Cost/Quality Trade-off in High-Level Synthesis for Testability *

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 trade-off algorithm, which is a core part of the allocation for testability process 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. The paper first presents the defined methodology and the considered criteria, and then gives example results.

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 the methodology which transforms the allocation steps of the high-level synthesis process into allocation for testability steps was introduced in [3]. This paper concentrates on the trade-off algorithm, which is a core part of the whole allocation for testability proces. The overall objective of this algorithm is to make the best possible cost/quality trade-off in terms of testability during allocation.

This article is organized as follows. The brief overview of already existing HL-SFT systems is done in Section 2. Section 3 details the description of the various criteria used for making the trade-off and presents the overview of the trade-off process itself, which is the core module of the allocation for testability software. Section 4 demonstrates this algorithm on an example and Section 5 gives some conclusions and perspectives.

2 Related Works

One of the first overviews of high-level test synthesis systems can be found in [4], [5] or [6]. Three years later more detailed overview is done in [7]. The Design for Test (DFT) techniques are summarized in [8]. The interested reader can refer to this links. From the number of existing HL-SFT systems we can distinguish two groups of HL-SFT systems according to the used test strategy (BIST or scan, alternatively test point insertion strategy), the third group mentioned below is the group of DFT systems:

- **BIST synthesis systems:** When targeting BIST methodology, the HL-SFT system must generate the circuit integrating a Test Pattern Generator (TPG) and Test Result Checker (TRC). A commonly used BIST method is based on BILBO registers. Such systems are SYNTEST [9], SYNCBIST [10] and RALLOC [11].

- **Synthesis for scan path and test point insertion:** Scan path is an effective test strategy which provides direct controllability and observability of memory elements by converting common registers

*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.

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

into scan ones and connecting them into a scan chain as a long shift register. Many of the existed systems, like for example PHITS-PS [12], the Gu et al. system [13] and Flottes et al. [14] both using testability analysis based on controlability and observability, use the partial scan technique to reduce the area and delay overhead. The Gu et al. system use as well test point insertion technique similar to GENESIS [15] etc.

- **Test synthesis at RTL:** At RTL although the circuit architecture is determited, the structural information available can still help test synthesis algorithms to economicaly add DFT circuitry even thought they generaly give bigger area and delay overhead than HL-SFT systems. From the great number of these system we can refer ADEPT [16] performing intelligent partial scan flip-flop selection based on RTL information and AMBIANT system [17] performing testability analysis on a RTL description to suggest modification targeted at improving testability of the design.

The general drawback of most above systems is that they generally consider only a circuit datapath or on the contrary only the circuit controller. As stated in [1] our objective is the global testable architectures (control and data part together). So from this point of view our HL-SFT environment is hardly to class to any of these groups. Moreover, our system does not start from the behavioral description of the device, in order to modify this description to include testability features like most above mentioned HL-SFT systems. We also do not modify real RTL architecture like DFT systems. In our case, we modify the virtual architecture (just before the RTL one) of a circuit to take into account testability features.

3 Trade-off Process

As stated in Section 1 the overview of our allocation for testability method was described in [3]. Here we will concentrate on the trade-off algorithm as a core algorithm of the whole allocation for testability process. 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.

3.1 Trade-off Criteria

The trade-off process is part of the allocation for testability process. This means that a library of FUs to allocate is available, each FU being described by its testability attributes, in addition to its design attributes. Just like a trade-off between the design criteria is made during the allocation step of High-Level Synthesis process, the test cost/quality trade-off has to be made between the testability criteria during the allocation for testability process, considering also other parameters which are: the design criteria themselves, the user requirements and some weighing factors to help the system balancing the various criteria. This section details all the criteria that have to be considered, which are the design criteria, the testability criteria, the user requirements, and the weighing factors.

