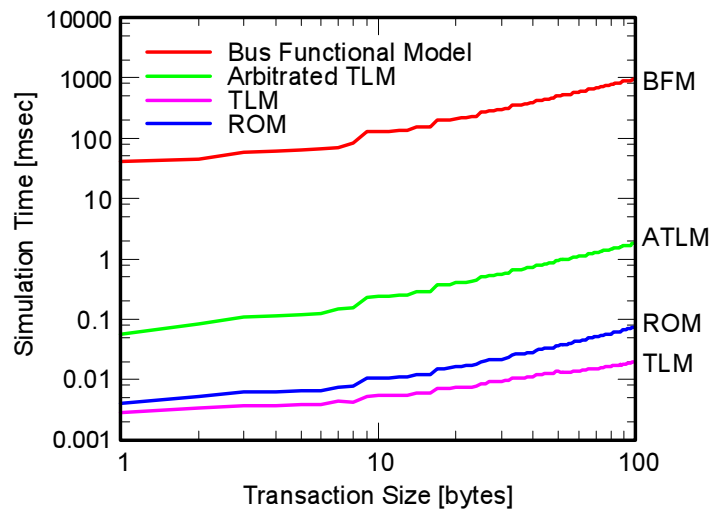


Figure 3.13: Transfer time using CAN models.

All models exhibit a characteristic increase in simulation time when exceeding a size divisible by 8. Then, another CAN frame is needed, which increases simulation effort and the probability of additional higher priority messages.

ROM and TLM execute five orders of magnitude faster than the BFM. The TLM reaches this speed (0.006ms 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, necessary for accurate timing, 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. For the CAN, ROM is the fastest to accurately model the communication, 12700x faster than the BFM.

In addition, we analyze the system simulation bandwidth to judge the scalability of a model and the overall performance. The system simulation bandwidth quantifies how much data is transferred by all nodes in a system in a given period of simulation time. The simulation bandwidth is reported in $\frac{MBytes}{sec}$. If only one sending node and one receiving node is used, the system simulation bandwidth would be equal to the simulation bandwidth, as reported in Section 2.3.1.

Similar to before, we use an adaptive setup, which ensures that all models transfer the same amount of user data, independent of the model accuracy. Table 3.3 lists the simulation bandwidth in $\frac{MBytes}{sec}$ for an increasing number of nodes. Half of the nodes are sending nodes, each issuing 16 byte transfers. In our traffic setup with multiple senders, higher priority senders leave spare bandwidth on the simulated bus, so that messages of different priority can be interspersed.

Table 3.3 shows that the accurate ATLM simulates, as in the earlier reported scenarios,

	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

Table 3.3: System simulation bandwidth [$\frac{MBytes}{sec}$].

multiple orders of magnitude slower than both ROM and TLM. With an increased number of simulated nodes, the TLM significantly drops in simulation performance. A system with 8 nodes yields only 41% of the simulation bandwidth than a system with 2 nodes (as the bandwidth drops from 12.7 MByte/s to 5.2 MByte/s). ROM on the other hand scales much better with an increased number of nodes. By simulating the additional nodes in the 8 node setup, the system simulation bandwidth of ROM remains at 73% of the bandwidth compared to the 2 node setup. We extrapolate from these results, that the ROM technique will scale well with an increase in model size and complexity.

3.5.3 Prediction Updates

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

Figure 3.14 shows an example where a long transaction is frequently preempted. Although this transfer is preempted 15 times, only 4 prediction updates are needed by the ROM. A closer look shows 8 preemptions during the initially predicted period, 4 in the next, then 2, and finally only one. This exponential decrease in number of preemptions indicates, that even under high bus load, only very few prediction updates are expected for most transfers.

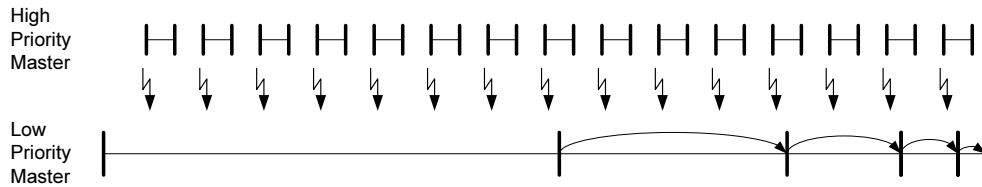


Figure 3.14: Exponentially decreasing number of prediction updates.

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

3.5.3.1 AMBA AHB

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

Figure 3.15(a) shows the number of prediction updates per low priority transaction in a histogram over 100000 transfers. Most transactions require only a single prediction update, despite the high amount of contention. As expected, the number of transactions with more than one prediction update reduces exponentially. Only 1.1% require 4 prediction updates. Note that 27.5% of the transactions complete without a single update. In other words, the initial prediction is actually correct for about a third of all cases. These are mainly small transactions (58% of the transactions without updates are 50 bytes or smaller in size).

3.5.3.2 CAN

We have validated the same fact using the CAN with 8 nodes (4 senders, 4 receivers). The histogram in Figure 3.15(b) shows the results for the condition in which we expect the most updates: the node with the lowest priority messages at a high bus contention of 51%.

The percentage of transactions requiring prediction updates reduces exponentially with the number of updates. 66% require no prediction update at all, with the majority being single frame messages. 23% require one prediction update, and only 0.5% need 4 updates.

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

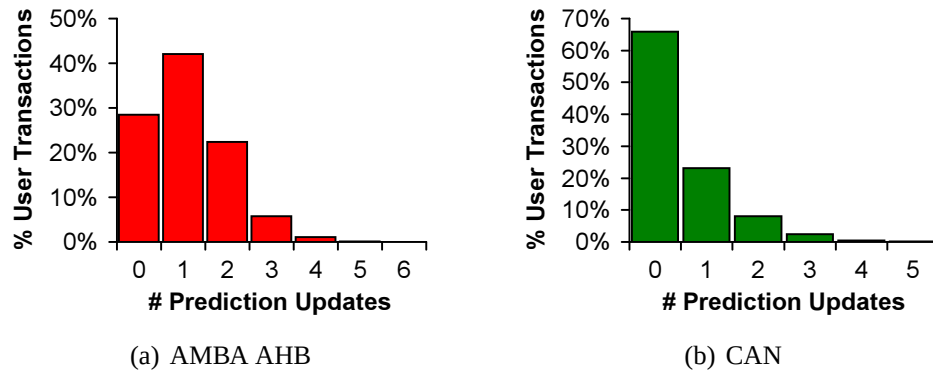


Figure 3.15: Histogram of number of prediction updates.

3.6 Generalization

Combining the results of the two bus systems with complementary characteristics, we now derive more general conclusions. While, for the existing multitude of embedded bus architectures, an extrapolation based on two individual models is difficult in general, our data is a strong indication of generality. The modeled bus systems are on opposite ends of the spectrum. While the AMBA AHB is an on-chip parallel bus system with a centralized arbitration scheme, the CAN is an off-chip serial bus with distributed arbitration. We estimate that the results of the AHB modeling are a good indicator for abstracting the IBM CoreConnect Processor Local Bus [48]. Furthermore, we believe the CAN modeling experience can be used as an indicator for other serial bus systems such as I²C [77] and Ethernet.

3.6.1 Escaping the TLM Trade-Off

Figure 3.16 shows the combined results for the AMBA AHB and the CAN in a graph with actual measurements. Note that Figure 3.16 resembles the generally TLM trade-off as outlined in the introduction. Again, the graph illustrates the trade-off between accuracy and speed. The x-axis denotes the bandwidth of the lowest priority node when using 100 byte transactions in a multi-node setup with two senders⁵. The y-axis denotes the average error in transfer duration for the lowest priority node when the bus shows 40% contention.

Our experimental results show that the traditional models (TLM, ATLM and BFM) follow the TLM trade-off. They are either accurate (i.e. the BFMs and the ATLM for the CAN) but slow with less than 0.4 MBytes/s, or they are fast (i.e. the TLMs) but exhibit an error of more than 35%.

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

⁵ Note that the simulation bandwidth reported in Figure 3.16 is lower than shown previously in Figure 2.24. While Figure 2.24 reflects a single node test setup, Figure 3.16 uses a multi-node setup. Concurrently simulating more communication nodes results in a lower simulation bandwidth for an individual node.

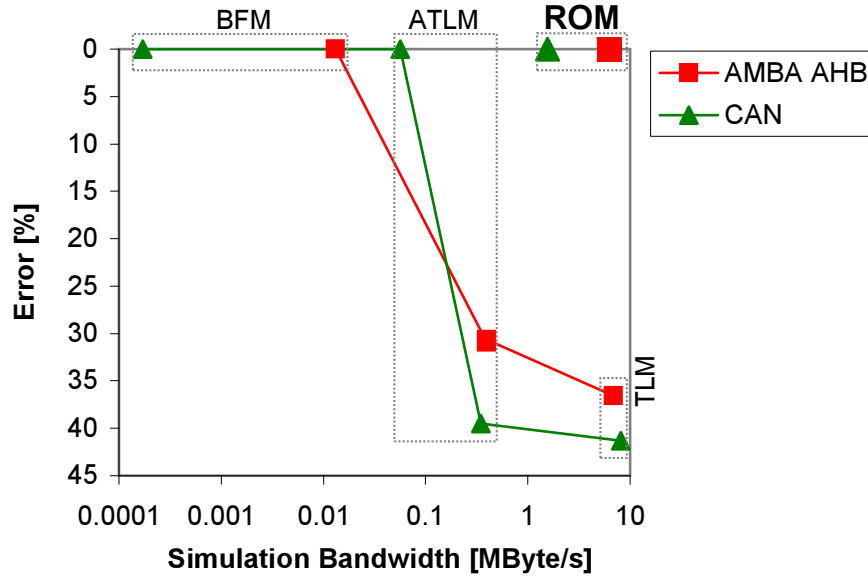


Figure 3.16: ROM beats the TLM Trade-Off.

3.6.2 Complexity Considerations

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

3.6.3 Limitations

As with TLM, ROM shares a limitation that an application issuing whole user transactions should only rely on the execution order of user transaction, and should not rely on the execution order at the *bus transaction* granularity. Differences may arise, when two masters simultaneously write a user transaction, which contains multiple bus transactions, to an overlapping address range. Then, the result would depend on scheduling of individual bus transactions that compose each user transaction. As both TLM and ROM may not model individual bus transactions, an application shall not rely on the bus transaction ordering.

In more detail, ROM uses a single *memcpy* to transfer data, regardless of the number of bus transactions. Therefore, the functional result reflects an atomic execution of the user transaction (including all its bus transactions), without interleaving bus transaction from another master. For calculation of the timing, on the other hand, interleaving is still considered to produce accurately timed results.

Note that we consider this restriction acceptable. A specification, that uses plain unsynchronized global memory for synchronization is unsafe, as it may incur race conditions. Instead, higher level synchronization mechanism should be utilized, which automatically translate to a safe implementation. On the other hand, if an unsynchronized global access is desired to explicitly capture race conditions, a user transaction that consists of single bus transaction can be used, as user transactions are accurately ordered.

As a second limitation, ROM does not support communication protocols, which include indeterministic aspects, as ROM (given a user transaction transfer request) relies on a static calculation of the communication schedule. For example, modeling Ethernet [65] using ROM would require an unpractical additional effort. Ethernet uses a Carrier Sense Multiple Access with Collision Detection (CSMA/CD) scheme for accessing a shared medium. It employs a randomized back-off strategy, once a collision has been detected. Hence, the transmission time over Ethernet depends on the random number generator. A timing accurate Ethernet model using the ROM technique would require modeling the random number generator of each participating bus node, which is not practical.

