A Uniform Test Generation Technique for Hardware/Software Systems

Gert Jervan, Petru Eles, Zebo Peng
Dept. of Computer and Information Science, Linköping University, Sweden
e-mail: {gerje, petel, zebpe}@ida.liu.se

Abstract

A novel hierarchical test generation technique for embedded systems containing hardware and software is proposed. Different from the traditional approaches, hardware and software parts of an embedded system are handled in a uniform way. We will in particular show how the proposed technique can be applied at the high levels of abstraction and how the software domain of the specification can also be successfully covered with experimental results.

1. Introduction

The increasing design complexity, combined with very tight time-to-market schedules, has generated very tough requirements for the productivity of embedded systems designers. Traditional design methods have been replaced by concurrent design of hardware and software and more work has been done at higher levels of abstraction. However the testing of the hardware and software parts of the system are still considered as totally different problems and solved with very different methods. There has been some work done in [1, 5, 11] to close the gap between these two different domains, but the area is not yet very well investigated.

In our approach, testability evaluation and test generation at the level of an implementation independent system specification are based on hierarchical test generation (HTG), a technique which has been successfully used until now for hardware test generation at the gate, logical and register-transfer (RT) levels. We apply HTG, using a decision diagram (DD) [10] based representation, and show that it can be used for both the hardware and software domains as well as for different levels of abstraction.

2. HTG for Hardware/Software Systems

Hierarchical test generation has been proposed in [2, 7, 8]. It has been used to generate tests for large sequential circuits. The main idea of the HTG technique is to use information from higher abstraction levels while generating tests for the lower levels. One of the main principles is to use a modular design style, which allows to divide a larger problem into several smaller subproblems and to solve them separately.

On the system level, the functionality of embedded system is described in the form of co-operating processes. Such a specification forms the basis for concurrent hardware/software design. To give the designer an opportunity to perform design for testability already in the early design stages, testability evaluation should be applied directly to the system specification. And a testability metric should be part of the cost function considered during system level synthesis, and in particular for hardware/software partitioning. Figure 1 shows how testability evaluation and test generation fit into such a system synthesis concept.

3. HTG for Specifications to be Implemented in Software

In our approach, decision diagrams are used for design modelling. In the general case, a DD is a directed, acyclic graph with nodes labelled by algebraic expressions. The main advantage of modelling with DDs lies in the fact that a uniform concept can be applied on different abstraction levels. An extended overview of DDs is presented in [10].

In a DD, non-terminal nodes represent logical conditions, terminal nodes represent operations, while branches hold the subset of condition values for which the successor node corresponding to the branch will be chosen.

One of our main objectives is to show how DDs can be used for test generation at the behavioural level. We concentrate on analysing how tests, generated from such a representation, can be used for testing the part of the system which finally will be implemented as software. At this level, for every internal variable and primary output of the design a data-flow DD will be generated. Terminal nodes of the data-flow DD represent arithmetic expressions. Further, an additional DD which describes the control-flow has to be generated. The control-flow DD describes the succession of statements and branch activation conditions. The DDs, extracted from a specification, will be used as a computational model in HTG for symbolic path activation.

3.1. Test Generation Algorithm

There are two types of tests which we consider in the current approach. One set targets nonterminal nodes of the control-flow DD (conditions for branch activation) and the second set aims at testing operators, depicted in terminal nodes of the data-flow DD.

The whole test generation task is performed in the following way. Tests are generated sequentially for each nonterminal node of the control-flow DD. Symbolic path activation is performed and functional constraints are extracted. Solving the constraints gives us the path activation conditions to reach a particular segment of the specification. In order to test the operations present in the terminal nodes of the data-flow DD, different approaches can be used. In this paper, we use mutation testing [4] for test generation for the operations at the terminal nodes. For path activation, a slightly modified version of the algorithm described in [7] is used.

3.2. Conformity Test

For the nonterminal nodes of the control-flow DD, conformity tests will be applied. The conformity tests target errors in branch activation. In order to test nonterminal node *IN1* (Figure 2), one of the output branches of this node should be activated.

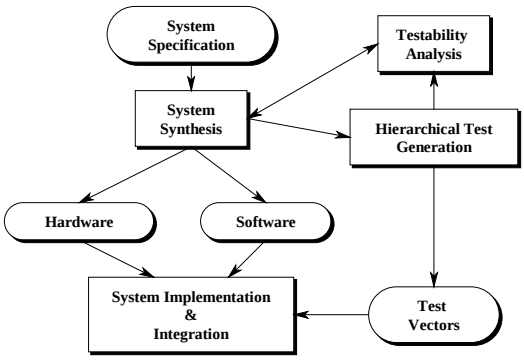

Figure 1. Test generation and testability analysis in a hardware/software co-design environment

This work has partially been supported by the Swedish National Board for Industrial and Technical Development (NUTEK).

Activation of the output branch means activation of a certain set of program statements. In our example, activation of the branch *IN1<0* will activate the branches in the data-flow DD where *q=1 (A:=X)*. For observability the values of the variables calculated in all the other branches of *IN1* have to be distinguished from the value of the variables calculated by the activated branch. In our example, node *IN1* is tested, in the case of *IN1<0*, if *X≠Y*. The path from the root node of the control-flow DD to the node *IN1* has to be activated to ensure the execution of this particular specification segment and the conditions, generated here, should be justified to the primary inputs of the module. This process will be repeated for each output branch of the node. In the general case there will be *n(n-1)* tests, for every node, where *n* is the number of output branches.

