Allocation for Testability in High-Level Synthesis Process

Peter Bukovjan* Meryem Marzouki Walid Maroufi

LIP6 Laboratory
Couloir 55-65, 4 Place Jussieu, 75252 Paris Cedex, France
Tel. (+33)1 44 27 74 74 - Fax. (+33)1 44 27 72 80
E-mail: Peter.Bukovjan@lip6.fr, Meryem.Marzouki@lip6.fr, Walid.Maroufi@lip6.fr

Abstract

This paper focuses on the allocation for testability process as part of a High-Level Synthesis for Testability system. The allocation for testability is mainly defined as the search for best cost/quality trade-offs between the designer requirements and testability means that can be proposed by the system, considering the available functional units in a library and the possibility to generate additional testability structures.

1 Introduction

When the High-Level Synthesis for Testability (HL-SFT) process is viewed as the automatic generation of testable architectures [1], [2], it involves on one hand the choice of the suitable test structures to be allocated according to a target test methodology and to a set of constraints dealing with cost and performances; on the other hand, the HL-SFT process involves a global test scheduling process. A global test scheduling process has been presented in [2], and this paper concentrates on a methodology which transforms the allocation steps of the high-level synthesis process into allocation for testability steps. The overall objective of this methodology is to make the best possible cost/quality trade-off in terms of testability during allocation.

2 Allocation For Testability Process

For the purpose of prototyping, the allocation for testability methodology has been developed on the basis of a university behavioral synthesis system named AMICAL [3], but all the algorithms presented later in the following sections are transferable also to others high-level synthesis systems.

AMICAL starts its allocation step after a scheduling step has been completed. Then it makes a virtual FUs allocation, on the basis of the operations described in the state table resulting from scheduling. This virtual FUs allocation selects the relevant FUs from a library of FUs described by their abstract view, which is in fact a collection of *design criteria* of the corresponding FU. After virtual FUs allocation, micro-scheduling, register placement, and connection allocation steps are processed, producing an abstract architecture which will be later transformed into a true RTL one and finaly synthesized by some of the comercial synthesis tool.

This AMICAL allocation process does not take testability into account at all. The proposed *allocation for testability* process starts right after AMICAL virtual FUs allocation has been made, and then proceeds to the following steps (see also Figure 1):

*Ph.D. student at the Department of Microelectronics - TU of Brno (Czech Republic) and at the Laboratory of Informatics - University Paris 6

- **Virtual FUs extraction from the previous allocation process:** Virtual FUs are black-boxes described by the *design criteria* of the corresponding real FU. These design criteria include functionality, area, bit-width, etc. They are extracted from the result of the allocation process of the behavioral synthesis system. This choice has been made in order to keep the allocation for testability process as independent as possible from the allocation process, thus reuse of allocation for testability software can be easily made in the future in conjunction with other behavioral synthesis software, provided that relevant interfaces are developed.

- **Corresponding testable FUs versions extraction from the library:** Whenever available, actual testable FUs versions corresponding to the previous virtual non-testable FUs are extracted from the library. This means that their *testability criteria* are identified, and stored in a *characteristic vector*. According to the *user requirements*, these testable FUs versions can be either of a BIST or a scan type. The subsequent trade-off among different testable versions or even the non-testable one of a given FU (together with the generation of additional testability structures) will be made against these characteristic vectors, the user requirements, some *weighing factors* and testability method constraints.

- **FUs allocation for testability itself:** After these pre-processing phases, the allocation for testability can start. It mainly encompasses the following five phases:

  - **Initialization phase:** The user is first asked to answer the question of desired *global testability method*. In the current version of the allocation for testability software, available global methods are: *non testable* (in this case, the allocation for testability software stops processing), *self-test, i.e. BIST*, and *external testing, i.e. scan*. Future versions of the software will allow to mix scan and BIST in the same device. After that, the user is asked to specify his testability requirements for all FUs to be allocated. As is detailed in [4], these requirements are described through a *requirement vector* R and a *weighing vector* W.

  - **Trade-off phase composed of the local and the global trade-off:** This step is the most important one of the whole allocation for testability process. At a given phase of the process the FUs should be chosen from the library. For each of these FUs, the software tries to choose its best version in the library. This trade-off is made in an interaction with the user, who gives his requirements and preferences. In case the requirements cannot be satisfied in the current state of the library then the user is asked to modify his requirements until a solution is found, which can either be the allocation of a testable version of a FU, or keeping the non-testable version and generating an additional testability structure means (e.g. a Test Pattern Generator - TPG - and a Test Result Checker - TRC - for BIST architecture, whenever the combination of the non-testable version plus the TPG and the TRC answers the user requirements), or the system tells him that no solution exists (this means that the AMICAL allocation choice will be kept for the given virtual FU, which is a non-testable solution).

  - **Actual allocation for testability:** This step has the purpose of replacing the allocated virtual FUs by the actual ones.

  - **Generation of additional testability structures when needed:** This step aims at making the actual generation of additional testability structures (TPG and TRC to be shared by all non-testable FUs, necessary scan chains, needed multiplexors, etc.).

  - **Micro-scheduling:** Just like AMICAL includes in its allocation process a micro-scheduling phase, the allocation for testability process needs own micro-scheduling step after an allocation for testability of virtual FUs has been completed.

Figure 1 summarizes the overall allocation for testability process, and its interaction with the rest of the system.