3.7 Summary Result Oriented Modeling

In this chapter, we have introduced a novel modeling concept, Result Oriented Modeling (ROM), and its application to the abstract modeling of communication in SoC design. ROM is a modeling approach similar to TLM that hides internal states and minimizes them in order to gain execution speed [89]. Moreover, ROM is based on an optimistic paradigm. It predicts the end result at the beginning. Under a disturbing influence, corrective measures are taken at the end, in order to adjust the prediction so that 100% accuracy is achieved.

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

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

Chapter 4

Abstract Processor Modeling

4.1 Introduction

Recent state-of-the art MPSoC increasingly employ a large number of processors, and the software content increases significantly. Validation and debugging of embedded software for such systems is becoming a tremendous challenge, and more and more infeasible with current methods.

Traditionally, embedded software is validated and debugged using Instruction Set Simulators (ISSs) which provide functional and timing accurate simulation on a host platform at a very fine granularity. However, interpreting ISSs simulate very slowly, especially when multiple instances are integrated into a MPSoC system simulation. Therefore, ISS-based validation and debugging is not sufficient to match the needs for rapid design space exploration at the system level. Thus, a higher level of abstraction is needed. The simulation performance will increase with the higher level of abstraction, however, the gain in speed is traded off with a decrease in timing accuracy, which has been described earlier as the TLM trade-off in Section 1.2.4.

Figure 4.1 shows an adapted TLM trade-off for the modeling of complete systems. ISS-based co-simulation will yield accurate timing results with a low simulation performance. On the other hand, executing pure C-code in a functional simulation will be very fast, however as timing is not modeled, very inaccurate in terms of timing. Only execution on the real hardware will yield fast and accurate results. However, real hardware may

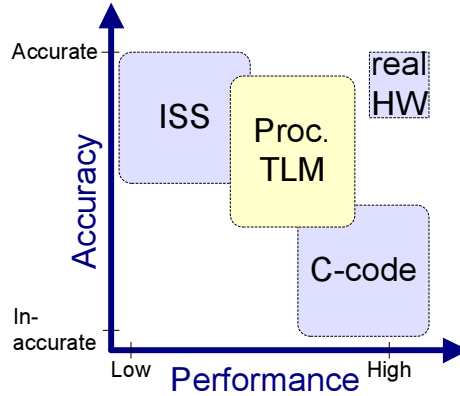


Figure 4.1: Trade-off in system simulation.

not be available early in the design process. For rapid design space exploration and early validation, an abstract processor model (indicated as Proc. TLM) is needed, which bridges the gap, in terms of simulation speed and accuracy, between ISS-based simulation and pure functional C-based simulation.

In this chapter, we present our approach of abstracting the software execution environment while retaining a high accuracy of the models. In particular, we describe our abstract processor modeling as an integral part of current SoCs. Our processor models provide an a high-level software execution environment with accurately timed execution, real-time operating system (RTOS) services and external communication. Our models exhibit highest simulation performance while maintaining an acceptable accuracy in simulated timing, and exposing the structure of the software architecture (e.g. drivers and interrupts). Using our abstract processor models in conjunction with transaction-level models (TLMs) of communication [88] dramatically increases the execution speed in a co-simulation environment. With high accuracy in timing, our models enable an early functional and fast simulation of the desired target architecture, clearly exposing the implications of programming and architectural decisions, and thus allowing rapid validation, debugging and design space exploration.

4.1.1 Problem Definition

With the importance of embedded software within a MPSoC design, fast simulation of embedded software is needed. The traditional ISS based co-simulation satisfies all func-

tional requirements, however at a very low speed. Faster simulation capabilities are needed for efficient exploration, validation and debugging.

We address the need for fast software simulation by abstracting the software execution environment providing timed execution, dynamic scheduling and external communication. We develop a corresponding high-level, abstract processor model. We aim to significantly increase simulation performance while maintaining an acceptable accuracy in simulated timing, and properly reflecting the structure of the software architecture (e.g. drivers and interrupts).

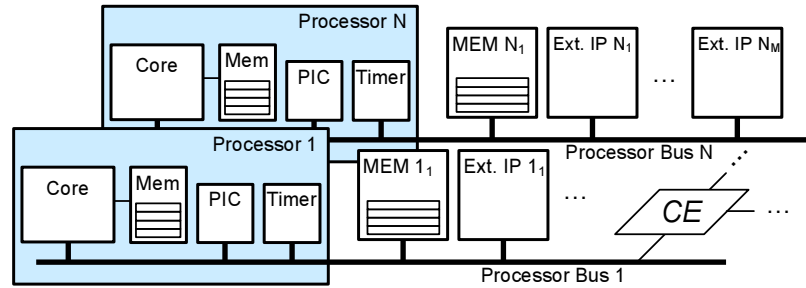


Figure 4.2: Generic MPSoC target architecture.

We focus on a Multi-Processor System-on-Chip (MPSoC) target architecture as outlined in Figure 4.2. It consists of a set of processors, where each processor is connected to a processor specific main bus. We assume that each processor contains an internal memory, which stores the execution binaries and local variables. Additionally, we associate a customizable Programmable Interrupt Controller (PIC) and a timer with each processor.

Each processor communicates with external memory (holding globally shared variables) and with custom hardware IP blocks (through memory mapped I/O and interrupts) over the processor main bus. Any number of IP blocks and memories may be connected to each bus. A processor can also communicate with other processors and external IPs or memories connected to other busses through one or more Communication Elements (CEs)¹.

At the input, we assume that the user application is given in the form of C code for each task, including information about task relations and explicit communication between

¹ In extension to what is shown in Figure 4.2, we assume that the busses may be arranged as a hierarchy of busses.

tasks. Furthermore, we assume that data about execution delays for each task is available. We make this assumption to segregate the two main error sources for modeling software execution: (a) target specific user code profiling and back annotation, (b) errors due to processor feature abstraction (see Section 4.4.1.2). For the purpose of this work, we focus on the processor modeling and therefore are interested only in the error due to feature abstraction. The equally important target specific user code profiling is outside of the scope of this publication.

4.1.2 Overview

The chapter is organized as follows: we will first outline the overall MPSoC development approach in Section 4.2. Section 4.3 then describes abstract processor modeling approach in detail by incrementally describing its features. In Section 4.4, we validate our modeling approach using set of industrial-strength MPSoC examples. We analyze the simulation performance and the achievable accuracy in detail. Finally, we conclude the chapter with a summary in Section 4.5.

4.2 Context: Our MPSoC Development Approach

Before we describe our abstract processor modeling, we will outline in this section our overall MPSoC development approach to show the context in which our processor models are used. Note that the overall MPSoC development approach is not a contribution of this dissertation, and has been described in [27]. Instead this chapter discusses the processor modeling itself, and the analysis thereof [92].

Figure 4.3 shows the framework for programming, validation and debugging of embedded software for MPSoCs. At the input, it assumes the user application as a set of communicating processes with C code for each task, including information about task relations and inter-process communication (IPC).

The system compiler takes the application specification and compiles it down to an implementation on a target architecture, creating a binary for each processor in a MPSoC architecture, describing the hardware components, and their connectivity. The designer

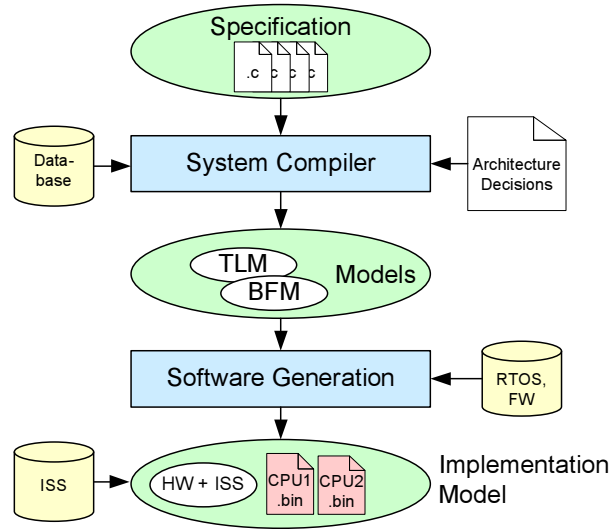


Figure 4.3: Software development framework (source [27]).

specifies the architecture by selecting components from the database. The designer also selects a mapping of application tasks and IPC channels to those components.

The system compiler automatically generates the processor models that we describe in this chapter. In a stepwise process, the system compiler inserts necessary implementations of task scheduling, bus drivers, interrupt handlers, hardware abstraction layers and other firmware. Furthermore, the system compiler performs back-annotation of task execution delays at the function level for timing feedback during simulation. At its output, the system compiler generates transaction-level (TLM) and bus-functional (BFM) models of the target platform running the application. In addition, the compiler can produce intermediate system models at varying levels of detail.

The generated models are then used as an input to the software generation back-end. This automatically generates the embedded software for each processor down to the final binary image. The software generation includes creating C code for the application tasks, bus drivers and firmware. It targets the code for a chosen RTOS implementation to perform processor initialization, multi-tasking and interrupt handling. The software generation furthermore includes cross-compilation and generates a binary for each processor.

The generated images can be directly loaded onto the target platform for execution. They are the final implementation for the software. The back-end also allows validation

of the final target binaries independent of the availability of the hardware platform. For that, the back-end generates additionally an ISS-based co-simulation model. This model is based on the system BFM. It describes all hardware and includes one ISS instance per processor.

Although all models are generated automatically by the system compiler, we do not want to deviate into the synthesis aspect, discussing the system compiler, in this chapter. Instead, we focus on the abstract processor modeling itself.

4.3 Abstract Processor Modeling

Our models are captured in a System Level Design Language (SLDL). We chose SpecC [29] for our experiments. The concepts however, are applicable to other SLDSs as well. The SLDL tool set provides a timed, fast, discrete event simulation natively on a simulation host. Our processor models abstract away the processor micro architecture and do not simulate the instruction set architecture (ISA). Instead, they abstractly reflect the behavior of the processor and the software execution environment, including all desired features like RTOS task scheduling, external bus communication, and interrupt handling with processor suspension, interrupt nesting, and interrupt priorities.

Observing SW execution on a typical processor, we can identify several layers of functionality. We have organized our models in layers along features. We will use these layers to incrementally describe our model. In total, we will show three intermediate models until reaching the actual proposed processor TLM. For our detailed analysis, we will consider the intermediate models, then the TLM, and finally also investigate the accuracy of a processor BFM. We will compare the models to an ISS-based reference model. This detailed analysis, including the intermediate models, will allow us to evaluate each feature individually for its cost in simulation speed and contribution to timing accuracy.

The subsequent sections describe in general each step of the processor modeling, starting with the uppermost layer.

4.3.1 Application

In order to achieve maximum simulation speeds, we execute the *application* natively on the simulation host. We use it as the innermost layer of our abstract processor, as depicted in in the box *CPU* in Figure 4.4.

The user code is captured in the SLDL as a hierarchical composition of behaviors separating computation and communication. Behaviors can be flexibly constructed to execute sequentially, in parallel, in a pipelined fashion, or state machine controlled.

