IST-2000-29212

CoTest

COTEST *- New Techniques for Supporting Testability in Co-design Environments*

COTEST/D1

Report on benchmark identification and planning of experiments to be performed

Matteo Sonza Reorda, Massimo Violante

Politecnico di Torino
Dipartimento di Automatica e Informatica
Torino, Italy

Gert Jervan, Zebo Peng

Linköpings Universitet
IDA/SaS/ESLAB
Linköping, Sweden

Contact person:
Matteo Sonza Reorda Dipartimento di Automatica e Informatica Politecnico di Torino C.so Duca degli Abruzzi, 24 I-10129 Torino TO Italy
Tel. +39 011 564 7055 Fax. +39 011 564 7099 E-mail: sonza@polito.it

Abstract

The document describes the benchmarks we have identified as test cases to be used during the COTEST project. Being the project focused both on the high-level generation of suitable test/validation vectors and on the high-level insertion of design for testability structures, we identified benchmarks of different characteristics and complexity. The document also outlines the experiments that we intend to perform during the project.

Related documents

COTEST Technical Annex

Table of Contents

1. Introduction

This report descibes a set of suitable test cases to be used for validating on meaningful examples the methodology developed during the process.

Being the project focused on two complementary aspects of test at the high level (vector generation and DfT strucures insertion), we developed two different classes of benchmarks.

The guidelines we followed during the benchmark selection process are outlined in the following sub-sections. Moreover, for each of the selected benchmarks, we also describe the experiments we intend to perform during the project, as well as the figures we intend to measure in order to quantify the effectiveness of the techniques developed during the project.

1.1. High-level vector generation

As far as the high-level test vectors generation is concerned, we selected two benchmarks, one representative for the class of data-dominated systems and one for that of control-dominated systems. The common denominator of these systems is the relatively small size in terms of lines of code describing the models, but the high complexity both in terms of circuit architecture and sequential depth. On the one hand, the limited size of the benchmarks simplifies the task of analyzing their structure in order to extract meaningful information for vector generation. On the other hand, the architectural and sequential complexity make vector generation non-trivial, and thus provides a suitable test bench for the algorithms developed during the COTEST project.

The two benchmarks we identified are described in section 2.

1.2. High-level design for testability structure insertion

As far as the high-level design for testability (DfT) structure insertion is concerned, we need to identify a benchmark whose high complexity justifies the adoption of design for testability structures. Moreover, the following criteria should be met:

- The system should be heterogeneous, thus comprising a wide set of modules requiring different DfT solutions. Such a kind of system allows validating the effectiveness of different DfT solutions, as well as the possibility of validating the coexistence in the same system of heterogeneous DfT solutions.
- The system should allow applying to each module several DfT solutions. By analyzing the effectiveness of each DfT solution on a single module and by relating their effectiveness with the overhead they introduce, the designer can gain a picture of the effectiveness of the DfT solutions developed during the project as well as a good

understanding of the overhead their requires. Possible results of this analysis are trade-off curves relating DfT effectiveness with resource overhead.

- The system should allow the implementation of alternative system-wide DfT schemes, in order to reason about DfT effectiveness and resource overhead at the system level.

The benchmark we identified accordingly to the aforementioned guidelines is described in section 3.

2. High-level vector generation

The purpose of this section is to describe the selected benchmarks and the design flow that is used in our experiments. According to the previously described guidelines, we selected one benchmark for the class of control-dominated applications (which is described in sub-section 2.1) and two for that of data-dominated applications (which are described in sub-section 2.2). The limited size of the selected benchmarks in terms of number of lines of code is motivated by the fact that, during the assessment phase, most of the techniques developed during the project will be applied by hand.

2.1. Control-dominated application

As an example of control-dominated applications we adopted a behavioral description of the well-known *Traffic Light Controller* (TLC) benchmark coming from the HLSynth´91 benchmark suite.