A trade-off process is obviously based on requirements and preferences, in addition to the FU intrinsic criteria. These requirements are provided by the user to the system. Trade-off mechanisms have always been used, and their principles are well known. They have been particularly used for the same purposes as our own ones, i.e. selecting test methodologies, but at a lower level of abstraction, namely at the logic structure level. Such kind of work has been developed in the late eighties by the University of Southern Califonia (USC) team, under the direction of Professor Melvin Breuer. This work was developed for PLAs, RAMs, ROMs, and combinational logic. Ten years after, the state-of-the-art has allowed to gain one level of abstraction and we are in our own work solving the same kind of problems at the whole circuit architecture level. In our case, the approach developed by Zhu et al. at the USC [18], [19] has been particularly useful, and we have adapted it to our context. The main vectors used by our trade-off algorithm are:

- **Characteristic vector V:** (or also local attributes vector) of a FU version is made up of the selected design attributes of given FU, plus testability attributes corresponding to chosen testability method

$V_i = (att_1, att_2, ..., att_c)$, where index i indicates the i-th FU version in the set of all versions of the given FU and c is the number of all criteria for chosen testability method. The design criteria together with the testability criteria are summarized in [20].

- **User requirements vector R:** contains the values requested by the user for some criteria. Depending on the corresponding criteria, the value set by the user at the beginning of the trade-off process can be an exact value, or a minimum, or a maximum allowed. In our case, we have identified the following criteria for the requirement vector: $R = \{method$, max_del, $bist\_style$, $scan\_style$, $part\_scan$, $reconfig\_scan$, $scan\_chains$, $controlab$, $observab$, $area\_over$, $fault\_cov$, clk_cycles, $extra\_I/O$, $tristate\_bus$, $intern\_loops$, $reset$, $test\_lenght$, $test\_time\}$. Of course, using of each requirement depends on the chosen global testability method.

- **Global attributes vector GA:** characterize the whole circuit. Global attributes correspond to the user requirements in a requirement vector R and they are calculated during the global trade-off: $GA = \{method$, max_del, $bist\_style$, $scan\_style$, $part\_scan$, $reconfig\_scan$, $scan\_chains$, $controlab$, $observab$, $area\_over$, $fault\_cov$, clk_cycles, $extra\_I/O$, $tristate\_bus$, $intern\_loops$, $reset$, $test\_lenght$, $test\_time\}$. The very important global attributes *controlab* and *observab* are calculated using the modified *RTL testability analysis* from [13].

- **Weighing vector W:** summarizes the weighing factors: The weighing factors are also provided by the user to give his preferences, ranking then his requirements by order of importance.

3.2 Local Trade-off

The local trade-off is made at the level of particular FUs. The input to the local trade-off is composed by all FUs to allocate with their extracted versions [1] from a FU library. The output of this process consists of the matrix M composed of selected and ordered FU versions (according to the local score function). The main phases during the local trade-off are:

- **Initialization of the matrix M:** The matrix M of n columns and m lines is initialized, where m is the number of all FUs to allocate and n is the maximal number of all FUs versions extracted from the FUs library. The ordered versions for each FU will be stored in the columns of this matrix and it will compose the output of the local trade-off process.

- **Trade-off for one FU:** is included in the local trade-off loop and contains the two main phases:
  - **Reduction:** At this step, all existing versions of FUs to be allocated have been collected, and they are then reduced to a smaller set of FU versions compatible with the TMCs of the chosen testability method. The reduction of the set of all FU versions is processed in view of compatibility with the TMCs imposed by the desired test method for the whole datapath. The main TMCs are:
    - **For scan methodology:**
      The compatibility among different scan styles, in order to decide the generation of one or more scan chains (differents clock signals, scan-enable input, etc.)
    - **For BIST methodology:**
      The TPG and TRC width, since a BISTed datapath can be reached either by allocating only BISTed FUs, or by allocating non-testable FUs and generating additional testability structures, with the constraint that only one TPG and TRC should be generated, and shared by all the non testable FUs of the datapath. Obviously, the datapath architecture can be made up of some BISTed FUs, and some non-testable FUs sharing a common TPG and TRC. In order to be shared, these additional test structures should satisfy a constraint related to the bit-width of the non-testable FUs. In the BIST case, the style compatibility is not taken into account, since any BISTed FU can be tested independently from the others, and these tests can be run in parallel.