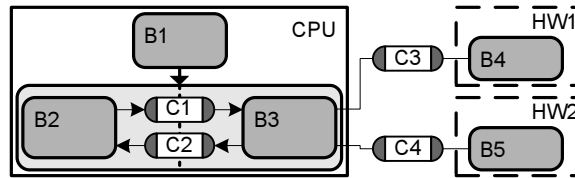


Figure 4.4: Application model and external communication.

Communication is expressed using a set of standardized abstract channels for high-level, typed message passing. These standardized channels offer synchronous and asynchronous communication with the option of one way, two way data traffic or synchronization only. Additionally, behaviors may communicate through shared variables. The connectivity between behaviors (either through channels or global variables) is expressed through ports, where channel/variable instances connect to. Therefore, our application model captures the three aspects of computation, communication and connectivity.

For processor modeling, we distinguish between internal communication (tasks on the same processor) and external communication (to other processing elements). The internal communication, as shown with the *C1* and *C2* channels in Figure 4.4, largely remains the same throughout the modeling. The external communication (e.g. as initiated by behavior *B3* through *C3* and *C4*) on the other hand, will be refined throughout the modeling stages to follow a generic protable communication stack (see [37]). Note that for illustration purposes, Figure 4.4 additionally shows external hardware components *HW1* and *HW2*, which are not part of the processor model.

At the level of the application model, external communication is identical in functionality as the internal communication (i.e. untimed, without bus connectivity). Global vari-

ables, which are shared between processing elements, are accessed directly at this abstraction level without special synchronization or bus communication.

The user application is executed natively on the simulation host inside the discrete event simulation environment to achieve highest simulation speeds. In order to simulate the target specific execution timing, the application C code is back-annotated with estimated execution timing. The granularity of timing back annotation significantly influences the achievable accuracy and the simulation performance. A very fine granularity of back annotating each individual C-operation would yield a high accuracy, since it captures data dependent execution². However, this granularity will also result in a large number of wait-for-time statements, therefore slowing down the simulation. On the other hand, a too coarse grain back annotation will capture data dependent execution sufficiently.

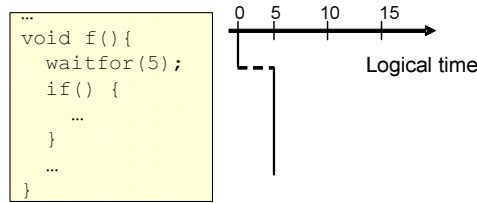


Figure 4.5: Timing back annotation.

The timing information can be automatically back annotated (e.g. see [18]). In our case, the system compiler inserts wait-for-time statements at the function level (see Figure 4.5). We chose to back annotate at the function level, since the applications of our examples are sufficiently separated into functions, which already express the data dependent execution.

4.3.2 Task Scheduling (OS Kernel)

Concurrent execution of software on an inherently sequential processor requires an operating system on the target. In order to explore the effects of dynamic scheduling decisions, the system compiler inserts an abstract RTOS model [38] as shown in Figure 4.6. The abstract RTOS model emulates the dynamic scheduling on top of the SLDL framework. Since it integrates with the simulation environment, it enables high execution performance, thus allowing early exploration of scheduling policy decisions.

²Also the preemption simulation improves significantly, which we will discuss in Section 4.3.2.

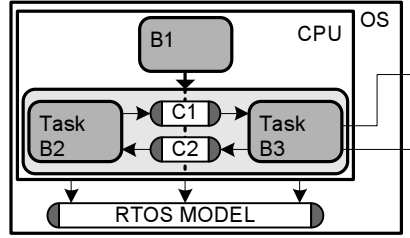


Figure 4.6: Task model.

In order to execute on top of the OS model, each parallel executing behavior is refined into a task (e.g. *Task B2*, *Task B2*), defining task execution control and scheduling policy specific parameters (i.e. task priority). A sequentially executing behavior becomes part of its parent task. For example, behavior *B1* becomes part of the processor main task, since it is only executed on startup. It spawns the two tasks *Task B2*, *Task B2*.

To simulate the dynamic scheduling, each primitive that potentially triggers scheduling is wrapped to interact with the abstract RTOS model. Such primitives include task start, channel communication, and wait-for-time statements.

As a result, the abstract RTOS model controls each task's state and decides the task scheduling specific to the selected scheduling policy (e.g. priority based or first come first serve (FCFS)). It switches between the tasks (Figure 4.7) so that at any point only a single task is running on top of the underlying SLDL. Our OS model allows to observe the number of context switches, the performance effects of the selected scheduling policy, and the idle time for each processor in the MPSoC system.

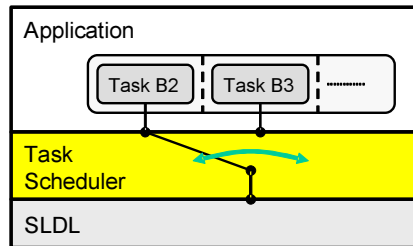


Figure 4.7: Abstract scheduler switching between tasks (source [38]).

drivers to synchronize using polling or interrupts. In case of polling, the generated driver code checks the external status with a designer selectable polling period. In the latter case of interrupt synchronization, the system compiler generates an interrupt handler that is registered to the external interrupt source.

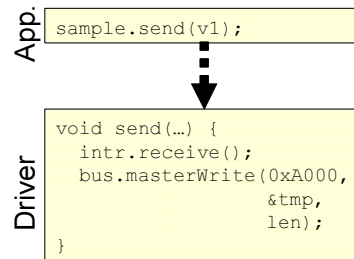


Figure 4.9: Example of inserted driver code for synchronization.

To simulate interrupt handling, the firmware model captures the interrupt inputs of programmable interrupt controller (PIC) connected to the processor (here *INTA - INTC*). This model abstracts away the actual processor interrupt for simulation speed. Interrupt inputs are implemented as channels and can be triggered directly by external sources (such as a slave). The interrupt input channel, then triggers execution of one or more user interrupt handlers. The example shows the option of interrupt sharing, where two user interrupts share a the same interrupt input. Here, each user interrupt handler contains additional code to test its interrupt source. Finally, the user interrupt handler uses a standard channel (i.e. semaphore) to release its driver after successful synchronization.

The firmware model is the first to contain the complete software (including the driver and interrupt code). The firmware layer marks the boundary between software implementation and hardware features of the processor.

4.3.4 Processor Transaction Level Model

To complete our processor model a description of the processor hardware is needed. This includes hardware interrupt handling, interrupt scheduling, timer and PIC hardware, and simulation of bus accesses at bus transaction granularity.

Thus far (i.e. the firmware model), an interrupt triggering HW directly calls the interrupt handler. Thus, interrupts are not scheduled and may execute concurrently to the user

The processor TLM offers interrupt scheduling to avoid that interrupts are executed in parallel with application tasks. The behavior *HW Int* monitors the processor interrupt line(s). Upon occurrence of an interrupt, *HW Int* suspends the processor simulation - all SW behaviors inside the HAL shell (as indicated by the half circle around the bottom right corner of the HAL shell). Therefore all tasks of that processor are now suspended.

The *HW Int* implements the HW interrupt scheduling and calls the system interrupt handler *SysInt* via a channel call. To handle multiple occurring interrupts, the *HW Int* observes interrupt priorities and nesting for an accurate scheduling. Additionally, it provides interrupt related control registers to the driver software e.g. for enabling and disabling interrupts.

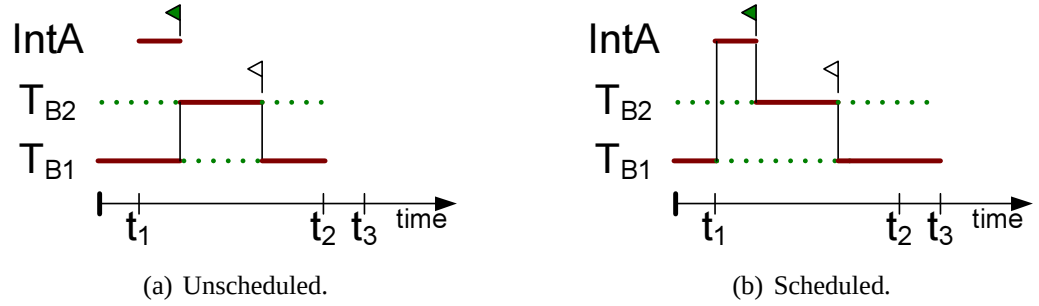


Figure 4.11: Hardware interrupt scheduling.

The system interrupt handler communicates through the bus interface with the PIC determining the incoming external interrupt. The system interrupt handler releases through the interrupt channel (e.g. *INTC*) the actual user interrupt (e.g. *UserInt2*). The user interrupt handlers are modeled as high priority tasks inside the abstract RTOS model. At this point, the hardware and low level interrupt handling is completed. The call chain, initiated by the *HW Int*, terminates and the HAL shell is released from the suspended state. Next, the abstract RTOS model schedules the highest priority ready task, the user interrupt handler *UserInt2* in this case.

As a result, the complete interrupt chain is replicated for an interrupt execution of a processor and is modeled closely to existing hardware. It allows us to accurately simulate the interrupt execution overhead and provides interrupt scheduling. The effects of the scheduled interrupts is shown in Figure 4.11(b). Opposite to the unscheduled version, the interrupt handler (*IntA*) now preempts the execution of all tasks. The simulation sequence of this example now correctly ends at *t₃*.

External communication in the processor TLM is modeled by a transaction level model of the bus, providing a cycle approximate simulation with user transaction granularity.

4.3.5 Processor Bus Functional Model (BFM)

To obtain a bus-functional variant of our processor model, see Figure 4.12, we employ a pin- and cycle-accurate model of the processor bus interface. The modeling of computation is identical to the TLM. The communication, on the other hand, is further refined. The system compiler introduces a bus specific Media Access Layer (MAC) that splits the arbitrary sized user transactions into bus transactions (bus protocol primitives). The BFM

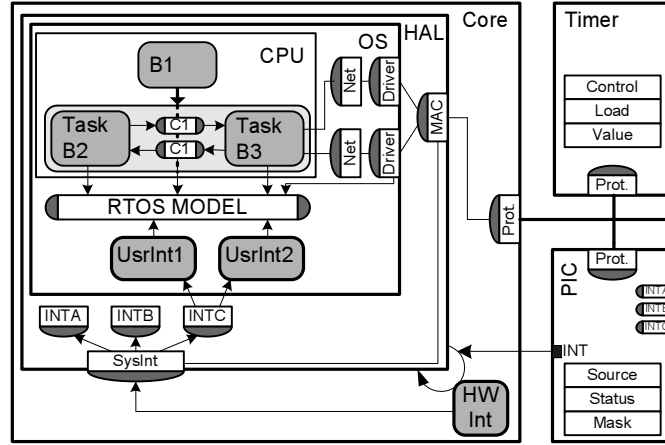


Figure 4.12: Processor Bus Functional Model.

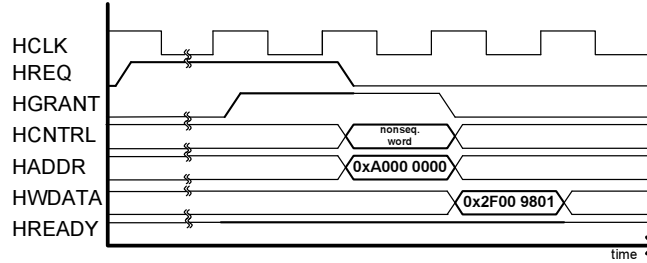


Figure 4.13: Bus trace in BFM.

includes protocol channels that provide bus access with a bus transaction granularity and implement each bus interface pin accurately. Channels implement state machines to realize the bus protocol to drive and sample the individual bus wires. Additional active components (such as arbiter, multiplexer) are inserted if needed to accurately implement the bus protocol. Using the BFM allows us attach RTL components and to create signal traces for the bus traffic (Figure 4.13) for debugging purposes.

