Pseudo-Random Behavioral ATPG

Anne-Lise COURBIS  Jean-Francois SANTUCCI
LG12P
EMA-EERIE
Parc Scientifique G.Besse
30000 Nimes
FRANCE
Tel : (33) 66 38 70 64  Fax : (33) 66 38 70 74
E-Mail : {courbis,santucci}@eerie.fr

Abstract
This paper deals with a new approach for the Automatic Test Pattern Generation of circuits described from a behavioral point of view in VHDL. This approach is based on a pseudo-random process characterized by the fact that criteria for computing the test length and evaluating the quality of the generated data come from the field of Software Engineering. This paper presents the bases of this new approach in the field of Hardware Engineering and some experimental results.

1: Introduction

Behavioral Automatic Test Pattern Generation (BATPG) has appeared as a new approach for testing digital circuits the structural implementation of which is unknown and which are described at high abstraction levels from a behavioral point of view. Most BATPG methods [1, 2, 3, 4] are based on a deterministic generation process using a formal fault model: each test sequence is constructed in order to detect a given fault belonging to a Fault List constructed from a Behavioral Fault Model. An another deterministic approach is proposed in [5]: test patterns are generated not to detect faults but to meet criteria of software engineering. Despite the use of heuristic criteria [5, 6, 7] to speed up the search for test sequences, these processes remain very time-consuming because of their deterministic nature.

In order to speed up the generation process, we present in this paper a new approach for BATPG based on a pseudo-random process. Such a process is usually performed for testing circuits at the gate level. The goal is to randomly generate test sequences using a process which is not time-consuming. The approach is successful if the random generation process allows the Fault List to be drastically reduced. Only faults which are not detected by the random sequences will be subjected to a deterministic generation process. Generating random sequences causes two basic problems: on the one hand, it is necessary to define the length of the random test sequence; on the other hand, it is necessary to define criteria which express the "quality" requirements that sequences have to fulfill. To solve both these problems, we have been concerned with testing techniques developed in the field of Software Engineering.

Having selected criteria from the field of Software Testing allowing the two aforementioned problems to be solved, we have studied how such criteria could be measured and applied to behavioral descriptions in order to generate pseudo-random test sequences. This study has resulted in the development of a technique supported by a tool allowing pseudo-random test patterns to be generated for behavioral descriptions written in the standard VHDL. The efficiency of this technique is addressed from an experimental point of view: test sequences are randomly generated for a set of benchmarks and the number of behavioral faults detected by these sequences is measured.

This paper is divided into three parts. In the first part, we present the pseudo-random generation process and criteria from the field of software engineering which have been selected for determining the test length and the requirements that sequences have to fulfill. The second part is devoted to the implementation of the pseudo-random generation process. Experiments which have been conducted in order to evaluate the efficiency of the pseudo-random generation are summarized in the third part.

1: The pseudo-random generation process

1.1: Principle

As has already been mentioned, the pseudo-random test pattern generation is an inexpensive way to generate test sequences. By definition, in the pseudo-random method, tests are generated through a quite random process guided by two types of information: the number of patterns which have to be generated and criteria that the patterns have to
fulfill. This information being determined, the main steps of
the pseudo-random generation consists in generating the
aforementioned number of random patterns and running the
behavioral description with the random patterns. If the
predefined criteria are not fulfilled, new random patterns
must be generated and applied to the behavioral description
until criteria are met.

In order to find criteria which could estimate the length
of test sequences and express the quality of test sequences,
we have been concerned with techniques developed in the
field of Software Engineering. Dynamic software testing
[9,10] consists in running the software with a set of test
data and check through an instrumentation if given criteria
are met. These criteria are defined according to the control
structure of the software being tested which is modeled as a
graph. They can be divided into two categories according
to the family of the metrics used: coverage-based metrics and
complexity-based metrics. In the next two parts, we will
present these metrics pointing out that they are suitable for
solving problems set up by random test generation. These
two criteria have been selected because they are defined on a
formal model which is graph based and independent of the
programming language of the software being tested.
Moreover, their evaluation can easily be automated.

1.2: Complexity-based metrics for determining the
length of test sequences

Complexity-based metrics have been first proposed to
evaluate the complexity of software in order not to design
software which cannot be easily tested. An interesting
application of the complexity metrics to the testing activity
was developed in [11]. Like most metrics, the McCabe
metric is based on a graphical representation of the control
part of the software being tested. Let G be a graph with n
nodes, e edges and p sub-programs. McCabe defined the
cyclomatic number of a graph G associated with the
control part of software, which represents the number of
linearly independent paths of graph G as being: \( V(G) = e - n + p \). He defined the cyclomatic complexity as being:
\( CV(G) = e - n + 2 p \). He proved that the cyclomatic
complexity is similar to a path coverage and represents the
minimum number of test data to be generated to test the
control part of software.