The TLC benchmark, whose interface is described in Figure 1, is described as a Finite Sate Machine (FSM) composed of 5 states and 9 state transitions and amounts to about 80 lines of behavioral-level VHDL code. As our experience confirms, this model closely matches the building blocks of more complex systems, where several small FSMs communicate. It is indeed common in designing complex control machines to partition them in simpler sub-machines, each one in charge of one atomic task, i.e., a task than cannot be further partitioned in simpler tasks.

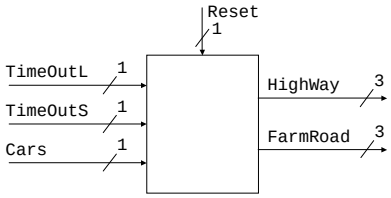

Figure 1: The TLC benchmark interface

Furthermore, the TLC benchmark is modeled according to a description style which closely resembles that used in describing the aforementioned atomic tasks: the FSM has an

initialization state in charge of managing the set-up of the machine and then few additional states describes the FSM reactive behavior. The state transition graph of this kind of machine is usually loosely connected: few transitions are allowed among all the possible ones, and state transitions occur only upon the reception of external events. The state transition graph of the TLC benchmark is depicted in Figure 2; for the sake of simplicity, we do not report the input events producing transitions between states and the output values the FSM produces.

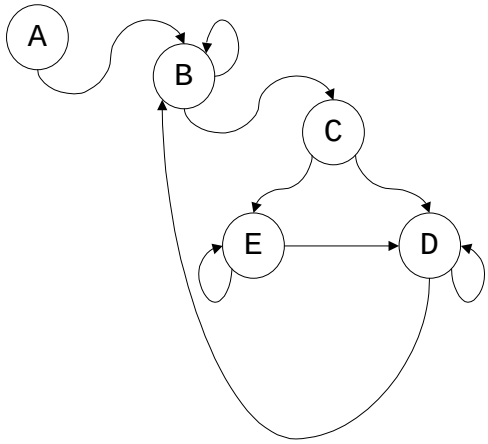

Figure 2: The TLC state transition graph

The experiments we intend to perform on the TLC benchmark are the following:

- *Synthesis*: an implementation of the benchmark will be designed by exploiting an automatic synthesys tools.
- *Gate-level ATPG*: the gate-level implementation will be analyzed in terms of testability by means of a gate-level ATPG. These experiments are intended in order to measure the fault coverage a traditional gate-level ATPG is able to attain as well as the test length in terms of number of vectors the gate-level ATPG computes, and CPU time for test vectors generation.
- *Behavioral-level ATPG*: the algorithms devised during the project will be applied in order to compute test vectors by reasoning on the benchmark behavior, only. The test vectors obtained during each experiment will then be evaluated by measuring the fault coverage they attain when applied to the aforementioned implementation of the benchmark.

2.2. Data-dominated application

As examples of data-dominated applications we adopted a biquadratic filter (BIQUAD) with programmable parameters and input synchronization and the DIFFEQ benchmark coming from the High Level Synthesis'92 benchmarks suite.

The behavioral description of the BIQUAD benchmark amounts to about 100 lines of behavioral level VHDL code and exploits 5 fixed-point multiplication operators and 4 fixed-point sum operators, plus a number of assignment and shift operators.

The block diagram of the benchmark, as well as its interface, is depicted in Figure 3.