4.3.6 ISS-based Cosimulation Model

In order to validate SW binaries and for a detailed cycle-accurate SW execution, we use an Instruction Set Simulator (ISS) based reference model. This model includes a behavioral model of all hardware components and an ISS instance for each processor in the system. Each ISS is wrapped to adapt the ISS API to the simulation environment, as shown in Figure 4.14.

We use ISSs that are available in library form with a cycle callable interface. Each ISS is called within an own wrapper module that executes the ISS cycle-by-cycle. The wrapper uses the ISS API to detect ISS external memory accesses and translates them to transactions on the simulated bus. It also listens to incoming interrupt signals to the processor and forwards them to the ISS. An ISS typically includes internal memory for storing the executed binaries and local variables.

From the software perspective, the ISS-based model represents the real implementation. It executes the final binaries, accesses external hardware through the bus and allows interrupt based synchronization.

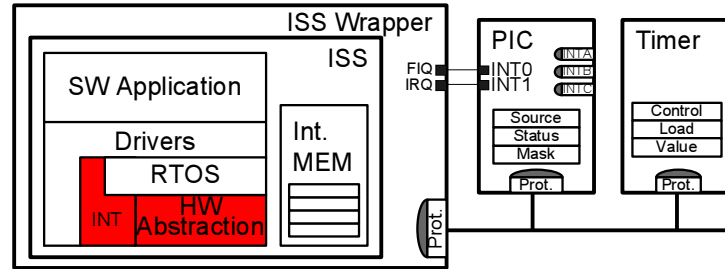


Figure 4.14: Bus Functional Model with ISS.

4.3.7 Model Benefits

To summarize our processor model, Table 4.1 lists the features captured and also indicates at which level each feature is introduced. The most abstract model at the application level implements only a single feature. On the other hand, the ISS reference realizes all listed features.

Features	Level
Target approx. computation timing	Appl. ↓
Task mapping, dynamic scheduling	Task ↓
Task communication, synchronization	Firmware ↓
Interrupt handlers, low level SW drivers	TLM ↓
HW interrupt handling, int. scheduling	BFM ↓
Cycle accurate communication	BFM - ISS ↓
Cycle accurate computation	↓

Table 4.1: Features and layers in abstract processor models.

Our most abstract model, the *application*, allows with the target approximate computation timing an early functional validation of the specification. It exposes the hierarchical task graph and allows observing of data and control flow across multiple processors. With the *task* model, additionally the effects of dynamic scheduling are observable in simulation. It enables selecting a proper scheduling scheme and priority distribution for all tasks in the system. It is the most abstract model that enables monitoring utilization of a processor. It therefore is important for design decisions regarding load balancing within a multi-processor system. The *firmware* model completes the modeling of software. It shows the synchronization across the network, interaction between interrupt handlers and tasks. This model enables firmware and driver debugging at a high-level.

The *TLM* adds a detailed model of the interrupt chain. The designer therefore can observe the interrupt latencies for each processor and the overall interrupt overhead in the processor load. Furthermore, the designer can validate the efficient selection of synchronization scheme and the interrupt priority assignment across the complex MPSoC. This model provides a detailed view of the busses in the system (e.g. utilization, contention and access latency). With this rich detail, the *TLM* serves as a validation platform with a high confidence level.

The *BFM*, in turn, allows observing all bus traffic in pin and cycle accurate detail. This exposes all communication details and provides observability beyond what is typically feasible in real hardware. Furthermore, the *BFM* eases the integration of RTL-only IP components. Finally, the cycle-accurate *ISS-based* model allows validation of the software binaries before availability of the final hardware. It provides fine-grained performance measurements of the system.

4.4 Experimental Results

To validate our approach for abstract processor modeling, we will now separately analyze the output of each compilation step to compare the effects with respect to simulation speed and timing accuracy.

We have applied our approach to an industry-strength cellphone example, including

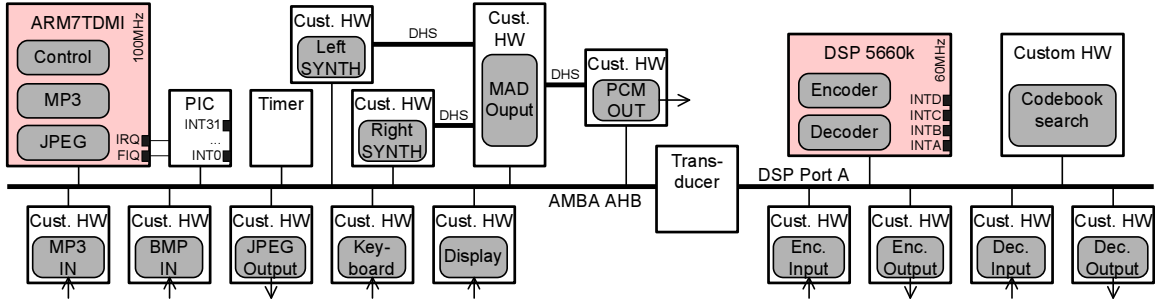


Figure 4.15: Example cellphone architecture.

three subsystems: a GSM 06.60 voice transcoder [28], a MP3 audio decoder, and a JPEG image compressor [10]. We have mapped these applications to a MPSoC platform (Figure 4.15) similar to the baseband platform used in the RAZR cellphone [40]. This platform includes two processors, an ARM7TDMI [2] and a Motorola DSP56600 [67] connected by a transducer. The ARM processor executes the cellphone control tasks, the MP3 decoding and the JPEG encoding, using the MicroC/OS-II [54] RTOS for priority scheduling. The DSP is dedicated to GSM transcoding. Its encoding and decoding tasks are scheduled by a custom priority-based scheduler.

Computation intensive operations, such as the MP3 synthesis filter and the GSM codebook search, are mapped to custom hardware blocks. Ten additional hardware units perform I/O operations, including the keypad and display interfaces to the user.

4.4.1 Setup

Following our modeling approach, we have specified the phone application and standalone versions of each subsystem in an SLDL³ and used our system compiler to automatically generate the individual models. Please note that our system compiler is separated into individual compilation stages ([74, 38, 1, 111]). We have simulated all⁴ models on a 3.0GHz Pentium IV running Linux Redhat Enterprise 3. For all tests, we use the same

³ We used SpecC[29] for our experiments. The models, however, are equally applicable to other SLDLs, including SystemC.

⁴Note the Motorola proprietary ISS for the DSP56600 was only available for Sun Solaris. Hence, we have executed ISS models that include the DSP56600 on a Sun Fire V240 with an 1.5GHz UltraSPARC IIIi processor. To estimate a comparable simulation time on the Pentium VI, we have divided the simulation time by 2.5 (empirically determined using representative code).

input data consisting of GSM voice samples, a MP3 audio stream and a BMP picture. To balance the impact of each application, the input data set is balanced to represent 1.5 seconds of real-time - voice/audio samples (for MP3, GSM) and computation time (for picture encoding). The hardware blocks are simulated at behavioral level with cycle-approximate timing. We use the ISS-based reference model for cycle-accurate execution of software. We integrated the ISS SoftwareARM (SWARM) [25] for simulating the ARM7TDMI and a Motorola proprietary ISS for simulating the DSP as outlined in Section 4.3.6.

4.4.1.1 Performance Metrics

We report the performance measurements with three metrics. First, we measure and report the simulation time. This is the time it takes to simulate our test data set on the simulation host. Second, we then use the simulation time to compute the speedup over the ISS-based simulation. This indicates the relative improvement over the reference solution. Finally, we calculate the number of cycles simulated per second, providing an absolute reference. For that, we first measure the number of cycles when using the cycle-accurate ISS, and then divide by simulation time of the abstract model. Please note that this metric does not include the cycles computed by hardware.

4.4.1.2 Accuracy Metrics

Accuracy can be segregated into many aspects. For the analysis purpose of this chapter, we will distinguish three aspects of accuracy: functionality, represented feature detail, and timing. For our analysis, we will fix the first, vary the second, and measure the third aspect.

The first aspect of functional accuracy requires that the model executes the algorithms described in the specification and that the correct data is received by external hardware components. Since this is a necessary requirement for a functional simulation, all our models are functionally accurate. The second aspect is the represented detail level, which states how many features of the processor are actually present in the model. We vary this aspect in our model under test, ranging from only a timed execution (as in the application model) to the pin accurate processor model BFM. The third aspect is the accuracy in timing which we measure in our tests.

Within the aspect of timing accuracy, we can identify two main sources of possible errors. One, the timing back annotation of the user code and two, the presence of modeled features in the actual processor model. Both contribute to the timing accuracy of a model. However, it is difficult to measure them separately.

In this chapter, we focus on the processor models and their inaccuracies due to feature abstraction. Accurate target-specific profiling of user code and the timing back annotation are outside the scope of this dissertation. We therefore minimize the effect of the timing back annotation by using the most accurate and available numbers. Therefore, the accuracy measured in our tests reflects only the errors induced by feature abstraction.

To isolate processor modeling features from task delay errors, we back-annotate accurate execution timing obtained by simulating the application on a cycle-accurate ISS. We use the same input data for obtaining the execution timing via the ISS, and for measuring the accuracy of our abstract models. As stated before, we back-annotate at the function level. While the ISS-based approach for the timing back-annotation is not efficient for exploration of new designs, it allows us here to examine the quality of our processor modeling.

We use the metric of the simulated delay per individual frame (speech frame for the GSM, PCM frame for the MP3, and stripe for the JPEG). While executing the model under test, we record the simulated delay for each frame and compare it against the cycle-accurate ISS-based reference model to calculate the timing error.

For the analysis of our processor models, we define the error in simulated frame delay as a percentage error over the reference model:

$$\begin{aligned}
 d_{ISS} &: \quad \text{frame delay in ISS simulation} \\
 d_{test} &: \quad \text{frame delay in model under test} \\
 error_i &= 100 * \frac{|d_{test} - d_{ISS}|}{d_{ISS}} \tag{4.1}
 \end{aligned}$$

Given this definition, accurate models exhibit 0% error.

We analyze the models of each compilation step focusing on two aspects: the *performance*, since an increased speed is the main goal of abstraction, and the loss in *accuracy*, as a side effect of abstracting features. Combining both, we can quantitatively evaluate

the quality of our processor abstraction. We first analyze individual subsystems and later examine the entire system.

4.4.2 Performance Analysis

The achievable performance depends significantly on the system setup, especially on the balance between computation in software and in hardware. Furthermore, the amount of bus communication and its complexity affects performance. To isolate the impact of the system composition, we analyze three sets of systems. First, we examine software-only solutions of each subsystem. Second, we investigate hardware-assisted versions of each subsystem. Finally, we will analyze the complete cellphone system as outlined before.

4.4.2.1 Performance of SW-only Systems

In the software-only version of each subsystem all computation is performed on the processor. Input and output are mapped to external hardware components to maintain a realistic test. These I/O blocks, however, perform a negligible amount of computation. Therefore, the model's performance is dominated by our processor model (including its bus interface and interrupt synchronization), and allows us to focus on it for the analysis.

Figure 4.16 shows the simulation time for each software-only subsystem, when individually simulated. Table 4.2 shows the numerical results and compares them in terms of speedup and simulation performance.