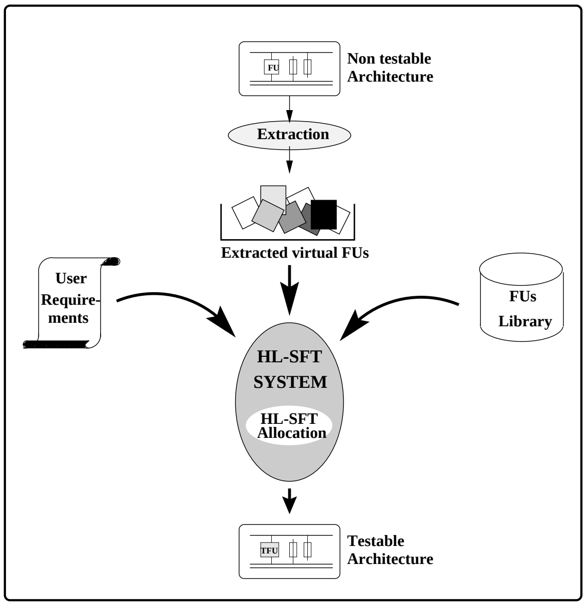

Figure 1: Schematic of the allocation for testability process

3 Overview of Trade-off Process

As stated in Section 2, the trade-off process is the core of the allocation for testability process. It starts after the extraction phases, and the initialization phase of the allocation for testability. The trade-off has the purpose of making the best choice between various versions (one non-testable and the others testable) of a given FU to be allocated. It uses the *characteristic vector* V, the *global attributes vector* GA, the *requirement vector* R and the *weighing vector* W of all FUs making the whole testable datapath architecture. In addition, the trade-off algorithm takes into account the mandatory requirements imposed by the chosen testability method. These mandatory requirements are called *testability method constraints* (TMCs).

The trade-off process searches first of all to satisfy the user's requirements. If no such FU version is found in the library then a dialogue starts between the trade-off module and the user, who is asked to modify step by step his requirements, being guided by the system. This is based on functions that are inspired by the work done in [5], since the same kind of problems have to be solved.

The trade-off algorithm can be decomposed into two main steps:

- Local trade-off (trade-off at the level of FUs - vector V compared with vector R)
- Global trade-off (trade-off at the level of circuit - vector GA compared with vector R)

4 Conclusion

This paper has focused on the allocation for testability methodology, which, added to the global test scheduling and control methodology presented in [2], forms a global HL-SFT environment [6]. Contrarily to other approaches reported in the literature ([7], [8], [9] and [10]), our HL-SFT environment does not start from the behavioral description of the device, in order to modify this description to include testability features. In our case, we proceed in a very pragmatic way, after a first pre-allocation of virtual FU has been made by a behavioral synthesis system, and *we modify the allocation algorithm to take into account testability features.* This way, we keep closer to the architecture and the structural view of the datapath, being so able to overcome the contradictory requirement of HL-SFT, i.e. to address physical faults starting at a high-level of description.

Future work is directed towards the extraction of as much as possible design and testability attributes from real FUs. More experiments with real cases of circuit datapaths are required, and also we would like to allow mixing different testability methods in the same datapath.

5 Acknowledgments

This research was performend with the financial support of the French government and with the financial support from the Grant Agency of Czech Republic - contract No. 102/98/1463.

References

[1] M. Marzouki, V.C. Alves, and A.R. Antunes. High-level synthesis for testability. In *38th IEEE Midwest Symposium on Circuits and Systems, Special Section on Testability Issues for Digital Circuits and Systems*, pages 718–721, Rio de Janeiro, Brazil, August 1995.

[2] M. Marzouki, V.C. Alves, A.R. Antunes, and W. Maroufi. Global test scheduling and control in a hl-sft environment. In *1st IEEE High-Level Design Validation and Test Workshop*, pages 14–17, November 1997.

[3] A. Jerraya, I. Park, and K. O'Brien. Amical: An iterative high-level synthesis environment. In *European Design and Test Conference*, Paris, France, 1993.

[4] P. Bukovjan, M. Marzouki, and W. Maroufi. Allocation for testability by searching cost/quality trade-off in high-level synthesis for testability. Technical report, LIP6 Laboratory, Paris, April 1998.

[5] X.A. Zhu and M.A. Breuer. A knowledge-based system for selecting test methodologies. In *IEEE Design and Test of Computers*, October 1988.

[6] T.E. Marchok, A. El-Maleh, J. Rajski, and W. Maly. Testability implication of performance-driven logic synthesis. *IEEE Design and Test of Computers*, 12(2):32–38, Summer 1995.

[7] L.J. Avra and E.J. McCluskey. High-level synthesis of testable design: An overview of university systems. In *IEEE International Test Conference - Test Synthesis Seminar*, Washington D.C., October 1994.

[8] S.K. Gupta. Synthesis of testable combinational circuits: An overview of university activities. In *IEEE International Test Conference - Test Synthesis Seminar*, Washington D.C., 1994.

[9] I. Pomeranz and S.M. Reddy. Synthesis of testable sequential circuits: An overview of university activities. In *IEEE International Test Conference - Test Synthesis Seminar*, Washington D.C. - USA, 1994.

[10] M.T.-C. Lee. *High-Level Test Synthesis of Digital VLSI Circuits.* Artech House Boston - London, 1997. ISBN 0-89006-907-7.