  In the current stage of the work, only BIST or only scan datapath is allowed. Future versions of the trade-off process will allow to mix the testability methods, but, as for now, the BIST reduction means that all existing FU versions will be reduced to BIST FU versions, and the same applies to the scan reduction.

[1] In the following text the short term *FU version* will replace the long term *characteristic vector of a FU version*. The symbol remains V and also the meaning of this term (definition in Paragraph 3.1) remains the same.

– **Ordering:** The ordering is made in this step only for testable FU versions, non-testable versions are included and ordered later during the global trade-off. For ordering of all testable FUs versions in the selection vector the *local score function* is used. The best score is given to the testable FU version which best meet the user requirements, with respect to the local test attributes. These versions are ordered in the descending order. The selection vector sv is later put as the corresponding column to the matrix M.

During a pass of the local trade-off loop, the trade-off for the related FU is provided according to the chosen global testability method. This trade-off generates a *selection vector* $sv = (V_1, V_2, ..., V_z)$, containing ordered FU versions of this FU, where z is the number of selected versions for corresponding FU. Then following test of the selection vector is made: If the vector sv is not empty ($z > 0$), it is put like the culomn to the matrix M. The local trade-off loop finishes after the matrix M contains ordered versions of all FUs to allocate. If the vector sv is empty ($z = 0$), no solution for selected global testability method exists. The whole trade-off process ends with no solution.

3.3 Global Trade-off

The global trade-off is made at the level of the whole circuit. The purpose of this trade-off is to find the best possible collection of FUs versions to satisfy user's requirements. For each FU the appropriate version is selected and put in this collection named *allocation vector* (av).

Like the input to the global trade-off the matrix M of ordered FU versions is considered. The result of the global trade-off is *the best allocation vector* composed of selected FU versions. The simplific flowchart of this algorithm is illustrated in Figure 1. Three loop can be distinguished in this flowchart:

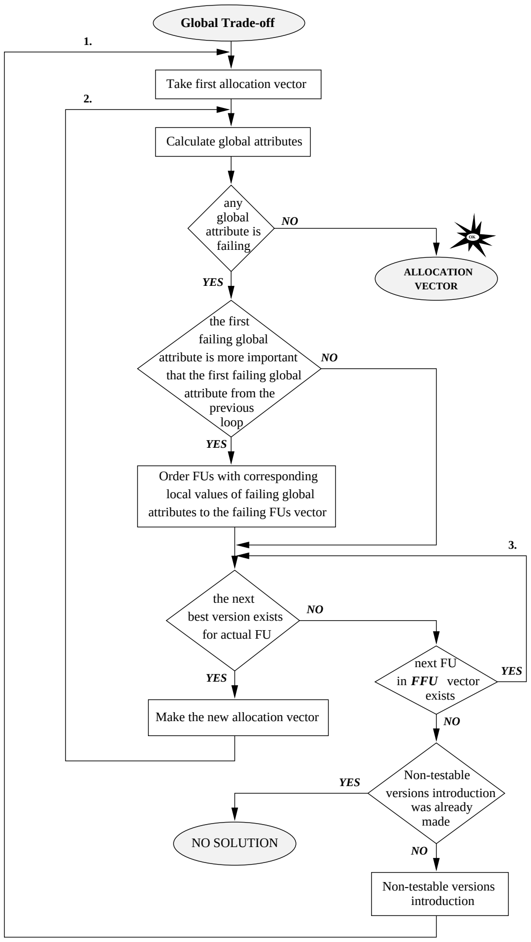
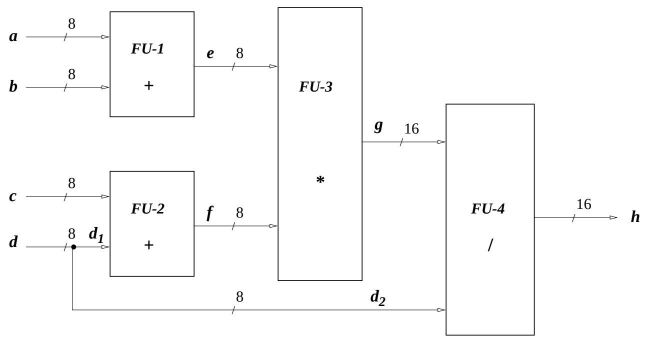