The simulation time for the SW-only subsystems remains in the same range up to the firmware model. Here, the model abstracts away most of the processor features and the simulation speed is limited by the algorithm computation. Starting with the *TLM*, processor features start dominating the simulation effort and the results become specific for the modeled processor. The JPEG shows exceptionally long simulation time for the application model. It's specification frequently uses pipelined constructs, which leads to many parallel simulation threads in the application model, slowing down the simulation. These pipelined constructs are statically scheduled in the task model, reducing the simulation time.

The *TLM* shows a sustained simulation performance of 307 MCycles/sec to 637 MCycles/sec depending on the application. We measured peak performance of 2,800 MCy-

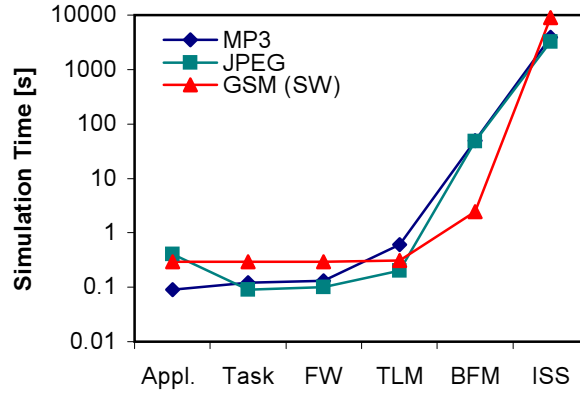


Figure 4.16: Simulation time for SW-only systems.

		Appl.	Task	FW	TLM	BFM	ISS
Simulation Time [s]	MP3	0.09	0.12	0.13	0.61	48.8	3921
	JPEG	0.40	0.09	0.10	0.20	48.6	3209
	GSM	0.29	0.29	0.29	0.31	2.5	9041 ⁴
Speedup	MP3	43563	32672	30159	6427	80	1
	JPEG	8022	35653	32088	16044	66	1
	GSM	31174	31174	31174	29163	3675	1
Sim. Perf. [$\frac{MCycles}{sec}$]	MP3	2080	1560	1440	307	3.8	0.05
	JPEG	318	1415	1273	637	2.6	0.04
	GSM	411	411	411	384	48.4	0.01

Table 4.2: Simulation performance of software-only systems.

cles/sec in code sections without external communication. Adding bit- and cycle-accurate communication in the *BFM* reduces the performance to about 3 MCycles/sec. The GSM application shows shallower drop in performance with adding details. The abstracter models are limited by the application complexity and the emulation of multiply accumulate operations. The *GSM TLM* and *BFM*, however, profit from the a simpler processor model (e.g. no external PIC), a simpler single master bus model, and from comparatively little communication.

4.4.2.2 Performance of HW/SW Systems

The HW/SW systems are the subsystems of the cellphone example. We use the MP3 with significant HW support (4 accelerator blocks and 4 busses in total) as shown in Fig-

ure 4.15. To examine multi-tasking between different applications, we also examine a combined MP3 and JPEG (at lower priority). The third example is the GSM with HW acceleration for the codebook search. Figure 4.17 shows the simulation time for each HW/SW subsystem and Table 4.3 gives the numerical results.

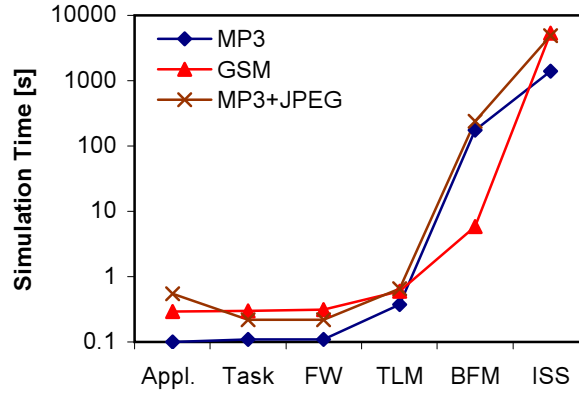


Figure 4.17: Simulation time for HW/SW Systems.

		Appl.	Task	FW	TLM	BFM	ISS
Simulation Time [s]	MP3	0.10	0.11	0.11	0.37	174.8	1383
	MP3+JPEG	0.55	0.22	0.22	0.66	239.4	4838
	GSM	0.29	0.30	0.31	0.60	5.8	5391 ⁴
Speedup	MP3	13835	12577	12577	3739	8	1
	MP3+JPEG	8797	21992	21992	7331	20	1
	GSM	18591	17972	17392	8986	926	1
Sim. Perf. [$\frac{MCycles}{sec}$]	MP3	1440	1309	1309	389	0.8	0.10
	MP3+JPEG	367	916	916	305	0.8	0.04
	GSM	341	330	319	165	17.0	0.02

Table 4.3: Simulation performance of HW/SW systems.

In general, the number of simulated cycles is lower with the additional hardware and communication, since we do not account computation on hardware components or bus communication toward the simulation cycles. With this definition, the TLM reaches half of the previous performance with a range from 165 MCycles/sec to 389 MCycles/sec. For the MP3, less than one third of the computation is performed on the processor (the SW-only

frame delay of 32ms drops to 9ms with the HW support)⁵ so that the *TLM* performance slightly increases. With the increased communication, the gap between *TLM* and *BFM* is larger. The *BFM* only reaches 0.8 MCycles/sec for the ARM based systems. The GSM with the simpler communication still reaches 17 MCycles/sec.

Summarizing the performance measurements, our processor abstraction provides very high simulation speeds. The achievable performance is higher with an increased software content (up to 637 MCycles/sec sustained, 2,800 MCycles/sec peak). Abstracting beyond the *FW* level is not efficient, since the performance does not increase further.

4.4.3 Accuracy Analysis

After having asserted the benefits of abstract processor modeling in terms of simulation performance, we will now look at the achievable accuracy with abstract simulation. Again we separate our analysis into SW-only and HW/SW systems.

4.4.3.1 Accuracy of SW-only Systems

As described earlier, we express error as the average timing error in the simulated frame delay. Table 4.4 shows the results for the SW-only version of the subsystems.

	Appl.	Task	FW	TLM	BFM	ISS
MP3 (SW)	46.19%	1.51%	1.51%	1.63%	1.63%	0%
JPEG	58.13%	3.64%	3.61%	2.32%	2.32%	0%
GSM (SW)	6.78%	1.13%	1.10%	0.72%	0.72%	0%

Table 4.4: Simulation accuracy of SW-only systems.

For SW-only subsystems, only modeling of the *Task* level significantly reduces the error. Here, parallel executing tasks are scheduled on the processor. Each further simulation level, which models the external communication in more detail, does not significantly reduce the error. The SW-only systems insignificantly use external communication. The error remains constant at about a 2% level. The *JPEG* subsystem still profits from the *TLM* simulation since it uses the bus slightly more comparing to the other two examples.

⁵ Note that we only count the processor cycles for computing the simulation performance. The cycles for computation in HW are ignored. However, CPU idle cycles are counted toward the performance.

4.4.3.2 Accuracy of HW/SW Systems

With the additional communication and synchronization present in the HW/SW systems, the abstract models exhibit a higher error amount (see Figure 4.18 and Table 4.5). For the ARM-based systems, the error drops linearly from $\approx 50\%$ error for the application down to few percent for the *TLM*. The *FW* level, that produced in excellent results in the SW-only measurements, now exhibits more than 10% error. The *TLM* simulates with less than 2% error. The *JPEG+MP3* has a slightly higher error of 3%, due to the frequent preemption of the JPEG encoder by the MP3 decoder (and our modeling of preemption).

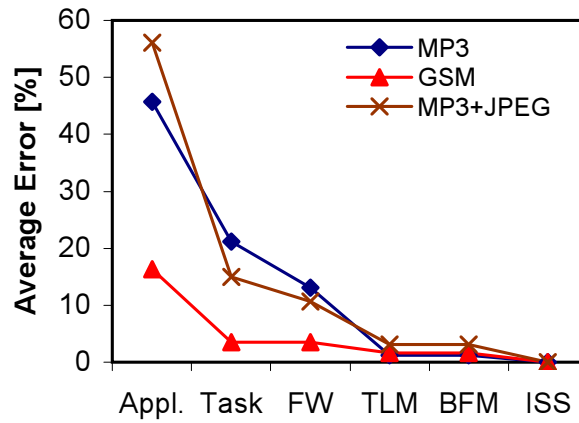


Figure 4.18: Accuracy of HW/SW systems.

	Appl.	Task	FW	TLM	BFM	ISS
MP3	45.72%	21.11%	13.02%	1.17%	1.17%	0%
MP3+JPEG	56.06%	14.90%	10.64%	3.09%	3.09%	0%
GSM	16.37%	3.56%	3.53%	1.68%	1.68%	0%

Table 4.5: Simulation accuracy of HW/SW systems.

4.4.4 Trade-off for System Simulation

Combining the performance and the accuracy measurements yields the trade-off in system simulation. We show this trade-off for a simulation of all three subsystems of the cellphone, as shown in Figure 4.15.

Figure 4.19 shows the results of our performance and accuracy analysis for the entire cellphone system with all subsystems running concurrently. It shows the simulation time and the average system error, which we define as the mean of the subsystem errors. For reference, Table 4.6 lists the detailed numerical results.

	Appl.	Task	FW	TLM	BFM	ISS
Avg. Error	42.2%	11.4%	7.25%	2.93%	2.93%	0%
Sim Time [s]	0.78	0.57	0.58	1.6	250	6469 ⁴
Sim. Perf. $[\frac{MCycles}{sec}]$	614	840	825	299	1.91	0.07
Speedup	8293	11348	11153	4043	26	1

Table 4.6: Trade-off for RAZR system simulation.

Our combined cellphone example simulates 3 seconds of real-time, with 180 million DSP cycles and 300 million ARM processor cycles. The simulation time increases over-exponentially with the added features. Our processor *TLM* executes 299MCycles/sec and is four orders of magnitude faster than the ISS-based co-simulation.

With an increased abstraction, a higher error in simulated time has to be accepted. Our processor *TLM* exhibits an average error of only 2.9%. Representing fewer processor features further increases the simulation performance at the cost of a larger error. The *firmware* compilation step, for example, simulates 825 MCycles/sec, with a 7.25% error.

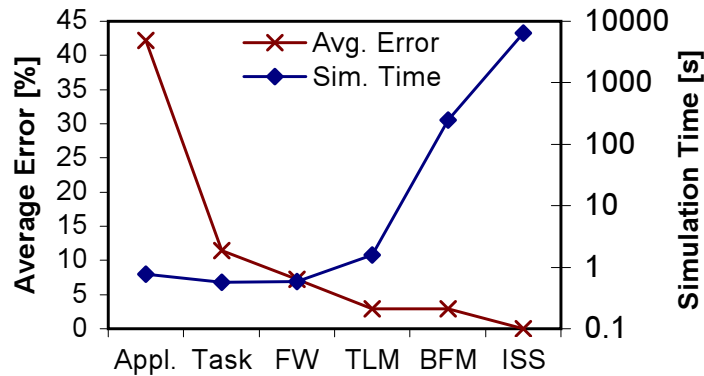


Figure 4.19: System performance and accuracy.

4.5 Summary Abstract Processor Modeling

In this chapter, we have presented our approach for system level software modeling using a system compiler [27] and abstract processor models [92]. We use a layered modeling approach. We have incrementally described our processor models with essential features of task mapping, dynamic scheduling, interrupt handling, low level firmware and hardware interrupt handling.

