Microprocessors and Microsystems 32 (2008) 288–295

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Graph based test case generation for TLM functional verification

Mohammad Reza Kakoee *, M.H. Neishaburi, Siamak Mohammadi

Nanoelectronics Center of Excellence, School of Electrical and Computer Engineering, University of Tehran, Iran

ARTICLE INFO

Available online 10 April 2008

Keywords:
TLM
Functional test case generation
SystemC
Verification
Coverage metrics

ABSTRACT

Describing complex systems at a high level of abstraction provides designers with the possibility of exploring multiple SoC design architectures before committing to the low level-details of a complete implementation. Transaction level modeling understandably expedites the design simulation and verification. During the verification process, generating good test cases plays a significant role in determining the quality of the design. Inadequate test cases may cause bugs to remain. In this paper, first, in order to generate test cases for a TL model, we present a Control-Transaction Graph (CTG) which describes the behavior of a TL Model. A Control Graph is a control flow graph of a module in the design and transactions represent the interactions such as synchronization between modules. Second, we define dependent paths (*DePaths*) on the CTG as test cases for a transaction level model, which can find communication errors in simulation. We also give coverage metrics for a TL model to measure the quality of the generated test cases. Finally, we apply our method on the SystemC model of AMBA–AHB bus and JPEG encoder and generate test cases based on the CTG of these models.

1. Introduction

Conventional RTL simulation is no longer fast enough for verification of today's complex system-on-a-chip (SoC) designs. Comprehensive verification of such sophisticated designs needs not only verification of the hardware implementation, but also verification of the embedded software. To achieve the necessary simulation performance, we must move to a higher, more abstract level of modeling. Transaction level modeling (TLM) [2,8] in SystemC [1] achieves 100x improvements in the simulation speed and enables architectural analysis and hardware/software co-verification.

Highly dependable construction of system level designs and proving their correctness are particularly important and arduous. Simulation is a practical and attractive method to achieve this goal. Although TLM expedites the verification of a hardware design [10,11,15], the problem of having high coverage test cases remains unsettled at this level of abstraction.

Admittedly, testbench development for SoC designs [7] requires even more effort than the design itself. The complexity of the code requires the use of a sophisticated software development environment.

To prove the correctness of a design by simulation, we must utilize all cases in the design input domain exhaustively. However, as this is not practical, we rather execute a test target program with test-data which are elements selected from a subset, according to some conditions on the input domain. These conditions are called test cases. Test cases play an important role in determining the quality of a design. They must be written carefully and if their number is not adequate, it is likely that some bugs will remain undetected in the design. In addition, test case overlaps would increase the verification cost.

To perform the verification in a minimum period of time, we need some metrics to detail the conditions. These metrics are called coverage metrics. At the evaluation step, by examining the coverage, we can determine how much of the design has been verified. By clearly setting up the coverage metrics before we start verifying a design, we can generate test cases that will satisfy these metrics.

1.1. Previous works and motivation

Taylor and Kelly proposed the concept of structural testing of concurrent programs [3] in software domain. They defined the concurrent state graph as a model of a concurrent program in software. Some EDA tools such as Specman elite [9] of Cadence provide test generation with assertion coverage. They offer a user defined constrained random simulation for RTL designs. Jindal and Jain in [5] describe a methodology for verification of TL models by using RTL Testbenches. Habibi et al. in [4] and [12] proposed a method based on FSM generation for TLM verification. However, they did not offer a methodology for generating test cases for it. Bombieri et al. [6] proposed a technique for reusing TLM testbenches at RT level with respect to both fault coverage and assertion coverage. Nevertheless, they used already written TLM

* Corresponding author.
E-mail addresses: kakoee@ieee.org (M.R. Kakoee), mhnisha@cad.ece.ut.ac.ir (M.H. Neishaburi), smohamadi@ut.ac.ir (S. Mohammadi).

0141-9331/$ - see front matter
doi:10.1016/j.micpro.2008.03.010

testbenches and did not explain how these testbenches are obtained. To the best of our knowledge, our previous work [15] is the first in generating test cases for functional verification of TL models. However, in that work we assumed that the TL model does not have any *wait statement* or *synchronization statement*. In this paper, we extend our graph to cover these kinds of statements and apply our method to a second case study.

The motivation behind this work is to investigate some techniques to provide high-quality test cases for the verification of an SoC in TLM domain. To show that these test cases are satisfactory, we define some coverage metrics. We introduce the Control-Transaction Graph (CTG) which is a directed graph to generate the appropriate test cases and coverage metrics.

1.2. Contribution

In this paper we propose a technique for generating test cases, which satisfies some defined coverage metrics, based on the SystemC code. We use the Control-Transaction Graph (CTG) as a model for Transaction Level designs. The CTG describes design modules as Control Graphs and Transactions between modules. We generate test cases based on the CTG. Some coverage metrics are defined based on this graph (i.e. CTG) to check the quality of test cases and to control the verification flow.

