Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 250 (2009) 33–48

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Verifying Communication Protocols Using Live Sequence Chart Specifications

Rahul Kumar[1] Eric G Mercer[2]

Computer Science Department, Brigham Young University,Provo, Utah 84606, USA

Abstract

The need for a formal verification process in System on Chip (SoC) design and Intellectual Property (IP) integration has been recognized and investigated significantly in the past. A major drawback is the lack of a suitable specification language against which definitive and efficient verification of inter-core communication can be performed to prove compliance of an IP block against the protocol specification. Previous research has yielded positive results of verifying systems against the graphical language of Live Sequence Charts (LSCs) but has identified key limitations of the process that arise from the lack of support for important constructs of LSCs such as Kleene stars, subcharts, and hierarchical charts. In this paper we further investigate the use of LSCs as a specification language and show how it can be formally translated to automata suitable for input to a model checker for automatic verification of the system under test. We present the translation for subcharts, Kleene stars, and hierarchical charts that are essential for protocol specification and have not been translated to automata before. Further, we successfully translate the BVCI protocol (point to point communication protocol) specification from LSC to an automaton and present a case study of verifying models using the resulting automaton.

1 Introduction

System on Chip (SoC) designs are fast moving towards a development environment that incorporates third party Intellectual Property (IP) cores and blocks. Due to the use of such heterogeneous IP cores, multiple communication protocols are required to achieve the desired interactions, behavior, and functionality. With this diverse development environment comes not only the burden of verifying the system under development, but also the third party modules and communication protocols, to ensure the correctness and compliance of the complete system with respect to the system specification. This need is especially important for vendors looking to market and promote their products in new markets and development environments. To reduce the verification costs and redundancy of verification (verified twice: once

[1] Email:rahul@cs.byu.edu

[2] Email:egm@cs.byu.edu

doi:10.1016/j.entcs.2009.08.016

by the IP core developer and once again by the integrator), IP cores are often verified against commonly accepted standards and specifications to provide compliance results that can be easily utilized in an integration environment. A significant issue that hinders the process is the lack of an accepted specification language that can be formally integrated into the verification environment.

Traditionally, English has been used as the specification language for describing communication protocols. Due to the ambiguous and informal nature of English, in our experience, it has proven to be an inefficient specification language for use in a formal verification environment. Other specification languages based on temporal logic have also been used to specify correctness requirements of systems. Due to the complex nature of the temporal logics and the lack of support in all verification tools, these specifications tend to be limited in their use and applicability. Although specification patterns developed in the past do help, some aspects of creating a complete specification for an arbitrary verification are always unique and have to be created ground up; thus, re-enforcing the difficulty of using temporal logics as a specification language.

Other research has also investigated the use of graphical languages such as Live Sequence Charts (LSCs) for specifying communication protocols, and have reported positive and encouraging results [5]. We choose LSCs as our specification language because of their direct applicability to specifying communication protocols that primarily describe inter-process communication. Additionally, their graphical and intuitive nature makes them extremely usable for everyone involved in the development process and not only experts of formal verification.

In the past, LSCs have been used both as a specification and a modeling language. Because of their inherent ability to specify communication patterns without data information, we choose to use LSCs as a specification language describing the correctness requirements of a system. Previous work in the area of using LSCs as a specification language in a formal verification environment has been effective in exploring a verification approach but has failed to provide a comprehensive solution that supports the entire LSC grammar, which includes constructs such as Kleene stars, subcharts and hierarchical charts. We show how a communication protocol implementation can be formally verified against an LSC specification by providing translations of the entire LSC grammar to an automaton that is similar in nature to a *never claim* generated by SPIN [8]. This automaton can then be used directly as input to a model checker for verification of the system under test. Further, we provide a case analysis where the entire Basic Virtual Component Interface (BVCI) protocol is translated to an automaton and Promela models are verified against the translated automaton [5].

Using a graphical specification language targeted towards communication protocols provides the inherent advantage of rapid development of specifications that are intuitive and useful throughout the development cycle of the product. Since LSCs can be used both as a modeling and a specification language, they provide a common medium for verifying requirements as well as systems. Using the translation to automaton as presented in this paper, the specification can now be applied