We evaluated our approach using an industrial strength cellphone example with GSM 06.60 speech transcoding, MP3 decoding and JPEG encoding mapped to a heterogeneous MPSoC with custom hardware blocks. We used our system compiler to automatically generate the processor models and intermediate compilation steps to analyze five levels of abstraction. We analyzed each level in detail with respect to the gain in simulation performance and the loss in accuracy.

Our results show the tremendous benefits of our abstract processor modeling of fast and accurate simulation. For SW-centric examples, our proposed *TLM* executes up to 600 MCycles/sec sustained and up to 2,800 MCycles/sec peak, which is multiple orders of magnitude faster than a traditional cycle-accurate *ISS*-based cosimulation. For the complex cellphone example, it reaches 300 MCycles/sec and exhibits an error of only 2.9%. An even higher speed of 825 MCycles/sec is achieved by the *firmware* model with an error of 7.25%.

Our processor models that are integrated into our ESL flow, which then automatically generates the model instances, enable a rapid design space exploration at an early stage of the design process. Automatically generating the models improves productivity for software development. It exposes performance implications of the design choices already early in the process. Furthermore, our processor TLM, with its rich set of details, serves as a platform for a fast functional validation with a high confidence level.

Our abstract processor models are complementary to existing *ISS*-based solutions. They do not replace *ISS*-based cosimulation for the validation of the final target binaries. Although our abstract models are already of significant value, they have in some limitations in their current form. A general limitation of abstract processor models is that their accuracy heavily relies on the accuracy of the target specific user code profiling and back annotation, which determine the simulation time for executing a basic block of straight line

code. Currently our annotated timing takes cache effects into account and we do not explicitly model caches at runtime. Hence, delays due to cache misses are only statistically covered. Furthermore, delays for the execution of operating system functions (e.g. during a context switch) are currently not captured.

Chapter 5

Summary and Conclusions

5.1 Summary

Abstract modeling, which uses the concept of hierarchy to hide implementation-level detail, is an important means to address the challenges of designing modern complex MPSoCs in the context of system-level design. It aims to capture a complete system as a composition of algorithms, their dependencies and communication. It allows the system designer to describe the system independent of the particular implementation, whether in software or hardware. A large portion of the system-level design relies on functional simulation using abstract models for system performance estimation, analysis, and validation. Additionally, virtual platforms are invaluable environments for development and debugging while real hardware may not be available. These important modeling applications drive the demand for fast and accurate abstract models.

Transaction Level Modeling has been established as the main approach for describing abstract models for a functional simulation. It abstracts communication above the level of pins and wires, and instead describes communication as larger transactions being transferred through the system. Transaction Level Models execute multiple orders of magnitude faster than comparable implementation-level models (such as RTL). Hence, they significantly contribute to an efficient design and validation environment. However, faster simulation speed through abstraction is typically traded-off for a lower simulation accuracy (e.g. with respect to simulated timing). In general, models are either fast, or accurate, which

defines the TLM trade-off. Although, TLM has been widely accepted, yet no detailed analysis of this trade-off has been performed before. Despite various efforts in the related work, abstraction levels, analysis methods, and modeling techniques were not clearly defined.

With the work reported in this dissertation, we have addressed three aspects of abstract modeling. First, we have analyzed the TLM trade-off in detail. Second, we have proposed an optimized modeling technique that eliminates the TLM trade-off. Third, we have introduced an approach for abstract modeling of processors as a software execution environment embedded into a system level design flow.

5.1.1 TLM Analysis

We have analyzed and investigated three communication protocols with a diverse set of characteristics for defining common abstraction levels and for quantifying the TLM trade-off. We have analyzed the on-chip standard bus architecture AMBA AHB with a centralized arbitration, the off-chip serial CAN with a decentralized arbitration, and a minimalistic processor bus, the ColdFire Master bus. Based on these diverse protocols, we have defined three abstraction levels for the general modeling of communication architectures. Our models abstract by decreasing granularity of data and arbitration handling. In particular, we have defined a Bus Functional Model (BFM) that models individual clock cycles, an Arbitrated Transaction Level Model (ATLM) which operates on bus transactions, and a Transaction Level Model (TLM), which handles arbitrary length user transactions.

We have defined test setups and metrics for a detailed analysis of the TLM trade-off in terms of simulation speed versus timing accuracy. Our results show, that modeling granularity is the key for an efficient abstract modeling. An abstraction based on a decreasing (coarsening) granularity yields at least one order of magnitude in speed improvement. On the other hand, decreasing (coarsening) the granularity significantly reduces the timing accuracy of the model. In general, a model can either be fast, or accurate, which defines the TLM trade-off. Our fast TLMs (with 100MBytes/s simulation bandwidth) show a significant error of up to 47%. On the other hand, the accurate models, the BFMs, simulate with less than 0.2MByte/s. The actual observed error depends on the bus contention, the bus features, and the traffic pattern transferred in the test.

5.1.2 Optimized Abstract Modeling Technique

We have introduced an optimized modeling technique, Result Oriented Modeling (ROM), and applied the modeling technique to communication modeling. ROM hides internal states, minimizing internal state transitions to gain simulation speed. In this aspect it is very similar to a granularity based TLM abstraction. In contrast however, ROM does not use incremental modeling and employs instead an optimistic prediction approach. Right at the start of a transaction, it predicts the end result (using the assumption of no additional preemptions). In case of a disturbing influence, i.e. a preemption during the runtime of the transaction, corrective measures are taken in the end to update the prediction and reach 100% timing accuracy.

We have demonstrated the tremendous benefits of ROM using complementary bus protocols, the on-chip parallel AMBA AHB and the off-chip serial CAN. ROM reaches near TLM performance, yet still exhibits 100% accuracy. With that, ROM eliminates the TLM trade-off for the analyzed communication protocols. As a result, models based on the ROM principle are more versatile and enable rapid design space exploration at very high fidelity. Additionally, ROMs accuracy enables using abstract models, where 100% accuracy is required, for example in the modeling of real-time systems.

5.1.3 Abstract Processor Modeling

To integrate simulation of software execution, we have introduced an approach for abstract processor modeling that abstracts software execution above the Instruction Set Architecture (ISA). Our processor model is organized in layers that incrementally add essential features for representing software execution: timed execution, task mapping, dynamic scheduling, interrupt handling, low-level firmware and hardware interrupt handling.

We have presented a detailed analysis of the trade-off between simulation accuracy and simulation performance, investigating five abstraction levels. Our analysis is based on a heterogeneous multi-processor example, executing GSM transcoding, MP3 decoding and JPEG encoding. Our experimental results show the significant advantages of abstract modeling. For SW centric applications our abstract model executes up to 600MCycles/s sustained, while a complex multiprocessor platform with hardware accelerators still reaches

300MCycles/s, with an error of only 2.9%. Modeling dynamic scheduling has shown in our examples to be the most significant contributor to an accurate simulation. Next important are capturing interrupt handling followed by expressing external communication.

Abstract processor models that integrate into an ESL flow are integral for an efficient design space exploration and for the development at an abstract level. Our versatile processor model has shown to be a viable alternative to traditional ISS-based co-simulation.

5.2 Contributions

In this thesis, we have presented an approach for efficient abstract modeling of MPSoCs. We have defined abstract modeling techniques and metrics for a rigorous quantitative analysis of abstract models. Our analysis results guide the developer of abstract models in which features to abstract. In addition, we provide guidance for the model user in selecting the most appropriate model for a given purpose. In particular, this thesis has contributed in three aspects (a) analysis of transaction level models for communication, (b) proposing a novel modeling technique, Result Oriented Modeling (ROM), and (c) in proposing abstraction of computation above the Instruction Set Architecture (ISA).

5.2.1 Analysis of Transaction Level Models for Communication

With respect to the analysis of transaction level models, we have contributed fourfold.

- We have classified TLM abstraction levels according to the granularity of data and arbitration handling.
- We have defined appropriate metrics and setups for a systematic quantitative analysis of TLM with respect to simulation performance and timing accuracy.
- We have quantified the TLM trade-off for three diverse communication protocols.
- We have generalized our observations from the analysis to guide communication model designers and model users.

5.2.2 Optimized Abstract Modeling Technique

We have contributed regarding an optimized modeling technique in the following ways.

- We have introduced a novel modeling concept, Result Oriented Modeling (ROM), which hides internal states, and uses an optimistic prediction and correction approach to reach accurate results at highest speed.
- Our ROM *eliminates* the TLM trade-off for a wide range of platforms, delivering highest speed and 100% accuracy at the same time.
- By introducing ROM, we have expanded the use of abstract modeling to domains that need 100% accuracy, e.g. real-time sensitive applications.

5.2.3 Abstract Processor Modeling

We have presented our abstract processor modeling approach for system-level design with the following contributions:

- We have introduced a versatile processor model that abstracts above the instruction set architecture for easy integration into system-level design environment for rapid design space exploration, debugging and development.
- We have analyzed the features for abstract processor modeling and characterized their influence to the trade-off between speed and accuracy.
- We have demonstrated significant benefits of our abstract processor modeling using industrial sized multi-processor heterogeneous examples.

5.3 Closing Remarks

Abstract modeling has emerged as a key aspect of system-level design. Fast and accurate models are essential for an early design space exploration, rapid prototyping, validation, debugging and development.

With the wide application of abstract models, a concise definition of proper abstraction levels is highly important. Furthermore, analyzing the effects of abstraction with regard to the TLM trade-off between simulation performance and simulation accuracy, is essential for guiding the developers and users of abstract models.

Overall, using well defined and efficient abstract models increases the design productivity and significantly impacts the design of digital embedded systems. It will ultimately result in more efficient products while reducing the time-to-market at the same time.

The techniques, presented in this dissertation, offer an interesting research platform. In particular, the modeling technique ROM can be extended to capture additional modeling aspects. First, with the increasing popularity of network on chip solutions, ROM can be extended to capture multiple homogeneous buses connected by routers, to optimize modeling communication across multiple hops. Expanding into modeling software execution, ROM can be applied to model preemptive scheduling on a processor, as an interrupt with subsequent scheduling decisions can be treated as a disturbing influence. This would greatly increase the accuracy of modeled interrupt latencies, which is often very important for embedded systems in a control context. Furthermore, the applicability of ROM to abstracting computation with fine granularity can be investigated, taking into account dynamic effects of architectural features.

Bibliography

- [1] Samar Abdi, Dongwan Shin, and Daniel D. Gajski. Automatic Communication Refinement for System Level Design. In *Proceedings of the Design Automation Conference (DAC)*, Anaheim, CA, June 2003.
- [2] Advanced RISC Machines Ltd. (ARM). ARM7TDMI (Rev 3) Product Overview.
- [3] Advanced RISC Machines Ltd. (ARM). SoC Developer with MaxSim Technology. <http://www.arm.com/products/DevTools/MaxSim.html>.
- [4] ARM. *AMBA Specification (Rev. 2.0)*, ARM IHI 0011A. Advanced RISC Machines Ltd. (ARM), 1999.
- [5] ARM. *AMBA AHB Cycle Level Interface (AHB CLI) Specification*, ARM IHI 0011A. Advanced RISC Machines Ltd (ARM), 2003.
- [6] Neil C. Audsley, Ken W. Tindell, and Alan Burns. The End of the Line for Static Cyclic Scheduling? In *Proceedings of the 5th Euromicro Workshop on Real-Time Systems*, pages 36–41, Oulu, Finland, 1993.
- [7] Jerry Banks, John Carson, Barry Nelson, and David Nicol. *Discrete-Event System Simulation*. Pearson, fourth edition, 2005.
- [8] Michael Barr. Embedded Systems Glossary. <http://www.netrino.com/Embedded-Systems/Glossary>, 2008.
- [9] Luca Benini, Davide Bertozzi, Alessandro Bogoliolo, Francesco Menichelli, and Mauro Olivieri. MPARM: Exploring the Multi-Processor SoC Design Space with SystemC. *Journal of VLSI Signal Processing*, 41(2):169–184, 2005.