Figure 1: Simplificated global trade-off flowchart

The first loop is to return the whole algorithm to the starting point in the case of the non-testable versions introduction. In this case the order of FU versions in the matrix M is changed and it is necessary to restart the global trade-off. The second loop is the main loop in the global trade-off algorithm. In this loop new allocation vectors are made by replacing failing FUs versions in these vectors by the good ones. This loop, in fact, carries out the vertical shift from the top to the bottom of the matrix M. The third loop carries out the bidirectional horizontal shift in the matrix M. It is defined to change the actual FU.

4 Results

Let us consider the combinational circuit from Figure 2(a). For simplification reason, only to demonstrate the trade-off process, there are no registers and multiplexors in this example. From previous allocation step, the virtual FUs together with their attributes are extracted (see Figure 2(b)).



(a)

	virtual FU attributes			
NAME	FU1	FU2	FU3	FU4
AREA	200	200	300	400
FUNCTION	ADD	ADD	MUL	DIV
MAX-DELAY	10	10	10	17
INNBR	16	16	32	32
OUTNBR	8	8	16	16

(b)

Figure 2: (a) Architecture of example circuit, (b) Attributes of extracted virtual FUs

Then the corresponding FU versions are searched in the available FUs library (see Figures 3(a) and 4).

	Requirements	Weights
TEST-METHOD	BIST	10
MAX-DELAY	75	8
BIST-STYLE	PS-RND	0
SCAN-STYLE		
FULL-SCAN		
PART-SCAN		
RECONF-SCAN		
SCAN-CHAINS		
CONTROLAB	90	0
OBSERVAB	90	0
AREA-OVER	15	10
CLK-CYCLES	1	0
EXTRA-IO	2	0
NETWORK	BUS	0
TRIST-BUS	NO	0
INTERN-LOOPS	1	0
RESET	NO	0
RETIMING	NO	0
TEST-LENGTH	30	0
TEST-TIME	50	0
FAULT-COVER	90	0

Dashed box on the right-hand column represents fixed requirement.

(a)

	Extracted FUs from library for FU1 and FU2				
NAME	ADD1	ADD2	ADD3	ADD4	ADD5
AREA	240	250	230	200	220
FUNCTION	ADD	ADD	ADD	ADD	ADD
MAX-DELAY	20	45	70	10	19
WIDTH	700	700	700	700	700
INNBR	16	16	16	16	16
OUTNBR	8	8	8	8	8
TEST-METHOD	BIST	BIST	BIST	NON_TEST	BIST
BIST-STYLE	PS-RND	PS-RND	PS-RND		PS-EXH
SCAN-STYLE					
FULL-SCAN					
PART-SCAN					
RECONF-SCAN					
SCAN-CHAINS					
CONTROLAB	80	90	60		80
OBSERVAB	70	80	50		70
AREA-OVER	20	25	15		10
FAUL_COVER	75	90	50		50
CLK-CYCLES	1	1	1		1
EXTRA-IO	3	3	4		2
NETWORK	BUS	BUS	BUS		BUS
TRIST-BUS	NO	NO	NO		NO
INTERN-LOOPS	1	1	1		2
RESET	NO	NO	NO		NO
RETIMING	NO	NO	NO		NO
TEST-LENGTH	20	30	50		50
TEST-TIME	40	60	100		100

(b)

Figure 3: (a) User requirements, (b) Extracted FUs versions from library for FU1 and FU2

