Design for Testability Reuse in Synthesis for Testability*

Peter Bukovjan, Meryem Marzouki and Walid Maroufi
LIP6 Laboratory
Couloir 55-65, 4 Place Jussieu
75252 Paris Cedex 05, France
Peter.Bukovjan@lip6.fr, Meryem.Marzouki@lip6.fr, Walid.Maroufi@lip6.fr

Abstract

This paper presents our Design for Testability reuse approach implemented in the allocation for testability system IDAT. In the context of High-Level Synthesis for Testability, the allocation for testability process mainly consists in searching for the best cost/quality trade-off between the designer requirements and testability means which can be proposed by the system, considering the available components in the library and the possibility to generate additional testability structures. The cost/quality trade-off is also based on the result of the testability analysis process.

1 Introduction

Most of the existing High-Level Synthesis (HLS) tools only focus on design criteria optimization, without considering testability issues at the high-level. The resulting circuit architecture, after lower level synthesis, is either very hard to test or testable with very bad performances and high area overhead.

To avoid these drawbacks, some authors have considered High-Level Synthesis for Testability (HL-SFT) solutions, where testability is taken into account before RT-Level generation. Comprehensive overviews of such solutions can be found in [2], [7], [11], [12], and [8].

These approaches suffer from two main disadvantages, the first one being that they consider the behavioral description of the circuit, which means that the resulting architecture will not allow an accurate mapping of the testability resources, and the second one being that they do not make profit of already existing testable components or testability structures. In addition, the solutions proposed in the literature are loosely interactive.

The HL-SFT approach that we have defined and implemented in our *Interactive Design for Test reuse in Allocation for Testability* (IDAT) system has the purpose of overcoming these drawbacks. The whole process is viewed as the automatic generation of testable architectures, starting from a *Rich Virtual Architecture* (RVA), and making profit of the *Design for Test (DFT) Reuse* concept [9]. The whole process is driven by cost/quality trade-off between user requirements related to a target test methodology and existing testable components in the library imposing different constraints. The methodology presented in this paper is complemented with a global test scheduling process [9].

This article is organized as follows: section 2 presents the overall IDAT system for allocation for testability, then section 3 overviews the testability analysis module, and section 4 explains the cost/quality trade-off process. Section 5 describes the implementation of the IDAT system and provides somes experimental results.

2 Allocation for testability

The IDAT (*Interactive DFT reuse in Allocation for Testability*) system starts from the RVA *Rich Virtual Architecture* description of the circuit to be synthesized. This RVA description is a draft description of the circuit architecture, where the operators have already been allocated, and the variables have been assigned to registers. Contrarily to the RTL architecture, fine synchronization information are not yet taken into account. The RVA description is then obtained after the allocation and binding steps of the HLS process. This draft architecture is enriched with design attributes, like area, bit-width, maximal delay, etc. quantified using generic units.

The RVA is modified by the IDAT system, taking into account the designer requirements in terms of testability, and availability of testable components and testability structures. The best test cost/quality ratio is searched.

*This work has been partly supported by MEDEA-SMT Project and by a grant from the Grant Agency of Czech Republic, contract No. 102/98/1463. The authors are grateful to Laurent Ducerf-Bourbon (LIP6) and to Vladimír Drábek and Zdeněk Kotásek (DCSE, Technical University of Brno) for their help.

The allocation for testability process consists in four steps: (1) extraction of the virtual architecture from the result of a HLS system, (2) extraction of testable versions of allocated components from a library, (3) first testability analysis of the virtual architecture and (4) test cost/quality trade-off itself, according to the testability methodology chosen by the designer: scan-based or BIST-based methodology. This process can itself call the testability analysis module, if needed.

3 Testability analysis

Our testability analysis process [4] is based on the calculation of combinational and sequential controllability and observability factors. Although some concepts are common to other approaches presented in literature (especially Gu's one [6]), our testability analysis process differs from them among others by the fact that we suppose the creation of a circuit test controller in both scan and BIST modes [9].

