Model Based Diagnostic Test-Scheduling

I. Turcsányi, Gy. Csertán and A. Pataricza

Technical University of Budapest

Department of Measurement and Instrument Engineering

E-mail: turcsanyi, csertan, pataric@mmt.bme.hu

Introduction

Fault-tolerant systems could not work without some kind of fault-diagnostic mechanism. The base of any fault-diagnostics is a well planned test strategy. The first step of test strategy design is is the generation of the systems test set. The quality of the tests set can be evaluated by testability analysis and afterwards an optimized ordered subset of the tests should be created. In the operational system this set will be executed from time to time.

In [CPS95] a method is presented for a dataflow model based automatic test pattern generation (ATPG) and testability analysis. In this paper we focus on the optimization of the generated test set. Since the primary concern is improving the performance of the system, optimization is done according to the execution time of the tests, i.e. it tries to minimize the execution time of the test set. Using the presented approach both on-line/off-line testing and the systems built-in-self-test can be designed.

For the description of the test set a timed dataflow notation is used. It supports both coarse granular and fine granular parallelism, thus it can be used in any phase of the design, i.e. from a high level behavioral model through a register-transfer-level model down to a gate-level model if necessary. Nodes represent the computational units of the system, while channels (FIFO queues) describe the connections among the units. This way the fault-propagation paths from the primary/test inputs of the system to the primary/test outputs of the system are described. Time label is attached to the nodes,

characterizing the execution time of activities of the system components. Since at higher level models not all design details are known, this time labels are not necessarily the exact execution time of the activities, instead they are a pessimistic approximation for the execution time. In this case the resulting schedule will not be optimal. Instead the result is the upper bound on the execution time of the test set and the designer can use it as a qualitative parameter. At lower levels of abstraction, when the approximation of execution time of activities becomes more precise, the results of scheduling can be used for qualitative reasoning.

Scheduling

Scheduling consists of assigning each test to a time value to the starting time of the subtask [AG93]. It is known, that in general case the decision whether a task set is schedulable or not is NP-hard [Kop93]. The scheduling algorithms can be divided into two groups: transformational scheduling and iterative constructive scheduling.

An approach to *transformational scheduling*, known as exhaustive search, is to try all possible known transformation on the initial schedule. Optimum schedule is guaranteed, but the number of possibilities is so huge that the computation can not be carried out for practical cases.

Another approach to scheduling is *iterative constructive scheduling*, in which a new subtask is scheduled at every step, until all subtasks are scheduled. In this approach two decisions must be made at each step: which subtask should be scheduled next, and when should this subtask start? These decisions can be made by local or global criteria. Using global criteria the resulting schedule is usually better but the computation is more complex.

ASAP Scheduling Probably the simplest scheduling algorithm is ASAP. At each step the subtask is chosen that can be started at the earliest time. (As Soon As Possible.) The algorithm does not look ahead to avoid later scheduling conflicts. If more than one subtask can be started at the same time arbitrary selection is done.

list scheduling In list scheduling the subtasks on the critical path have priority over the other subtask. It means that subtasks on the critical path has to be scheduled

without delay, otherwise the whole schedule will be longer. This algorithm is more complex then ASAP, but due to the more global scheduling criteria it leads to a better scheduling.

Scheduling in the DFN model

In our model the test set of the system is a dataflow network, where the different tests of the test set are independent subnetworks. Using the scheduling terminology, tests correspond to tasks and dataflow nodes of the tests correspond to subtasks. In this case scheduling aims at the shortest execution of the network. In doing it, the following constraints have to be dealt with: The dataflow node executes only a single activity at a time, thus the capacity of each resource is one. The node reads tokens from the inputs only after the expiration of the previous firing rule. If a node tries to put a token into a full channel it will be treated as a collision and the scheduling of the current test gets into conflict.

The algorithm for ASAP can be outlined as follows. At each step the algorithm examines the available resources and determines the set of tests that can be started in this step. A test that could be started in this step, but will get into conflict with other tests in a subsequent step, is not element of the set. Then the first element of the set is scheduled, and the time of the scheduling is increased. If no tests can be started in this step, the time of the scheduling is increased.

In list scheduling the critical path priority of a test is defined as follows. If there are no logical dependencies in the test set the critical path priority of each test equals the execution time of the test. If logical dependencies are defined the critical path priority equals the execution time of the test plus the maximum of critical path priorities of the successor tests. In each step of the scheduling the test with the highest priority is scheduled.

Comparison of the different scheduling algorithms

Unfortunately exhaustive search can not be executed, thus results of other scheduling strategies can not be compared to the optimum schedule. Instead they have to be com-

pared to each other. Since results depend on the properties of the system and the test set respectively, comparison can only be done on a statistical basis.

For this reason a large variety of system models and test sets have to be evaluated. Therefore the system models and the tests sets have been generated randomly. The ranking of the different algorithms has been done using many inputs with the same statistical properties. This way we can compare not only the average properties of different scheduling strategies, but we get some insight into their worst and best case behavior. In the paper comparison will be given for the ASAP, list and modified ASAP algorithms.

References

[AG93] J. R. Armstrong and F. G. Gray. *Structured Logic Design with VHDL*. Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[CPS95] Gy. Csertán, A. Pataricza, and E. Selényi. Dependability Analysis in HW-SW co-design. In *Proceedings of the IEEE International Computer Performance and Dependability Symposium, Erlangen, Germany*, April 1995.

[Kop93] H. Kopetz. Six Difficult Problems in the Design of Responsive Systems. In H. Kopetz and Y. Kakuda, editors, *Responsive Computer Systems*, volume 7 of *Dependable Computing and Fault-Tolerant Systems*. Springer-Verlag Wien, 1993.