User requirements and weights are specified in figure 3(a). For simplification the trade-off is made only with $area\_over$ (area overhead) and max_del (maximal delay) attributes. The remaining attributes are eliminated by setting up the corresponding weights to 0. The result of the local trade-off is the matrix M of ordered FUs versions:

$$M = \begin{pmatrix} ADD5 & ADD5 & MUL3 & DIV4 \\ ADD3 & ADD3 & MUL1 & DIV3 \\ ADD1 & ADD1 & MUL4 & DIV2 \\ ADD2 & ADD2 & MUL2 & DIV1 \end{pmatrix}$$

	Extracted FUs from library for FU-3				
NAME	MUL1	MUL2	MUL3	MUL4	
AREA	360	390	345	360	
FUNCTION	MUL	MUL	MUL	MUL	
MAX-DELAY	35	25	60	40	
WIDTH	700	700	700	700	
INNBR	16	16	16	16	
OUTNBR	16	16	16	16	
TEST-METHOD	BIST	BIST	BIST	BIST	
BIST-STYLE	PS-RND	PS-RND	PS-RND	PS-EXH	
SCAN-STYLE					
FULL-SCAN					
PART-SCAN					
RECONF-SCAN					
SCAN-CHAINS					
CONTROLAB	99	98	65	98	
OBSERVAB	90	90	55	90	
AREA-OVER	20	30	15	20	
FAUL_COVER	95	95	60	90	
CLK-CYCLES	1	1	1	1	
EXTRA-IO	3	3	4	2	
NETWORK	BUS	BUS	BUS	BUS	
TRIST-BUS	NO	NO	NO	NO	
INTERN-LOOPS	1	1	1	2	
RESET	NO	NO	NO	NO	
RETIMING	NO	NO	NO	NO	
TEST-LENGTH	20	30	50	50	
TEST-TIME	40	60	100	100	

(a)

	Extracted FUs from library for FU-4				
NAME	DIV1	DIV2	DIV3	DIV4	DIV5
AREA	560	500	460	420	400
FUNCTION	DIV	DIV	DIV	DIV	DIV
MAX-DELAY	30	35	26	31	17
WIDTH	700	700	700	700	700
INNBR	16	16	16	16	16
OUTNBR	8	8	8	8	8
TEST-METHOD	BIST	BIST	BIST	BIST	NON_TEST
BIST-STYLE	PS-RND	PS-RND	PS-RND	PS-EXH	
SCAN-STYLE					
FULL-SCAN					
PART-SCAN					
RECONF-SCAN					
SCAN-CHAINS					
CONTROLAB	95	98	65	75	
OBSERVAB	90	93	55	65	
AREA-OVER	40	25	15	5	
FAUL_COVER	90	95	60	70	
CLK-CYCLES	1	1	1	1	
EXTRA-IO	3	3	4	2	
NETWORK	BUS	BUS	BUS	BUS	
TRIST-BUS	NO	NO	NO	NO	
INTERN-LOOPS	1	1	1	2	
RESET	NO	NO	NO	NO	
RETIMING	NO	NO	NO	NO	
TEST-LENGTH	20	30	50	50	
TEST-TIME	40	60	100	100	

(b)

Figure 4: Extracted FUs versions from library for FU3 and for FU4

The first allocation vector is $av_1 = (ADD5, ADD5, MUL3, DIV4)$, then $area\_over = 11.36$ % and $max\_del = 110$. User requirements for the global area overhead is 15 % and for the global maximal delay is 75 ns. During the searching of failing global attributes, one attribute was identified like failing and stored to the Failing Global Attributes (FGA) vector$FGA = (max_del) = (110)$. The global attribute max_del depends only on the local max_del attributes. The local Failing Global Attributes ($LFGA$) matrix will contain only one column, so this matrix can be changed to the $LFGA$ vector, where $LFGA = (max\_del_{ADD5}, max\_del_{ADD5}, max\_del_{MUL3}, max\_del_{DIV4}) = (19, 19, 60, 31)$. All FUs are ordered according to their failing contribution to the maximal delay problem (according to the $LFGA$ vector) to the Failing FUs (FFU) vector: $FFU = (FU3, FU4, FU1, FU2)$.