1.3: Coverage-based metrics for determining the
quality of test sequences

Most coverage metrics are based on different criteria
which are used both to evaluate the test data and to help in
the generation of the test data. In this part, we present the
conventional coverage metrics used for evaluating test data
quality [10]. A simple example is shown in Figure 1(a) to
illustrate the different criteria introduced. In Figure 1(b),
the different test cases are listed and we will show in the
following which test case has to be applied to cover a given
criterion.

\[
\begin{align*}
&\text{begin} \\
&\text{if } ((A < B) \text{ and } (C = 5)) \\
&\quad \text{then} \\
&\quad C <-- 4 \\
&\quad \text{endif;} \\
&D <-- 5
\end{align*}
\]

(a)

(b)

**Figure 1**

(a) A description to be tested and
(b) all possible test cases

. The simplest criterion, called linear criterion,
consists in evaluating if any statement of the software has
been exercised at least once. According to the example in
Figure 1(a), the alone test case C1 expressed in Figure 1(b)
has to be applied to cover the linear criterion.

. The second family of criteria aims at evaluating the
coverage of all possible execution paths. As such coverage
cannot be obtained, the structure of the software is
partitioned into sub-paths and the coverage is evaluated
according to these sub-paths. Two kinds of coverage criteria
have been defined depending on the type of path coverage:
the branch coverage and the condition coverage. The
branch coverage requires every possible outcome of all
decision points to be exercised at least once. For the
example in Figure 1(a), this criterion is covered by the test
cases C1 and one of the other test cases expressed in Figure
1(b). The condition coverage requires that each condition in
decision statement take all possible outcomes at
least once. To cover this criteria, all the test cases
expressed in Figure 1(b) have to be exercised.

2: Implementation

Our goal is to generate test patterns of circuits from
their VHDL behavioral descriptions. According to the
approach selected for generating test patterns, the main
problem is to develop a Software Analyzing Tool allowing
the cyclomatic complexity and the coverage metrics to be
evaluated for a set of test patterns. Such a tool has never be
developed for VHDL programming languages. Nevertheless,
they have been developed for conventional languages such as
Fortran, C or Ada. In order to evaluate the efficiency of
our new approach, we have decided to use an existing
Software Analyzing tool : Logiscope [12]. Logiscope
allows instrumentation or software probes in the form of
source language statements to be inserted into the software
being tested in order to collect statistics and coverage during
its execution. Thus the use of an existing Analyzing Tool
implies that a language must be chosen into which VHDL
descriptions will be translated. We have chosen Ada because
Ada allows concepts of VHDL (such as concurrent
processes, specific data types) to be easily translated. The overview of the pseudo-random generation generation is described in Figure 2. In the first step, the VHDL description is translated into ADA. In the second step, the Complexity Analyzer of LOGISCOPE is performed allowing the number of generated test sequences to be computed. The random generation process is then applied during step 3: it results in a package of test sequences expressed in ADA. These sequences are submitted in step 4 to the Dynamic Analyzer of LOGISCOPE allowing their coverage to be estimated. The random process will be performed until the coverage is declared as being acceptable (step 5). Finally, the set of generated test sequences are translated into VHDL (step 6).

3: Efficiency of the pseudo-random generation process.

The Pseudo-Random Test pattern Generation (PRTG) process presented in Section 2 allows Test Patterns to be generated from Behavioral Descriptions written in the standard VHDL. In this section, we present the approach which has been used to highlight the efficiency of the Pseudo-Random Generation Process. In the first subsection, we present the fault modeling scheme which has been chosen. The second subsection is devoted to the experiments which have been carried on a set of Behavioral Descriptions. For each Behavioral Description belonging to this set of descriptions, the pseudo-random process has been performed and evaluated by determining the percentage of behavioral faults detected.

3.1: Behavioral fault modeling.

The generic Behavioral Fault Model which has been taken into account for carrying out experiments is defined in [13]. Generic faults belonging to this model are of four types:

- . The value attribute of a data is stuck at one of the lower or upper extremes of its definition domain.
- . The output of an operation or a function may fail such that it permanently returns v1 or v2, where v1 and v2 express the lower and upper extremes of the range of the operation.
- . A wrong path is selected in a selective or repetitive control structure.
- . A process is always or never activated independently of the value of its sensitive signals.

![Figure 2](image)

The Pseudo-random Generation Process Using Logiscope
3.2: Experiments and results.

Five medium-size VHDL behavioral descriptions have been selected for validating the pseudo-random generation. Results of the pseudo-random process are analyzed with respect to its time consumption and the percentage of behavioral faults detected. In the best case, which corresponds to the smaller description (4-bit adder), 95% of behavioral faults are detected. In the worst case, which corresponds to the bigger description (an ALU), 91% of behavioral faults are detected. The experimentation has been performed on a sequential circuit (the s344, from the ISCAS'89 benchmark) : 10 sequences of 5 test patterns have been generated and 92.5% of behavioral faults have been detected. Lastly, we have to point out that the required CPU-time for performing the pseudo-random process revealed to be negligible according to the one required for the deterministic generation process.

4: Conclusion

In this paper, we have presented a new approach for Behavioral Test Pattern Generation for digital circuits using concepts and tools from the field of Software Engineering. This approach is supported by the dynamic white-box strategy which consists in using several software coverage metrics. The experimental results have pointed out the efficiency of the pseudo-random approach in terms of time-consumption and the number of detected faults. At the present time, experiments have been conducted on ADA descriptions and necessitate the translation of VHDL descriptions in ADA. In future work, emphasis will be put on the development of an ad-hoc VHDL Analyzing Tool for analyzing control parts and computing the coverage metrics. Furthermore, the validation of the behavioral fault model is not addressed in this paper. However, it is a crucial open problem which must be solved in order to see people from industry becoming confident in Behavioral Testing.

References