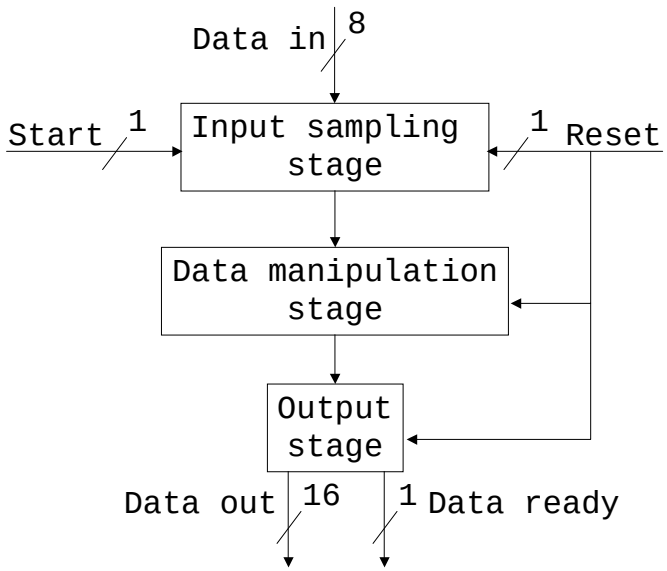

Figure 3: Block diagram of the BIQUAD benchmark

Two aspects of the BIQUAD benchmark are worth being highlighted:

- Input samples and parameter values must be provided to the model according to a given synchronization scheme. Any functional vector generation algorithm should thus be able to match this synchronization scheme in order to gain full control over the module behavior. By exploiting information gathered during the analysis of the high-level model of the benchmark behavior we can simplify the task of understanding the synchronization mechanism. We can thus save processing time that is likely to be lost in case this analysis is performed while at the gate level.
- Due to the high number of operators the model exploits and to its synchronization mechanism, several synthesis options are available. By tuning the level of resource sharing allowed by the synthesis algorithm, different implementations of the given behavior are possible. Moreover, the adoption of different sized pipelines increases the range of possible implementation strategies. These alternatives pose a challenging question about the effectiveness of the vectors generated while at the high-level, where only the behavior is known, and few details on the model implementation are available.

The behavioral description of the DIFFEQ benchmark amounts to about 60 lines of behavioral level VHDL code and exploits 6 fixed-point multiplication operators and 4 fixed-

point sum operators, plus a number of assignment operator. The data-path of the benchmark is depicted in figure 4.

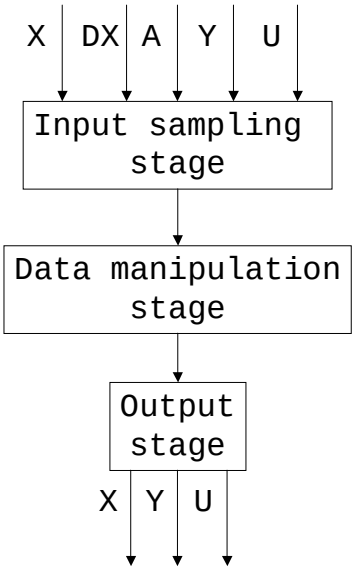

Figure 4: Block diagram of the DIFFEQ benchmark

The experiments we intend to perform on the BIQUAD and DIFFEQ benchmarks are the following:

- *Synthesis*: alternative implementations of the benchmarks will be designed, by tuning the following parameters:
  - Area: the behavioral-level synthesis tool will be constrained in adopting different levels of resource sharing, thus obtaining different implementations which trade off circuit area with number of control states.
  - Latency: we will let the high-level synthesis tool select the optimal trade-off between resource occupation and latency in order to minimize both.
- *Gate-level ATPG*: as in the previous case, the alternative gate-level implementations will be analyzed by means of a gate-level ATPG. The attained fault coverage, the test length and CPU time for test vectors generation will be measured.
- *Behavioral-level ATPG*: the algorithms devised during the project will be exploited in order to compute test vectors by reasoning on the benchmark behavior, only. The test vectors obtained during this experiment will then be evaluated by measuring the fault coverage they attain when applied to the aforementioned alternative implementations of the benchmark.

2.3. Experimental flow

The flow to be adopted during the experiments is shown in Figure 5.