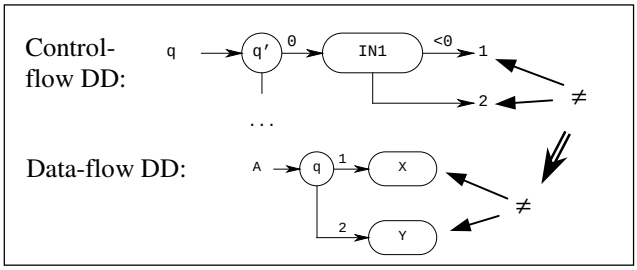

Figure 2. Conformity test

3.3. Testing Arithmetic Operators

As mentioned earlier, test vectors for the terminal nodes can be generated based on different approaches, and our HTG technique does not impose a specific one. Currently we use a mutation based fault model [4] for testing terminal nodes of the data-flow DD. We are using a library of operator mutations, which describes for each operator a set of corresponding mutants and conditions, which can distinguish between the mutant and the original operator. For example, if we have the expression:

```
x:=(a+b)-c;
```

To rule out the fault that the first "+" is changed to "-", b must not be 0 (because a+0=a-0). Additionally, to rule out the fault that instead of "+" there is "*", we have to assure that a+b≠a*b. For more details about operator mutants, we refer the reader to [6].

4. Experimental Results

Experiments were conducted in the environment consisting of our hierarchical test generator, the library of mutants for different arithmetic operators, and the Generic Coverage Tool (GCT) [9] which measures the quality of the generated test cases. Conversion between different representations (VHDL, C, Fortran and DD) is performed by the corresponding translation tools. In order to evaluate our results we compare them with those produced by the software test generation tool Mothra [3].

Experiments were carried out on three embedded software examples, which have different structures. Table 1 presents the experimental results of our approach in comparison with the results achieved by Mothra. The fault coverage presented in Table 1 is computed by GCT and reflects synthetically several different coverage criteria (statement coverage, branch coverage, loop coverage etc.). As observed, the mutation based testing tool Mothra generates a much larger set of test vectors, which, at the same time, produce a weaker coverage.

5. Conclusions

This paper describes a novel hierarchical framework for test generation in hardware/software systems. Hardware and software parts of an embedded system can be handled in a uniform way. The same DD representation can be used for describing systems at different abstraction levels, including the system level. Based on this representation reasoning about testability in the early design phases and test generation for both the hardware and the software domain is possible.

6. Acknowledgements

The authors would like to thank Prof. Raimund Ubar from Tallinn Technical University for his helpful discussions concerning the Decision Diagrams.

7. References

[1] G.Al-Hayek, C.Robach, "An Enhancement Process for High-Level Hardware Testing Using Software Methods," *IEEE European Test Workshop (ETW98),* Barcelona, Spain, 1998, pp. 215-219

[2] J. D. Calhoun, F. Brglez, "A Framework and Method for Hierarchical Test Generation," *IEEE Transactions on Computer-Aided Design*, Vol. 11, No. 1, January 1992

[3] R. DeMillo, D. Guindi, K. King, M. M. McCracken, J. Offutt. "An Extended Overview of the Mothra Software Testing Environment," *2nd Workshop on Software Testing, Verification, and Analysis*, Banff, Canada, July 1988, pp. 142-151

[4] R. A. DeMillo, R. J. Lipton, F. G. Sayward, "Hints on Test Data Selection: Help for the Practical Programmer," *IEEE Computer*, Vol.11, No.4, Apr. 1978

[5] O.P.Diaz, I.C.Teixeira, J.P.Teixeira, "Metrics for Quality Assessment of Testable Hw/Sw Systems Architectures," *IEEE European Test Workshop (ETW98),* Barcelona, Spain, 1998, pp. 205-209

[6] W. E. Howden, "Weak Mutation Testing and Completeness of Test Sets," *IEEE Transactions on Software Engineering,"* Vol. SE-8, No.4, July 1982

[7] G. Jervan, A. Markus, J. Raik, R. Ubar, "Hierarchical Test Generation with Multi-Level Decision Diagram Models," *7-th IEEE North Atlantic Test Workshop*, West Greenwich, RI, USA, 1998, pp. 26-33.

[8] J. Lee, J. H. Patel, "ARTEST: An Architectural Level Test Generator for Data Path Faults and Control Faults," *IEEE International Test Conference (ITC'91),* 1991, pp.729-738

[9] B. Marick, "Using Weak Mutation Coverage with GCT," Testing Foundations, 1992

[10] R. Ubar, "Test Synthesis with Alternative Graphs," *IEEE Design and Test of Computers*, Vol. 13, No. 1, pp. 48-57, Spring 1996

[11] H.P.E.Vranken, M.F.Witteman, R.C.Van Wuijtswinkel, "Design for testability in hardware software systems" *IEEE Design & Test of Computers* Vol. 13, No. 3, pp. 79-86, Fall 1996

Design module	Number of lines in the specification	Number of branches	Number of mutants	Mothra			Our approach	
				Number of generated test cases	Number of optimised test cases	Fault coverage	Number of generated test cases	Fault coverage
Square	38	12	813	707	5	77.65%	10	94.12%
Mult	20	6	478	449	3	84.00%	6	90.00%
FFT	31	4	1682	1639	4	83.91%	6	86.21%

Table 1 Experimental results