The *combinational controllability* (CC) and *combinational observability* (CO) respectively measures the difficulty to set a desired value to an internal node from the circuit primary inputs and to observe this internal node value from the circuit primary outputs. The combinational factors range from 0 to 1. The sequential factors - *sequential controllability* (SC) and *sequential observability* (SO) - range from 0 to infinity, and respectively measures the sequential depth of the node and the estimated number of clock cycles in test mode to reach the corresponding combinational value. The *controllability* C and *observability* O of a given node is calculated from corresponding combinational and sequential factors. After that, the *testability* of the node T is calculated so as to undertake subsequent actions on hard-to-test nodes. To express the whole datapath testability the *global testability indicator* (T_{global}) was defined.

A *propagation algorithm* has been developed in order to calculate the controllability and observability of the virtual architecture nodes. This algorithm is based on progressive calculation of the controllability and observability for each circuit node. Details on the testability analysis algorithm can be found in [4].

4 Trade-off process

The trade-off process is the core of the allocation for testability process. Its criteria include the user requirements and their corresponding weights. The trade-off process has the purpose of making the best choice between various versions of a given component to be allocated, and to suggest inclusion of additional testability structures, in order to satisfy the user requirements. In addition, the trade-off algorithm takes into account the mandatory requirements imposed by the chosen testability method (BIST or scan).

The trade-off process searches first of all to satisfy the user requirements. If this is impossible with the existing library then a dialogue starts between our system and the user, who is asked to modify step by step his requirements, being guided by the system.

Testability problems are global in nature. A 100% testable component may show testability problems when it is embedded into a whole datapath. To avoid this situation, the trade-off algorithm is decomposed into two main modules: the *local trade-off* module, which acts at the component level, and the *global trade-off*, which operates at the datapath level.

The trade-off process is part of the allocation for testability process. This means that a library of components to allocate is available, each component being described by its testability attributes, in addition to its design attributes.

Just like a cost/quality trade-off between the *design attributes* of related components (area, maximal delay and power consumption) is made during the allocation step of a classical HLS process, the new cost/quality trade-off has to be made between the *testability attributes* of related components (area overhead, extra delay, extra power consumption, testability, test parallelism, etc.) during the allocation for testability process, considering also other parameters which are the *user requirements* and some *weighing factors* to help the system balancing the various requirements. The design and testability attributes of each component are summarized in a *characteristic vector*. Actual circuit components configurations are stored in the *allocation vector*. The user requirements saved in the *user requirements vector* are classified into three groups according to their relationship in a component or circuit testability description, and this also corresponds to the user expertise in the field of HL-SFT and DFT (beginner, advanced, expert).

Since the trade-off process can be made at the local level (among different component versions) and at the global level (circuit architecture), the attributes can also be distinguished as *local attributes* represented by the characteristic vector and *global attributes*, which are calculated from local attributes, using equations given in [3], and then saved to the *global attributes vector*. Most of the attributes exist at both local and global levels. In the sequel, only the global trade-off process will be detailed. The local trade-off has the purpose of ordering different testable versions of a given components, taking only into account local attributes of this component (e.g. area overhead, extra I/O, etc.).

The goal of the global trade-off is depending upon the choice of the global testability methodology made by the designer: scan or BIST methodoly.

In case of scan methodology, the goal of the global trade-off is to find the optimal set of library components to be replaced by their scan version, when needed, after testability analysis. To reach this goal, the global trade-off algo-

rithm first evaluates all the global attributes for the *allocation vector*, which contains all the components allocated in the datapath. Then, the algorithm searches in this vector all the failing components with respect to the user requirement vector, using the testability analysis process, until the best solution is found, or until ending with no solution found.

In case of BIST methodology, a Test Pattern Generator (TPG) and a Test Result Checker (TRC) are added to the datapath. The global trade-off algorithm attempts to connect all hard-to-test points (corresponding to non BISTed components) to the TPG and the TRC. Trade-off between test parallelism and area overhead is made when some component exist also in BIST equivalent. Such BIST component can increase test parallelism degree of the circuit, but the reduction of TPG and TRC complexity can be negligible compared to resulting area overhead increase.

The trade-off algorithm uses as computing functions: (1) a *distance function* to estimate the difference between the value of a given attributes in the allocation vector and its corresponding value in the user requirement vector, (2) a *penalty-credit function* (PCF) which normalizes the resulting value of the distance function, and takes into account the weight of the requirements, and (3) a *score function* which attributes a score to a component (local trade-off) or to the datapath (global trade-off), in order to classify the different proposed alternatives. These functions are very similar to those used in [13] and [14], but they operate at a different level of abstraction.