The next best version for FU3 (from the maximal delay point of view) is MUL1. The allocation vector is $av_2 = (ADD5, ADD5, MUL1, DIV4)$. Then $area\_over = 10.91$ % and $max\_del = 80$. The FGA vector contains again the attribute max_del. $FGA = (max\_del) = (80)$ such that $LFGA = (max\_del_{ADD5}, max\_del_{ADD5}, max\_del_{MUL1}, max\_del_{DIV4}) = (19, 19, 31, 30)$.

$$FFU = (FU3, FU4, FU1, FU2) \tag{1}$$

The next best FU version for FU3 from the maximal delay point of view after MUL1 is MUL2. The new allocation vector is $av = (ADD5, ADD5, MUL2, DIV4)$. Then $area\_over = 16.36$ % and $max\_del = 75$ and $FGA = (area\_over) = (16.36)$. In this point, there is no more problem with the maximal delay, but there is a problem with the area overhead. The area overhead attribute has higher weight than the maximal delay attribute. Return to the FFU vector (1) to search next best FU version for FU3. There is not a next best version so we pass to the next FU from the FFU vector (FU4). The next best FU version for FU4 from the maximal delay point of view after DIV4 is DIV3.

The new allocation vector is $av_3 = (ADD1, ADD1, MUL1, DIV3)$, then $area\_over = 14.54$ % and $max\_del = 75$. The FGA vector is empty. All problems with failing global attributes are solved, the global trade-off finishes with the allocation vector av_3.

More examples can be found in [20].

5 Conclusion

This paper has focused on the trade-off algorithm, which, added to the allocation for testability methodology presented in [3] and global test scheduling and control methodology presented in [2], forms a global HL-SFT environment [1]. As stated in Section 2, contrarily to other approaches reported in the literature ([4], [5],

[6] and [7]), 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.

References

[1] 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.

[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] P. Bukovjan, M. Marzouki, and W. Maroufi. Allocation for testability in high-level synthesis process. In *Proceedings from Electronic Devices and Systems International Conference*, Brno, Czech Republic, June 1998.

[4] 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.

[5] 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.

[6] 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.

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

[8] M. Abramovici, M.A. Breuer, and A.D. Friedman. *Digital System Testing and Testable Design*. An imprint of W.H. Freeman and Company, New York, 1990. ISBN 0-7167-8179-4.

[9] C.A. Papachristou, S. Chiu, and H. Harmanani. Syntest: A method fot high-level synthesis with self-testability. In *Proceedings of IEEE International Conference on Computer Design*, pages 458–462, October 1991.

[10] I.G. Harris and A. Orailoglu. Syncbist: Synthesis for concurrent built-in self-testability. In *Proceedings of European Design Automation Conference*, pages 101–104, Cambridge, MA, Octobre 1994.

[11] L. Avra. Allocation and assignment in high-level synthesis for self-testable data path. In *Proceedings of the International Test Conference*, pages 463–472, 1991.

[12] T.-C. Lee, N.K. Jha, and W.H. Wolf. Behavioral synthesis of highly testable data paths under the non-scan and partial scan environments. In *Proceedings of Design Automation Conference*, pages 292–297, Dallas, June 1992.

[13] 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.

[14] M.L. Flottes, R. Pires, B. Rouzeyre, and L. Volpe. Scanning datapaths : A fast and effective partial scan technique. In *Proceeding of Design Automation and Test in Europe (DATE 98),*, Paris, February 1998.

[15] S. Bhatia and N.K. Jha. Genesis: A behavioral synthesis system for hierarchical testability. In *Proceedings of European Design and Test Conference*, pages 272–276, February 1994.

[16] V. Chickermane, J. Lee, and J.H. Patel. Addresing design fot testability at the architectural level. *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, 13(7):920–934, July 1994.

[17] P. Vishakantaiah, T. Thomas, J.A. Abraham, and M.S. Abadir. Ambiant: Automatic generation of behavioral modification for testability. In *Proceeding of International Computer Design*, pages 63–66, Boston, October 1993.

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

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

[20] 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.