- [10] V. Bhaskaran and K. Konstantinides. *Image and Video Compression Standards: Algorithms and Architectures, 2nd edition*. Kluwer Academic Publishers, 1997.
- [11] Bosch. *CAN Specification*. Robert Bosch GmbH, 2.0 edition, 1991.
- [12] A. Bouchhima, I. Bacivarov, W. Yousseff, M. Bonaciu, and A.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] Gunnar Braun, Achim Nohl, Andreas Hoffmann, Oliver Schliebusch, Rainer Leupers, and Heinrich Meyr. A Universal Technique for Fast and Flexible Instruction-Set Architecture Simulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 23(12):1625–1639, 2004.
- [14] Denny Brem and Dietmar Müller. Interface Based System Modeling of a CAN using SVE. In *Proceedings of the EkompasS Workshop*, Hanover, Germany, April 2003.
- [15] Joseph Buck, Soonhoi Ha, Edward A. Lee, and David G. Messerschmitt. Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems. *International Journal of Computer Simulation*, 4(2):155–182, April 1994.
- [16] Doug Burger and Todd 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] Lucai Cai, Andreas Gerstlauer, and Daniel D. Gajski. Multi-Metric and Multi-Entity Characterization of Applications for Early System Design Exploration. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, Shanghai, China, January 2005.

- [19] Harold W. Cain, Kevin M. Lepak, and Mikko H. Lipasti. A Dynamic Binary Translation Approach to Architectural Simulation. *ACM SIGARCH Computer Architecture News*, 29(1):27–36, 2001.
- [20] Marco Caldari, Massimo Conti, Marcello Coppola, Stephane Curaba, Lorenzo Pieralisi, and Claudio 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] Pramod Chandraiah, Hans Gunar Schirner, Nirupama Srinivas, and Rainer Dömer. System-On Chip Modeling and Design: A Case Study on MP3 Decoder. Technical Report CECS-TR-04-17, Center for Embedded Computer Systems, University of California, Irvine, June 2004.
- [22] Bob Cmelik and David Keppel. Shade: A fast instruction-set simulator for execution profiling. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Nashville, Tennessee, United States, 1994.
- [23] Marcello Coppola, Stephane Curaba, Miltos Grammatikakis, and Giuseppe 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.
- [24] CoWare. Virtual Platform Designer. <http://www.coware.com>.
- [25] M. Dales. *SWARM 0.44 Documentation*. Department of Computer Science, University of Glasgow, Nov. 2000.
- [26] Rainer Dömer. *System-Level Modeling and Design with the SpecC Language*. PhD thesis, University of Dortmund, Germany, April 2000.
- [27] Rainer Dömer, Andreas Gerstlauer, Junyu Peng, Dongwan Shin, Lukai Cai, Haobo Yu, Samar Abdi, and Daniel Gajski. System-on-Chip Environment: A SpecC-based Framework for Heterogeneous MPSoC Design. *EURASIP Journal on Embedded Systems*, 2008.

- [28] European Telecommunication Standards Institute (ETSI). *Digital cellular telecommunications system; Enhanced Full Rate (EFR) speech transcoding*, final draft edition, 1996. GSM 06.60.
- [29] Daniel D. Gajski, Jianwen Zhu, Rainer Dömer, Andreas Gerstlauer, and Shuqing Zhao. *SpecC: Specification Language and Design Methodology*. Kluwer Academic Publishers, 2000.
- [30] Michael Gasteier and Manfred Glesner. Bus-Based Communication Synthesis on System-Level. In *Proceedings of the International Symposium on System Synthesis*, San Diego, CA, USA, November 1996.
- [31] Andreas Gerstlauer. *Modeling Flow for Automated System Design and Exploration*. PhD thesis, University of California Irvine, U.S.A., April 2004.
- [32] Andreas Gerstlauer, Rainer Dömer, Junyu Peng, and Daniel D. Gajski. *System Design: A Practical Guide with SpecC*. Kluwer Academic Publishers, 2001.
- [33] Andreas Gerstlauer and Daniel D. Gajski. System-Level Abstraction Semantics. In *Proceedings of the International Symposium on System Synthesis*, Kyoto, Japan, October 2002.
- [34] Andreas Gerstlauer, Gunar Schirner, Dongwan Shin, and Junyu Peng. Necessary and Sufficient Functionality and Parameters for SoC Communication. Technical Report CECS-TR-06-01, Center for Embedded Computer Systems, University of California, Irvine, May 2006.
- [35] Andreas Gerstlauer, Gunar Schirner, Dongwan Shin, Junyu Peng, Rainer Dömer, and Daniel D. Gajski. System-On-Chip Component Models. Technical Report CECS-TR-06-10, Center for Embedded Computer Systems, University of California, Irvine, May 2006.
- [36] Andreas Gerstlauer, Dongwan Shin, Rainer Dömer, and Daniel Gajski. System-Level Communication Modeling for Network-on-Chip Synthesis. In *Asia and South Pacific Design Automation Conference*, Shanghai, China, January 2005.

- [37] Andreas Gerstlauer, Dongwan Shin, Junyu Peng, Rainer Dömer, and Daniel D. Gajski. Automatic Layer-Based Generation of System-On-Chip Bus Communication Models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 26(9):1676–1687, September 2007.
- [38] Andreas Gerstlauer, Haobo Yu, and Daniel D. Gajski. RTOS Modeling for System Level Design. In *Proceedings of the Design, Automation and Test in Europe (DATE) Conference*, Munich, Germany, March 2003.
- [39] Frank Ghenassia. *Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems*. Springer, 2005.
- [40] Chitra Giridhar. Trendy phones incorporate sophisticated engineering, EDN Asia. <http://www.edn.com/index.asp?layout=article&articleid=CA6290467>.
- [41] Tony Givargis and Frank Vahid. *Embedded System Design*. 2002.
- [42] Thorsten Grötter, Stan Liao, Grant Martin, and Stuart Swan. *System Design with SystemC*. Kluwer Academic Publishers, 2002.
- [43] David Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231–274, June 1987.
- [44] Florian Hartwich and Armin Bassemir. *The Configuration of the CAN Bit Timing*. Robert Bosch GmbH, 1999. <http://www.can.bosch.com>.
- [45] Anssi Haverinen, Maxime Leclercq, Norman Weyrich, and Drew Wingard. SystemC based SoC Communication Modeling for the OCP Protocol. October 2002.
- [46] Ken Hines and Gaetano Borriello. Optimizing Communication in Embedded System Co-simulation. In *Proceedings of Codes/CACHE*, Braunschweig, Germany, March 1997.
- [47] Shinya Honda, Takayuki Wakabayashi, Hiroyuki Tomiyama, and Hiroaki Takada. RTOS-Centric Hardware/Software Cosimulator for Embedded System Design. In

Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), Stockholm, Sweden, September 2004.

- [48] IBM. 128-bit Processor Local Bus Architecture Specification, SA-14-2538-04. http://www-306.ibm.com/chips/techlib/techlib.nsf/productfamilies/CoreConnect_Bus_Architecture, 2004.
- [49] Windriver Systems Inc. Wind River VxWorks Simulator. http://www.windriver.com/products/development_tools/ide/wind_river_vxworks_simulator.
- [50] International Organization for Standardization (ISO). *Reference Model of Open System Interconnection (OSI)*, second edition, 1994. ISO/IEC 7498 Standard.
- [51] Gilles Kahn. The Semantics of a Simple Language for Parallel Programming. In J. L. Rosenfeld, editor, *Information Processing '74: Proceedings of the IFIP Congress*, pages 471–475. North-Holland, New York, NY, 1974.
- [52] Torsten Kempf, Malte Dörper, Rainer Leupers, Gerd Ascheid, Heinrich Meyr, Tim Kogel, and Bart 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.
- [53] Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Applications*. Kluwer Academic Publishers, 1997.
- [54] Jean J. Labrosse. *MicroC/OS-II: The Real-Time Kernel*. CMP Books, 2002.
- [55] Kanishka Lahiri, Anand Raghunathan, and Sujit Dey. System-Level Performance Analysis for Designing On-Chip Communication Architectures. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, volume 20, pages 768–783, June 2001.

- [56] Marcello Lajolo, Claudio Passerone, and Luciano 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.
- [57] William LaRue, Sherry Solden, and Bishnupriya Bhattacharya. Functional and performance modeling of concurrency in VCC. *Concurrency and Hardware Design — Advances in Petri Nets, Volume 2549 of Lecture Notes in Computer Science / J. Cortadella, A. Yakovlev, G. Rozenberg (Eds.)*, pages 191–227, November 2002.
- [58] Edward A. Lee and David G. Messerschmitt. Synchronous Data Flow. *Proceedings of the IEEE*, September 1987.
- [59] Rainer Leupers, Johann Elste, and Birger Landwehr. Generation of Interpretive and Compiled Instruction Set Simulators. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, January 1999.
- [60] Paul Lieverse, Pieter van der Wolf, and Ed Deprettere. A Trace Transformation Technique for Communication Refinement. In *Proceedings of the International Symposium on Hardware-Software Codesign (CODES)*, Copenhagen, Denmark, April 2001.
- [61] Naraig Manjikian. Multiprocessor Enhancements of the SimpleScalar Tool Set. *ACM SIGARCH Computer Architecture News*, 29(1):8–15, 2001.
- [62] Grant Martin and Wolfgang Müller. *UML for SOC Design*. Springer, 2005.
- [63] Peter Marwedel. *Embedded Systems Design*. Kluwer Academic Publishers, 2003.
- [64] MathWorks, Inc. Simulink - Simulation and Model-Based Design. <http://www.mathworks.com/products/simulink/>.
- [65] Robert M. Metcalfe and David R. Boggs. Ethernet: Distributed Packet Switching for Local Computer Networks. *Commun. ACM*, 19(7):395–404, 1976.
- [66] Motorola. *MCF5206 ColdFire Integrated Microprocessor User’s Manual*, 1997.

- [67] Motorola Inc., Semiconductor Products Sector, DSP Division. *DSP56600 16-bit Digital Signal Processor Family Manual, DSP56600FM/AD*, 1996.
- [68] Wolfgang Müller, Rainer Dömer, and Andreas Gerstlauer. The Formal Execution Semantics of SpecC. In *Proceedings of the International Symposium on System Synthesis*, Kyoto, Japan, October 2002.
- [69] Achim Nohl, Gunnar Braun, Oliver Schliebusch, Rainer Leupers, Heinrich Meyr, and Andreas Hoffmann. A Universal Technique for Fast and Flexible Instruction Set Architecture Simulation. In *Proceedings of the Design Automation Conference (DAC)*, New Orleans, LA, 2002.
- [70] Object Management Group (OMG). OMG Systems Modeling Language (SysML). <http://www.omgsysml.org/>.
- [71] "Object Management Group (OMG)". Unified Modeling Language (UML). <http://www.uml.org/>.
- [72] Open SystemC Initiative, <http://www.systemc.org>. *Functional Specification for SystemC 2.0*, 2000.
- [73] Sudeep Pasricha, Nikil Dutt, and Mohamed Ben-Romdhane. Fast Exploration of Bus-based On-chip Communication Architectures. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Stockholm, Sweden, September 2004.
- [74] Junyu Peng, Samar Abdi, and Daniel D. Gajski. Automatic Model Refinement for Fast Architecture Exploration. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, Bangalore, India, January 2002.
- [75] Carl A. Petri. *Kommunikation mit Automaten*. PhD thesis, University of Bonn, Bonn, Germany, 1962.
- [76] Philips Semiconductors. P8xC592: 8-bit microcontroller with on-chip CAN. <http://www.semiconductors.philips.com>.