5 Implementation and Results

IDAT has been implemented using C++. Data-path components are represented as objects with related attributes and related methods to compute these attributes.

In the frame of the ALLIANCE environment [1] the input to the IDAT system is a VHDL description of the datapath virtual architecture, which is automatically translated into the Rich Virtual Architecture (RVA) format. In the frame of other environments, it is necessary to create the corresponding driver to RVA. It is also possible to directly use an RVA description created manually. An RVA description contains, in addition to the virtual architecture description itself, information on the type of the signals (DATA or CONTROL), information on the component function, and information on its design attributes. Other inputs to IDAT are the user requirements with regards to testability.

Output of IDAT is the modified RVA description of the datapath (in VHDL), together with the list of all architectural modifications made during the allocation for testability process. This output is provided back to the HLS system, which then completes its tasks.

The Diffeq architecture [10] has been used as benchmark. Diffeq is originally described at the behavioral level, while the input of our process is a virtual architecture (section 2). We have then used as virtual architecture a simplified version of its RTL one, which was generated by the HLS system HAL presented in [10].

Although this circuit is often used for benchmarking, it has been difficult to compare our results with those reported in [2], [7], [11], [12] and [8] in terms of fault coverage and test generation time. One reason is that we don't have a sequential ATPG available at our site, but the main, intrinsic, reason is that the results of our system are strongly related on one hand to the provided virtual architecture, which depends upon the HLS system used, and on the other hand to existing library of testable components.

The Diffeq scheme [10] with necessary labels to components and signals is depicted in figure 1. Attributes are extracted from original components, together with controllability and observability transfer factors (CTF and OTF) used in testability analysis [4] and calculated using equations from [5]. The user (beginner in this case) requirements are : SCAN methodology, with a maximum area overhead of 3% (weight 1) and a minimum testability of 60% (weight 2). For clarity purposes, weights of the requirements power consumption and test parallelism are set to 0.

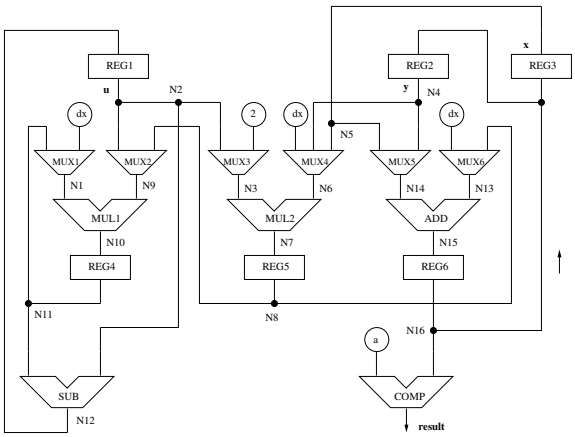

Figure 1. Diffeq datapath generated by HAL

The allocation for testability algorithm extracts from a library of testable components all testable equivalents of circuit components (registers and functional units).

Testability analysis results are summarized in table 1. First column indicates the nodes in the benchmark circuit. Then, the controllability, observability, and testability results for each node are given when the three allocations vectors av_1, av_2 and av_3 are considered. Vector av_1 corresponds to the original (non testable) components. The global attribute $testability$ does not correspond to user requirement and it is necessary to improve it.

The testability analysis process identifies as worst testable nodes N2, N3, and N12 (see figure 1). IDAT pro-

