

A UML BASED FRAMEWORK FOR TRANSACTION LEVEL MODELING

VAIBHAV JAIN



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY DELHI

APRIL 2015

©Indian Institute of Technology Delhi (IITD), New Delhi, 2015

A UML BASED FRAMEWORK FOR TRANSACTION LEVEL MODELING

by

VAIBHAV JAIN

Department of Computer Science and Engineering

Submitted

in fulfillment of the requirements of the degree of

Doctor of Philosophy

to the



Indian Institute of Technology Delhi
April 2015

Certificate

This is to certify that the thesis titled **A UML Based Framework for Transaction level Modeling** being submitted by **Mr. Vaibhav Jain** for the award of **Doctor of Philosophy** in Computer Science and Engineering is a record of bona-fide work carried out by him under our guidance and supervision at the Computer Science and Engineering Department, Indian Institute of Technology Delhi. The work presented in this thesis has not been submitted elsewhere, either in part or full, for the award of any other degree or diploma.

Anshul Kumar

Professor

Dept. of Computer Science & Engineering
Indian Institute of Technology Delhi
New Delhi, India 110 016

Preeti R Panda

Professor

Dept. of Computer Science & Engineering
Indian Institute of Technology Delhi
New Delhi, India 110 016

Acknowledgments

First of all, I wish to express my profound gratitude to my supervisors Prof. Anshul Kumar and Prof. Preeti R Panda for their continued encouragement and invaluable suggestions during this work. It has been an honor to work with them.

I would like to thank my colleague Aryabratt Sahu, Anant Vishnoi, Neeraj Goel, G Krishnaiah, BVN Silpa, Sharat Verma, Arun Parakh, Lava Bhargava and all the other colleagues for their cooperation and support. I wish to thank Prof. M Balakrishnan, Prof. G. S. Visweswaran and Dr. Kolin Paul for their valuable feedback and encouragement.

I would also like to thank John Aynsley from Doulos and Puneet Arora from Circuitsutra for their valuable feedback.

I would also like to thank the staff members of Computer Science department especially Mrs. Vandana Ahluwalia and Mr. S. D. Sharma for their help. They have opened their hearts and their doors, supplying me with resources I never could have found without them.

Finally, It could never been without my family support. I have faced real tough moments during my PhD tenure as I lost my loving father in a tragic accident and also lost my grandmother. With great love, patience and encouragement shown by my family including my wife Priyanka and mother.

Vaibhav Jain

Abstract

In recent time we have witnessed a rapid growth in the design of complex Systems-on-Chip devices (SoC). As the complexity of such systems is increasing, new methodologies are required to close the productivity gap for SoC design. Raising the abstraction level for SoC design using SystemC based Transaction Level Modeling (TLM) [1] has become an widely used solution. TLM allows HW/SW co-simulation and early verification using virtual system models of SoCs. For SoC designs at transaction level, models from various sources may need to be integrated which may raise the interoperability problems, leading to deadlock, loss of transactions, byte-ordering issues etc. TLM standard defines rules to ensure interoperability. However, TLM library does not provide any support for verifying these rules. On the other hand, manually debugging of interoperability errors raised by violation of TLM rules at application level is quite difficult. Therefore, automatic compliance checking is desirable. Compliance checkers [2] are used to monitor any violation of TLM rules during execution and find the exact source of error. Since these compliance checkers perform online checking, these are quite memory intensive and result in a lot of overhead during verification of TLM rules. Transactional models use a generic base protocol for communication between TLM components. When these virtual models require greater amount of accuracy then it is necessary to extend these models. TLM extension mechanism allows introduction of a more detailed protocol within the TLM framework. Compliance checking of such

models requires additional checkers for verification of the new protocol introduced.

On another front, Unified Modeling Language (UML) [3] with recent developments has shown a lot of potential towards the design of real-time and embedded systems. UML offers several diagrams like sequence diagram, state machines which have already found applications in requirement specifications, test benches and architectural/behavioral modeling. With UML 2 and later versions, UML can be tailored for modeling of a specific domain like SoC. Now, SoC designers are looking towards UML for improving the specification, design, implementation and verification processes. In recent time, several UML profiles have been proposed towards real-time and embedded systems like UML profile for SoC, System Modeling Language (SysML) etc. However, these profiles only help in modeling and to a certain extent code generation aspect.