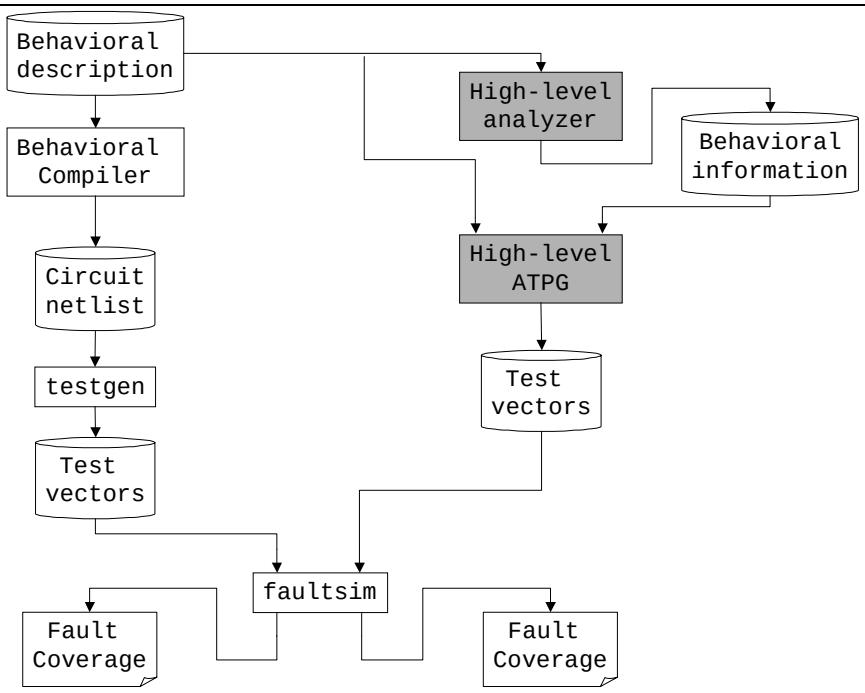

Figure 5: The experimental flow

On the left-hand of Figure 5, the traditional approach to test vector generation is reported. Starting from a behavioral description of the circuit, the Behavioral Compiler tool by Synopsys will be used to generate a netlist implementing the desired behavior. Then, the Synopsys gate-level ATPG (testgen) will be used to compute a set of test vectors. Finally, the Synopsys fault simulator (faultsim) will be used to evaluate the attained fault coverage.

On the right-hand of Figure 5, the behavioral-level test generation process analyzed by the COTEST project is reported. The process is based on two modules:

- *High-level analyzer*: it analyzes the behavioral description of the circuit and derives useful information that will be exploited by the high-level ATPG. Examples of the information it provides are the state transition diagram of the behavioral circuit description, and the latency of the input/output stages of the model. Additional information provided by the user can be exploited, e.g., the latency of the model in terms of number of stages composing the pipeline that the designer intends to use in the gate-level circuit implementation.
- *High-level ATPG*: it generates test vectors starting from the behavioral-level model of the circuit and the information the high-level analyzer provides. In order to

successfully develop an effective test generation tool two key issues have to be addressed:

- *High-level fault model*: it is the one adopted by the test vector generation algorithm and it has to be both suitable for application on the behavioral-level model of the circuit and representative of actual faults affecting the gate-level circuit implementation.
- *Test generation algorithm*: it is the algorithm used for derive test vectors. In order to prove the feasibility of high-level test generation is has to provide test vectors attaining the highest fault coverage as far as the high-level fault model is considered.

At the bottom of Figure 5 the approach we intend to adopt for assessing the effectiveness of the high-level test generation approach is reported. The test vectors coming from the two ATPGs, the gate-level and the high-level ones, are fault simulated by means of the same tool. The two obtained fault coverage figures are comparable, since generated by the same tool working on the same circuit and by using the same fault list. Therefore, by using the fault coverage attained by the gate-level tool as a reference, we can evaluate the effectiveness of the high-level generated vectors.

3. High-level design for testability insertion

The purpose of this section is to describe the selected benchmark and the experiments we intend to perform on it. As it was described in Section 1, the benchmark design should be heterogeneous and relatively complex to justify the adoption of DfT structures and to evaluate the effectiveness of different DfT solutions.