	av_1			av_2			av_3		
	C	O	T	C	O	T	C	O	T
a	1	0	0.50	1	0	0.50	1	0.38	0.69
dx	1	0	0.50	1	0.44	0.72	1	0.53	0.76
res	0	1	0.50	0	1	0.50	0.47	1	0.73
cst	0	0	0	0	0.37	0.18	0	0.41	0.20
N1	1	0	0.50	1	0.44	0.72	1	0.41	0.71
N2	0	0	0	0.75	0.47	0.61	0.66	0.44	0.55
N3	0	0	0	0.75	0.37	0.56	0.66	0.41	0.53
N4	0	0.52	0.26	0	0.37	0.18	0.66	0.60	0.63
N5	0	0.52	0.26	0	0.37	0.18	0.50	0.60	0.55
N6	1	0	0.50	1	0.37	0.68	0.66	0.43	0.54
N7	0.88	0	0.44	0.63	0.44	0.54	0.55	0.51	0.53
N8	0.63		0.31	0.55	0.47	0.51	0.50	0.53	0.52
N9	0.63	0	0.31	0.75	0.47	0.61	0.66	0.44	0.55
N10	0.54	0	0.27	0.63	0.55	0.59	0.55	0.53	0.54
N11	0.46	0	0.23	0.55	0.60	0.57	0.50	0.55	0.53
N12	0	0	0	0.48	0.75	0.61	0.43	0.75	0.59
N13	1	0	0.50	1	0	0.50	1	0.53	0.76
N14	0	0.52	0.26	0	0.52	0.26	0.66	0.60	0.63
N15	0	0.57	0.28	0	0.57	0.28	0.57	0.66	0.62
N16	0	0.82	0.41	0	0.82	0.41	0.53	0.75	0.64
area	+0%			+1.7%			+2.51%		
test	36.5%			52.7%			62.9%		

Table 1. Testability analysis results

poses to improve the testability of the first of these nodes (N2) by replacing register REG1 which is connected to N2 by its testable (scan) equivalent. The new allocation vector is then av_2. Improvements due to this vector are shown in the corresponding columns.

However, nodes N4, N5, N14, N15, N16, cst and res remain uncontrollable, thus IDAT starts a second iteration to improve their testability (except node cst which is a constant, so we can only improve its observability), IDAT suggests to replace register REG2, which is connected to node N4 (node with the smallest testability) by the testable one. The resulting allocation vector is av_3, improving the testability results as shown in the corresponding colums: the testability of all nodes has been increased, or have remained the same, and there is no remaining uncontrollable node.

All the user requirements have been satisfied, as shown in last two lines of table 1, and the allocation for testability process ends with allocation vector av_3, which is then translated into modifications of the virtual architecture.

6 Conclusion

This paper has described the IDAT system, designed for DFT reuse during allocation for testability of a datapath. The DFT reuse is driven by test cost/quality trade-off between user requirements related to a target test methodology and existing testable components in the library, imposing different circuit characteristics. Provided results show the capabilities of the system. IDAT is working on a virtual architecture of the datapath, enriched with design attributes of each components. Next step is to integrate the IDAT system in a HLS system designed at our laboratory, named UGH (User-Guided High-Level Synthesis System). UGH is being developed in the framework of the European ESPRIT project COSY/OMI.

References

[1] Alliance vlsi cad system. http://www-asim.lip6.fr/alliance/, 1999.

[2] L. Avra and E. 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.

[3] 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, February 1999.

[4] P. Bukovjan, M. Marzouki, and W. Maroufi. Testability analysis in high-level synthesis for testability. In *Proceedings from 10th European Workshop on Dependable Computing*, pages 71–75, Wien, Austria, May 1999.

[5] X. Gu. *RT Level Testability Improvement by Testability Analysis and Transformations*. PhD thesis, Linkoping University, Sweden, 1996. LiU-Tek-Ph.D 414, ISBN 91-7871-654-3.

[6] X. Gu, K. Kuchcinski, and Z. Peng. Testability analysis and improvement from vhdl behavioral specification. In *Proceesings of the European Design Automation Conferenece with EURO-VHDL*, pages 644–649, 1994.

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

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

[9] M. Marzouki, V. Alves, A. 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 1996.

[10] P. Paulin, J. Knight, and E. Girczyc. Hal: A multi-paradigm approach to automatic data path synthesis. In *Proceedings of the Design Automation Conference*, pages 263–270, June 1986.

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

[12] K. Wagner and S. Dey. High-level synthesis for testability: a survey and perspectives. In *Proceedings of the Design Automation Conference*, pages 131–136, 1996.

[13] X. Zhu. *A Knowledge-Based System for Testable Design Methodology Selection*. PhD thesis, University of Southern California, Los Angeles, August 1986. CRI-86-23.

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