This thesis is targeted towards utilizing UML model validation capabilities for verifying transactional models and showing UML potential beyond modeling and code generation abilities. We have proposed a UML based framework for TLM that allows modeling, code generation and verification of TLM rules and rules of other protocols built over TLM. The proposed framework includes a TLM profile which allows UML based Transaction level modeling and provides support for different coding styles. The profile also helps in verifying TLM static rules during model development. The verification of TLM rules during model development helps in detecting the modeling errors early. However, verification of TLM dynamic rules requires execution of the model. We have developed an offline strategy based on UML sequence diagrams derived from execution traces and have shown that it is more efficient than the usual approach of online verification during model execution. Next, we extended our framework by allowing compliance checking of user defined protocols. This is based on a more general dynamic rule checking strategy and support for checker generation from protocol rules.

Contents

List of Tables	v
List of Figures	vii
1 Introduction	3
1.1 Background	3
1.2 Motivation	4
1.3 Objective	5
1.4 Review of Previous Work	6
1.5 Overview of the Work Done	9
1.6 Thesis Organization	10
2 UML for SoC Design	13
2.1 Introduction	13
2.2 UML 2	16
2.3 UML profiles for Embedded System Design	18
2.3.1 SysML	18
2.3.2 UML Profile for MARTE	20
2.3.3 UML Profile for SoC	21
2.3.4 UML Profile for SystemC	23

2.4	Object Constraint Language	24
2.5	Summary	26
3	Transaction level Modeling and Compliance Checking	27
3.1	Introduction	27
3.2	TLM 2	29
3.2.1	Coding Styles	30
3.2.2	Temporal Decoupling	32
3.2.3	Direct Memory Interface	33
3.3	TLM Rules	34
3.4	TLM Compliance Checking	35
3.4.1	Related Work	36
3.4.2	TLM Protocol Checking	38
3.4.3	Experiment	39
3.5	Summary	40
4	UML Profile for TLM	41
4.1	Introduction	41
4.2	A SysML profile for TLM 2	43
4.2.1	Discussion	44
4.2.2	TLM Profile Structure	46
4.3	Case Study 1 : A Digital Photo Frame	48
4.3.1	Description	49
4.3.2	Modeling with TLM Profile	50
4.4	Case Study 2 : A Regular Mesh NoC	53
4.4.1	Description	53
4.4.2	Modeling with TLM Profile	54

4.5	Case Study 3 : A Simple SoC using Or1kSim	56
4.5.1	Modeling with TLM Profile	57
4.6	SystemC Code Generation	58
4.7	Summary	61
5	UML based TLM Rule Checking	63
5.1	Introduction	63
5.2	UML based Rule Checking	66
5.2.1	Understanding TLM semantics	67
5.2.2	Understanding OCL Constraints	70
5.2.3	TLM Static Rules Checking	74
5.2.4	Experiment	77
5.3	UML based TLM Dynamic Rule Checking	78
5.3.1	Sequence Diagram Representation	79
5.3.2	Sequence Diagram Generation	80
5.3.3	TLM Rule Checking using Sequence Diagram	81
5.4	Implementation	87
5.4.1	Calculating Protocol Checking Overhead	89
5.4.2	Results	90
5.5	Summary	93
6	Protocol specific Checking over TLM Models	95
6.1	Introduction	95
6.2	Proposed Strategy	97
6.3	Protocol Specification and Checker Generation	100
6.3.1	Generating Attribute Checks	100
6.3.2	Phase Transition Checks	103

6.3.3	Checker Strategy	107
6.4	Implementation	108
6.4.1	Generating Extensions and Protocol Trait Class	111
6.4.2	Benefits	112
6.5	Experimental Results	113
6.6	Summary	117
7	Conclusions	119
7.1	Research Contributions	119
7.1.1	UML Profile for TLM	120
7.1.2	TLM Dynamic Validation using UML	120
7.1.3	Protocol Specific Checking over TLM Framework	121
7.2	Future Research Directions	122
7.2.1	Enabling automatic TLM refinement	122
7.2.2	Incorporating model analysis capabilities	123
	Bibliography	125
	Appendix A	131
	Appendix B	137
	Appendix C	145
	Appendix D	149