3.1. The F4/F5 model

In order to provide a benchmark to be used during the high-level DfT insertion and evaluation, we developed a model of the F4/F5 block of the ATM protocol, covering the block functionality as specified by standard references. The model consists of several processes, each implementing a relatively independent part of the block functionality. The model handles multiple Virtual Paths (VPs) and Virtual Channel (VC) connections passing through the block. The management of ATM cells depends on the current "status" of the virtual path or virtual channels, and thus local tables are used for recording the global state of the block.

One of the basic ideas with ATM is that all signaling will be sent on the same physical carriers as the user data. To handle this there is a need for special units, which introduce and extract this information in the nodes of the network. The F4/F5 block covers the Operation and Maintenance (OAM) functionality of the ATM switches.

The name of the F4/F5 block refers to the levels of OAM functionality that the block covers. The F4 level handles the OAM functionality concerning virtual paths and the F5 level handles the OAM functionality concerning virtual channels. The levels F1 to F3 handle OAM functionality at the physical layer of the protocol. The structure of the F4/F5 model is depicted in Figure 6.

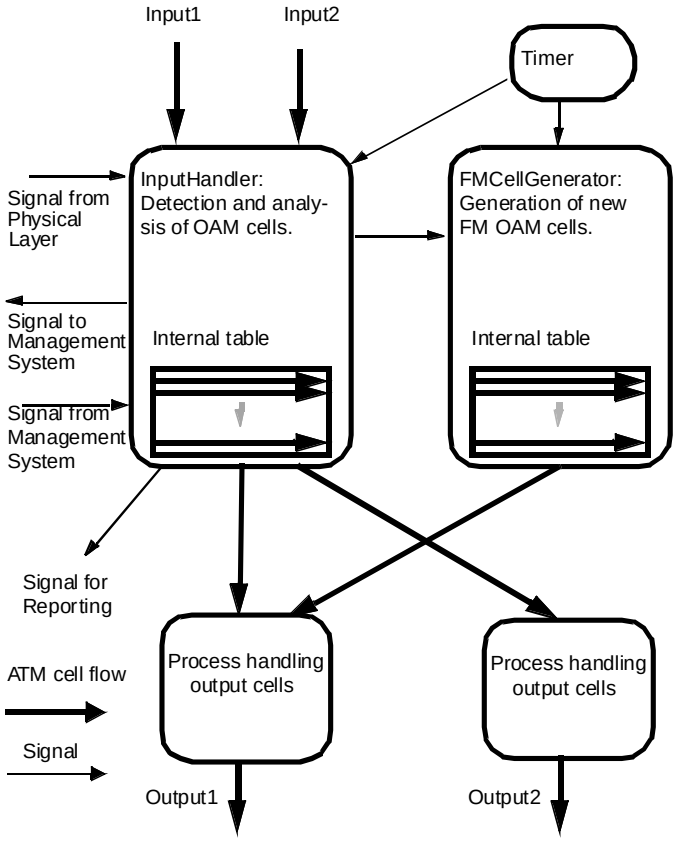

Figure 6: The structure of the F4/F5 model

The F4/F5 block has the following main functionalities:

- *Fault management*: when the appearance of a fault is reported to the F4/F5 block, special OAM cells will be generated and sent on all affected connections; if the fault persists, the management system should be notified.
- *Performance monitoring*: normal functioning of the network is monitored by continuous or periodic checking of the transmission of cells.
- *Fault localization*: when a fault occurs it might be necessary to localize it further. For this purpose special loop-back OAM cells are used.
- *Activation/deactivation*: a special protocol for activation and deactivation of OAM functions that require active participation of several F4/F5 blocks, e.g., performance monitoring, has to be implemented.

An F4/F5 block is present on each physical link connected to the ATM switch. Since all connections are bi-directional, the F4/F5 block has two inputs and two outputs.