- [77] Philips Semiconductors. *The I²C-bus specification*, 2.1 edition, January 2000. document order number: 9398 393 40011.
- [78] Wei Qin and Sharad 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.
- [79] Mehrdad Reshadi, Prabhat Mishra, and Nikil 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.
- [80] Adam Rose, Stuart Swan, John Pierce, and Jean-Michel Fernandez. Transaction Level Modeling in SystemC. <http://www.systemc.org>, April 2005.
- [81] Alberto L. Sangiovanni-Vincentelli. Quo Vadis SLD: Reasoning about Trends and Challenges of System-Level Design. *Proceedings of the IEEE*, 95(3):467–506, March 2007.
- [82] Adriano Sarmiento, Wander Cesario, and Ahmed Amine 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.
- [83] Gunar Schirner and Rainer Dömer. Abstract Communication Modeling: A Case Study Using the CAN Automotive Bus. In A. Rettberg, M. Zanella, and F. Ramig, editors, *From Specification to Embedded Systems Application*, Manaus, Brazil, August 2005. Springer.
- [84] Gunar Schirner and Rainer 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.
- [85] Gunar Schirner and Rainer Dömer. Using Result Oriented Modeling for Fast yet Accurate TLMs. Technical Report CECS-TR-05-05, Center for Embedded Computer Systems, University of California, Irvine, May 2005.

- [86] Gunar Schirner and Rainer Dömer. Accurate yet Fast Modeling of Real-Time Communication. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Seoul, Korea, October 2006.
- [87] Gunar Schirner and Rainer Dömer. Fast and Accurate Transaction Level Models using Result Oriented Modeling. In *Proceedings of the International Conference on Computer Aided Design (ICCAD)*, San Jose, CA, November 2006.
- [88] Gunar Schirner and Rainer 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.
- [89] Gunar Schirner and Rainer Dömer. Result Oriented Modeling a Novel Technique for Fast and Accurate TLM. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 26(9):1688–1699, September 2007.
- [90] Gunar Schirner and Rainer Dömer. Introducing Preemptive Scheduling in Abstract RTOS Models using Result Oriented Modeling. In *Proceedings of the Design, Automation and Test in Europe (DATE) Conference*, Munich, Germany, March 2008.
- [91] Gunar Schirner and Rainer Dömer. Abstract Communication Modeling and Analysis. *ACM Transactions on Embedded Computer Systems*, accepted for publication August 23, 2007.
- [92] Gunar Schirner, Andreas Gerstlauer, and Rainer Dömer. Abstract, Multifaceted Modeling of Embedded Processors for System Level Design. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, Yokohama, Japan, January 2007.
- [93] Gunar Schirner, Andreas Gerstlauer, and Rainer Dömer. Automatic Generation of Hardware dependent Software for MPSoCs from Abstract System Specifications. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, Seoul, Korea, January 2008.

- [94] Gunar Schirner, Trevor Harmon, and Ray Klefstad. Late Demarshalling: A Technique for Efficient Multi-language Middleware for Embedded Systems. In *Proceedings of the International symposium on Distributed Objects and Applications (DOA)*, Larnaca, Cyprus, October 2004.
- [95] Gunar Schirner, Gautam Sachdeva, Andreas Gerstlauer, and Rainer Dömer. Modeling, Simulation and Synthesis in an Embedded Software Design Flow for an ARM Processor. Technical Report CECS-TR-06-06, Center for Embedded Computer Systems, University of California, Irvine, April 2006.
- [96] Gunar Schirner, Gautam Sachdeva, Andreas Gerstlauer, and Rainer Dömer. Embedded Software Development in an System-Level Design Flow: Case Study for an ARM Processor. In *Proceedings of the International Embedded Systems Symposium*, Irvine, CA, June 2007.
- [97] H. Schlingloff, M. Conrad, H. Dörr, and C. Sühl. Modellbasierte Steuergerätesoftwareentwicklung für den Automobilbereich. In *Automotive Safety and Security 2004 - Sicherheit und Zuverlässigkeit für automobile Informationstechnik*, Munich, Germany, October 2004.
- [98] SEMATECH Inc. International Technology Roadmap for Semiconductors (ITRS), 2004 Update, Design. <http://www.itrs.net/>, 2004.
- [99] SEMATECH Inc. International Technology Roadmap for Semiconductors (ITRS), 2007 Edition, Executive Summary. <http://www.itrs.net/>, 2007.
- [100] Marco Sgroi, Michael Sheets, Andrew Mihal, Kurt Keutzer, Sharad Malik, Jan M. Rabaey, and Alberto L. 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.
- [101] Robert Siegmund and Dietmar 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.

- [102] WindRiver Systems. VxWorks. <http://www.vxworks.com>.
- [103] SystemVerilog. <http://www.systemverilog.org>.
- [104] Ken Tindell, H. Hanssmon, and Andy J. Wellings. Analysing Real-Time Communications: Controller Area Network (CAN). In *Real-Time Systems Symposium*, December 1994.
- [105] Frank Vahid, Sanjiv Narayan, and Daniel D. Gajski. SpecCharts: A VHDL Frontend for Embedded Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 14(6):694–706, June 1995.
- [106] P. van der Putten, J.P.M. Voeten, M.C.W. Geilen, and M.P.J. Stevens. System level models for real-time communication. In *EUROMICRO*, September 1999.
- [107] VaST Systems. VaST Tools and Models for Embedded System Design. <http://www.vastsystems.com>.
- [108] Robert Walstrom, Joseph Schneider, and Diane T. Rover. Teaching System-Level Design using SpecC and SystemC. In *Proceedings of the Conference on Microelectronic Systems Education*, 2005.
- [109] Andreas Wieferink, Malte Doerper, Tim Kogel, Rainer Leupers, Gerd Ascheid, and Heinrich Meyr. Early ISS Integration into Network-on-Chip Designs. In *Proceedings of International Workshop on Systems, Architectures, Modeling and Simulation (SAMOS)*, Samos, Greece, July 2004.
- [110] Sungjoo Yoo and Kiyoun Choi. Synchronization Overhead Reduction in Timed Cosimulation. In *Proceedings of IEEE International High Level Design Validation and Test Workshop*, November 1997.
- [111] Haobo Yu, Rainer Dömer, and Daniel Gajski. Embedded Software Generation from System Level Design Languages. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, Yokohama, Japan, January 2004.

- [112] Jianwen Zhu and Daniel D. Gajski. A Retargetable, Ultra-Fast Instruction Set Simulator. In *Proceedings of the Design, Automation and Test in Europe (DATE) Conference*, Munich, Germany, March 1999.
- [113] Jianwen Zhu and Daniel D. Gajski. OpenJ: An Extensible System Level Design Language. In *Proceedings of the Design, Automation and Test in Europe (DATE) Conference*, March 1999.
- [114] Jianwen Zhu and Daniel D. Gajski. Compiling SpecC for Simulation. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, Yokohama, Japan, January 2001.
- [115] Vojin Zivojnovic, Steven Tjiang, and Heinrich Meyr. Compiled Simulation of Programmable DSP Architectures. In *Proceedings of the IEEE Workshop on VLSI Signal Processing*, Sakai, Japan, 1995.

Index

- Abstraction, 4, 33
 - Communication, 9, 22, 39, 49, 57, 70
 - Computation, 13, 25, 100
- Accuracy, 34
 - Analysis, 61, 83, 117
 - Metric, 35, 112
- AMBA, 38, 70
- Application Specific Integrated Circuit (ASIC), 2
- Arbitrated Transaction Level Model (ATLM), 32, 40, 49, 71
- ARM7TDMI, 111
- Behavior, 6, 101
- Bus contention, 36
- Bus cycle, *see* Granularity
- Bus Functional Model (BFM), 32, 40, 49, 71, 107
- Bus transaction, *see* Granularity
- Channel, 6, 101
- ColdFire master bus, 56
- Computer Aided Design (CAD), 5
- Controller Area Network (CAN), 47, 77
- Cumulative duration, 35
- Data Flow Graph (DFG), 14
- Delay model, 72
 - Dynamic, 72
 - Static, 72
- Design decision, 6
- Design process, 98
- DSP56600, 111
- Embedded system, 1
- Error distribution, 45
- Field Programmable Gate Array (FPGA), 2
- Finite State Machine (FSM), 14
- Finite State Machine with Datapath (FSMD), 14
- Firmware, 104
- Granularity, 30
 - Bus cycle, 31
 - Bus transaction, 30
 - User transaction, 30
- GSM, 110
- Hardware Abstraction Layer (HAL), 15
- Hierarchical Concurrent Finite State Machine (HCSM), 14
- Host compiled RTOS, 15, 26

- HW/SW Co-Design, 4
- Individual duration, 35
- Instruction Set Simulator (ISS), 14, 26
 - SoftwareARM (SWARM), 112
- Interrupt, 105, 106
- ISO/OSI layers, 10
- ISS-based cosimulation, 26, 108
- JPEG, 110
- Locked transfer, 39
- Matlab, 8
- Media Access Layer, 31
- Media access layer, 12
- Methodology, 5
 - Bottom-up, 5
 - Meet-in-the-middle, 6
 - Top-down, 5, 6, 98
- Model of Computation (MoC), 13, 14, 25
- MP3, 110
- Multi-Processor System-on-Chip (MP-SoC), 2, 111
- Performance
 - Analysis, 60, 85, 114
 - Metric, 33, 112
- Physical layer, 13, 31
- Pin- and Cycle-Accurate Model (PCAM), 18
- Processing Element (PE), 6
- Programming State Machine (PSM), 14
- Protocol layer, 13, 31
- Register Transfer Level (RTL), 4, 7
- Result Oriented Modeling (ROM), 67
 - Algorithm, 80
 - AMBA, 73
 - Analogy, 69
 - CAN, 78
 - Complexity, 92
 - Corrective measures, 67
 - Disturbing influence, 67, 74
 - Limitations, 92
 - Prediction, 67
 - Preemption, 74, 89
- RTOS, 15, 99, 102
 - MicroC/OS-II, 111
 - VxWorks, 15, 26
- Scheduling, 102
- Simulation bandwidth, 33
- Simulink, 8
- SpecC, 8, 82, 100
- SpecCharts, 14
- Specification model, 6, 16
- SysML, 8
- System Level Design Language (SLDL), 7, 21
- System-level Design, 3
- SystemC, 8
- SystemVerilog, 8
- Time-to-market, 2, 28

- Timing annotation, 102, 112
- TLM trade-off, 18, 28, 62, 65, 91, 95, 118
- Trace-based analysis, 22
- Transaction Level Model (TLM), 17, 32,
39, 49, 70, 105
- Unified Modeling Language (UML), 7
- Unlocked transfer, 39
- User transaction, *see* Granularity