List of Tables

2.1	SoC Profile Stereotype examples	22
3.1	TLM Rule Examples	35
3.2	TLM Rule Checking Report	39
4.1	Example Of Stereotype Definition	42
4.2	Lack of support for TLM Concepts in existing UML Profiles	43
4.3	TLM Profile Stereotypes	48
4.4	Code Generation Statistics for DPF and Mesh NoC	62
5.1	Examples of TLM Rules expressed as OCL Constraints	75
5.2	Result of Model Validation performed on a model shown in Fig. 5.3 . .	78
5.3	TLM rule types and checking strategies	83
5.4	Rule Checking on different TLM Models	90
5.5	Protocol checking overhead on DPF models	91
5.6	Comparison of TLM compliance checking strategies for Simple SoC Model	92
6.1	TLM Rules Types & their Representation	97
6.2	TLM Rules Types & their Representation	99
6.3	AMBA Rule specification using FOL expression	101
6.4	Example of single path for TLM Base Protocol	105

6.5	Example of multiple transition between two states	105
6.6	Protocol phase transition example	109
6.7	Attribute Initialization/Accessibility Rules	111
6.8	Comparison of SystemC & UML approaches for compliance checking .	116
6.9	Comparison of SystemC & UML approaches for compliance checking .	117
6.10	Comparison of TLM compliance checking strategies for Simple SoC Model	117
D.1	Predicates used for AMBA rule specification	150

List of Figures

2.1	Application of UML in SoC Design Flow [4]	14
2.2	Use of Meta-Object Facility Architecture	17
2.3	SysML Diagram Types	19
2.4	Architecture of MARTE Profile	20
2.5	Class Diagram for Cache memory using HRM Profile	21
2.6	Class diagram for a module using FIFO to send and receive data	23
2.7	SystemC profile stereotype examples	24
2.8	Composite Structure diagram for Counter system	25
3.1	Transaction models and role in system design [5]	28
3.2	TLM 2 Framework used for System Model [6]	30
3.3	Example of TLM Coding Styles	32
3.4	Temporal Decoupling Example	33
3.5	Existing Strategy for TLM Compliance Checking [6]	38
4.1	UML Profile Development Overview	42
4.2	Modeling difference between UML SystemC and proposed profile	45
4.3	TLM Profile Relationship with Other Profiles	46
4.4	Example of Stereotype declaration	47
4.5	Example of TLM Stereotypes Structure	49

4.6	TLM Model Example (A Digital Photo Frame)	50
4.7	Block Definition of DPF	50
4.8	Internal Block Diagram of DPF	51
4.9	Behavioral View of DPF representing CPU Initiator's Thread	52
4.10	4x4 regular mesh NoC architecture	53
4.11	Block definition of 4x4 mesh NoC	54
4.12	Internal block definition of a Router	55
4.13	StateMachine representing behavior of an Arbiter	56
4.14	Block definition of SoC	57
4.15	Internal Block definition of sc_top	58
4.16	SystemC Code Generation	59
4.17	Matching source tree node and constructing result fragment	60
4.18	A Template declared in XSL Stylesheet	61
4.19	Code Generation Example for Figure 4.9	61
5.1	OCL Type Hierarchy	71
5.2	Class Diagram of the part of UML metamodel	76
5.3	Example Model and another instance violating TLM Rules	78
5.4	UML Sequence Diagram depicting transactions of a TLM model	81
5.5	TLM dynamic rules verification	88
5.6	Comparison of TLM compliance checking strategies for DPF model	92
5.7	Comparison of TLM compliance checking strategies for 4x4 Mesh NoC	93
6.1	Proposed strategy for verification of TLM and user-protocol	98
6.2	Example of TLM extension for AMBA AHB Phase Transitions	99
6.3	Protocol Checker composition	100
6.4	Rule Checking over a TLM Framework	102