The ATM benchmark is composed of 4 synchronized finite state machines whose behavior is described in VHDL language. Communication among the modules composing the system is implemented by resorting to two alternative solutions: by using signals or by using send

and receive primitives. The former approach is open to a simpler implementation of the benchmark, while the second one allows easily evaluating different communication solutions. The model amounts to about 1,300 lines of code.

Although the model is complex, it is composed of relatively independent parts. Such a model allows us to extract smaller parts of the system for initial analysis without falling into the complexity trap. At the latter stage of the project we will possibly be able to analyse the applicability of our methods to more complex systems by using the complete design.

3.2. Planned experiments

The experimental work we intend to perform will focus on modifications of system descriptions to support design for testability techniques at the system level. The first and the most important task is to find a suitable way to describe DfT structures at the system level. To perform this task we intend to isolate one or several modules in the benchmark design and to identify hard to test parts in them by using high-level vector generation approach. During the next stage we will apply DfT modifications to the initial specification and evaluate the effectiveness of the final solution (in terms of required area, time and fault coverage improvement). For evaluation purposes we eventually have to synthesise the design to the gate-level and to apply classical gate-level ATPG/DfT tools to demonstrate the improvement in the gate-level fault coverage or reduction in the ATPG/DfT tool CPU time.

As it is described in chapter 2.3 for the high-level vector generation, the reference design flow for WP 3 also contains two parts (see Figure 6). On the left side there is a traditional design flow consisting of high-level and logic synthesis tools and DfT modifications at the gate-level. The right side of the design flow is similar, but different in case of high-level DfT insertions. It starts from DfT modifications at the high-level. After appropriate synthesis steps the DfT modifications on a gate-level can be applied again. The objective of the project is to evaluate whether that the new design flow, where the DfT modifications were introduced at the high-level is able to reduce the gate-level DfT effort and hardware overhead as well as test time (and/or size of test patterns).

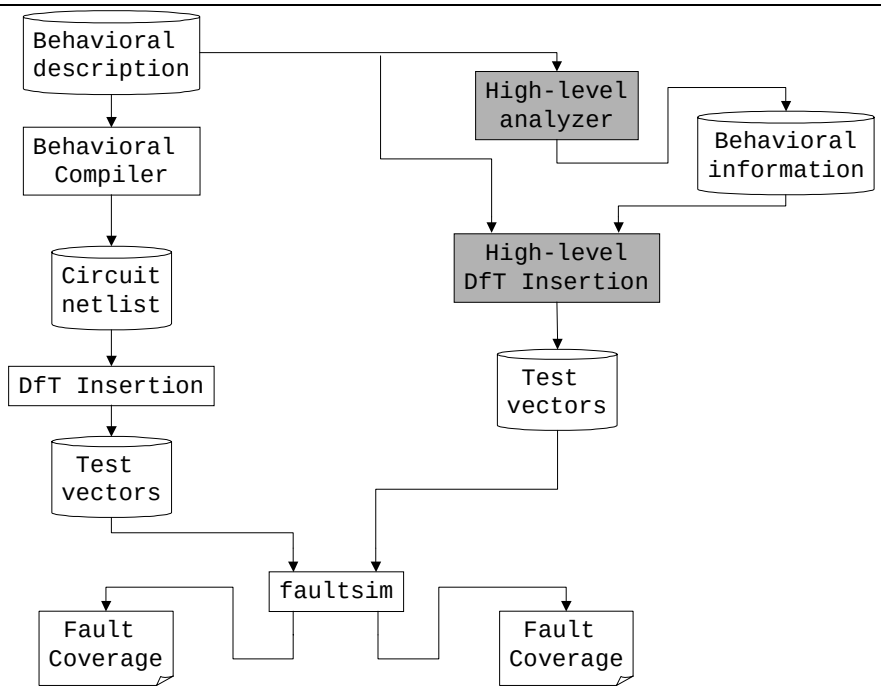

Figure 6: The experimental flow