The rest of this paper is organized as follows: the next section represents the Control-Transaction Graph (CTG) as a model for transaction level designs. Section 3 describes our technique for generating test cases by using CTG. Section 4 defines coverage metrics for Transaction Level Models. Section 5 describes the algorithms which generate test cases on the CTG. Section 6 gives an outline of our test case generation tool. In Section 7 we apply our method on two real SystemC models as case studies. Finally, the last Section concludes the paper.

2. Representing a TL model by CTG

In this section we introduce the CTG describing the behavior of a transaction level model. A CTG includes two parts: Control Graphs and Transactions. Each Control Graph represents a module independently and transactions show the relations between modules.

2.1. Control Graph

A TL model consists of concurrent modules which communicate with one another. A Control graph (CG) represents the abstract control flow of a module in a TL model. Because each module is regarded as a sequential process, we can deduce a control flow graph from the source code. Each Node in the control graph denotes either a transaction statement or a flow control statement which includes transaction statements. Other statements such as assignment statements that contain no transaction statement are ignored in making the Control Graph. Edges in the Control Graph express control transfer between nodes.

Transaction statements characterize the concurrent behavior of a transaction level model. For example, in a SystemC TLM, transaction statements are such statements as "*put*" and "*get*" methods of *sc_interface*.

Since the nodes in the CG are just flow control statements or transaction statements, the number of nodes (|N|) in the control graph is much less than the number of statements in the code; therefore, the process of generating this graph from the code is not much time consuming. The formal semantic of a Control Graph is as follows:

$$CG \equiv \{N, E, s, f\},$$

where N is a set of nodes and E is a set of edges in CG. If $e = (u, v) \in E$ then $u, v \in N$. The start and final nodes are s and f, respectively. Since a TL model has multiple modules, it has multiple Control Graphs. We express a tuple of Control Graphs corresponding to a TL model T as $CG_s(T)$:

$CG_s(T) \equiv \{CG_i = (N_i, E_i, s_i, f_i) | 1 \leqslant i \leqslant \text{num}(T)\}$, where num($T$) denotes the number of modules in T.

2.2. Transactions in the CTG

Transactions in CTG represent the concurrent communications between any two modules and consequently any two CGs in a TL model.

Definition 1. Given modules M_x and M_y communicating with each other and control graphs CG_x and CG_y representing these modules, respectively. A set Syncs is defined as follows:

$$\text{Syncs}(CG_x, CG_y) = \{\text{sync} = (\alpha, \beta, I) | \alpha \in N_x, \beta \in N_y\},$$

where triplet (α, β, I) represents a simultaneous execution between two nodes $\alpha \in N_x$ and $\beta \in N_y$ with an identifier I.

Similarly we define two sets *Comms* and *Waits* as follows:

$$\text{Comms}(CG_x, CG_y) = \{\text{com} = (\alpha, \beta, J) | \alpha \in N_x, \beta \in N_y\},$$

where (α, β, J) represents communication from α to β with an identifier J.

$$\text{Waits}(CG_x, CG_y) = \{\text{Wait} = (\alpha, \beta, K) | \alpha \in N_x, \beta \in N_y\},$$

where (α, β, k) represents the possibility of α waiting for β with an identifier K. "Wait" has two states: for waiting and for not waiting.

Definition 2. Consider the definitions of Syncs, Comms and Waits as sets of all triplets of simultaneous executions, communications and waits, respectively in a TL model as follows:

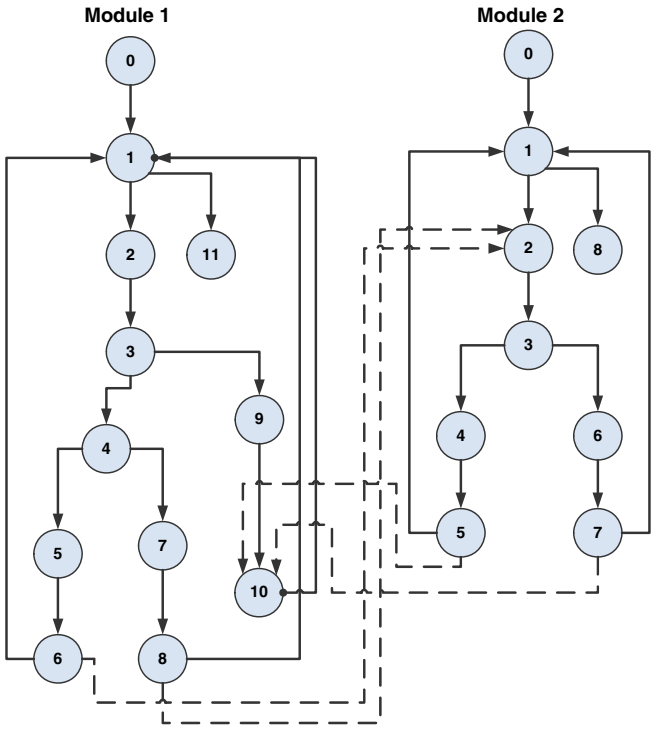

Fig. 1. Graphical form of a CTG sample.