Introducing Redundant Computations in a Behavior for Reducing BIST Resources*

Ishwar Parulkar†
Sun Microsystems
Sunnyvale, CA 94086
ishwar.parulkar@eng.sun.com

Sandeep K. Gupta and Melvin A. Breuer
University of Southern California
Los Angeles, CA 90089-2562
{*sandeep, mb*}*@poisson.usc.edu*

Abstract

The degree of freedom that can be exploited during scheduling and assignment to minimize BIST resources is often limited by the data dependencies of a behavior. We propose transformation of a behavior by introducing redundant computations such that the resulting data path requires few BIST resources. The transformation makes use of *spare capacity* of modules to add redundancy that enables test paths to be shared among the modules. A technique is presented for introducing redundant computations that reduce the BIST resource requirements of a data path without compromising the latency and functional resource constraints.

1 Introduction and Motivation

A typical data path contains registers and functional modules, such as adders and multipliers, that are selected from a pre-designed library. One cost-effective way of testing such data paths employs built-in self-test(BIST). An important consideration in the application of a BIST technique is the area overhead incurred in modifying functional registers to BIST resources, i.e. registers that generate test patterns and compress test responses. A data path synthesized without any consideration for BIST resources can have high BIST area overhead. The scheduling and assignment stages of high-level synthesis can incorporate test requirements thereby reducing BIST area overhead [1],[2],[3],[4]. However the degree of freedom available during scheduling and assignment for minimizing BIST resources is often limited by the data dependencies of a behavior. In such cases, alternate behavioral descriptions need to be explored. At the logic level, it has been shown that introduction of redundancy can be beneficial for certain testability objectives [5]. Transformations have been applied at the behavioral level for optimization of area, performance, fault tolerance and partial scan overhead [6],[7],[8],[9]. In this paper, we propose a transformation aimed at optimizing the cost of BIST resources required to make a synthesized data path self-testable.

*This work was supported by the Advanced Research Projects Agency and monitored by the Department of the Army, Ft.Huachuca, under Contract No. DABT63-95-C-0042. The information reported here does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

†This work was done when the author was with the University of Southern California, Los Angeles, CA 90089.

1.1 Minimal Intrusion BIST

Minimal intrusion BIST involves the modification of only a subset of the functional data path registers to perform test functions. Depending on the test function, four different types of test registers (called BIST resources) are possible: 1) test pattern generation capability only (TPG), 2) test response compression or signature analysis capability only (SA), 3) test pattern generation and response compression capability at different times (BILBO), and 4) simultaneous test pattern generation and response compression capability (CBILBO). Different mappings of test register type to functional registers exist so that all functional modules in the data path are tested. Selection of registers for testing a module largely depends on how these registers are connected to the rest of the modules. For example, from the registers available for test response compression of a module, it is beneficial to choose one that can also compress responses from some other module, or maybe generate test patterns for some other module. For minimizing BIST area overhead, the design is analyzed globally to determine the BIST resources for each module such that all functional modules are tested with a minimum BIST area overhead.

1.2 Spare Capacity of Modules

Modules in most data paths perform useful computation in some clock cycles and are idle during other cycles. The percentage of time a module is idle is called the *spare capacity* of the module. For example, consider the DFG shown in Fig. 1(a). Suppose that a data path is synthesized from this DFG that executes in 3 clock cycles using one adder module A, one multiplier mod-

1-58113-049-x-98/0006/$3.50

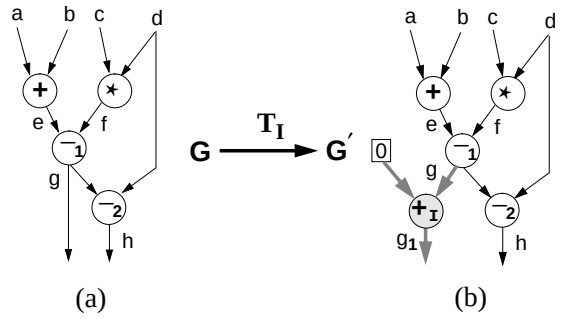

Figure 1: I-transformation

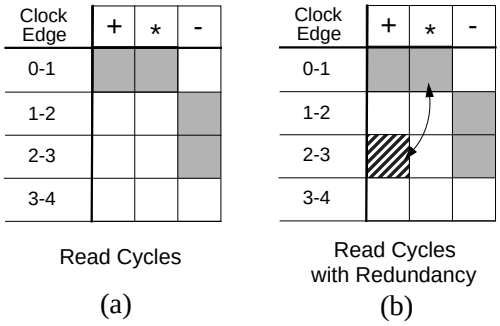

Figure 2: Read Cycles of Operations

ule M and one subtractor module S. Modules A and M have a spare capacity of 66% since they perform computations only in clock cycle 1 and are idle during clock cycles 2 and 3. Module S has a spare capacity of 33%. By performing *redundant* computations in the modules during the clock cycles they are not performing *useful* computations, sharing of paths that carry test data can be increased.

Fig. 2(a) shows the state (busy or idle) of the read ports of operations in DFG of Fig. 1(a), assuming a minimum latency schedule. The shaded boxes represent when the input ports (read ports) are busy. Note that the read ports of the addition and multiplication operations are busy at the *same* time. A redundant read can be introduced for the addition operation during idle time (shown hashed in Fig. 2(b)). Since the redundant read of the addition occurs at a different time than the read of the multiplication, the two read actions can share hardware for data transfer which could be used to transfer test data for both the operations. A similar scenario exists for write cycles of operations.

2 I-transformation and BIST Resources

Many operations used in a behavioral specification have an identity value associated with them. For example, a multiplication operation has an identity value of 1 and addition operation has an identity value 0. If one of the operands of these operations is equal to the identity value then the output of the operation is the same as the other operand.

Definition 1 *An identity mode DFG node* (**I-node**) *is a node whose operation type has an identity value and one of its operands is set to a constant corresponding to the identity value.*

Definition 2 (**I-transformation**) *Replace edge e of a DFG with an I-node such that the source of e is the source of an input of the I-node and the destination of e is the destination of the output of the I-node. The other input of the I-node corresponds to the identity value of the I-node operation.*

We denote the transformation as $G \stackrel{T_I}{\rightarrow} G'$. Fig. 1(a) shows a DFG G which is transformed into the DFG G' in Fig. 1(b) by the addition of an I-node (shown shaded in the figure). The variable input of the I-node, g is the same as the output variable of $-_1$ and a new variable g_1 is created that is the output of the new addition I-node. The value of the variable g_1 in G' is the same as variable g in G and hence G' and G are functionally equivalent. The concept of exploiting identity modes of functional modules in *synthesized* data paths for transporting test data was suggested in [10]. An I-node is an analogous concept at the behavioral level with the distinction that the identity mode is used to move functional data without changing it.

We propose the use of I-transformations to modify DFGs which, when followed by scheduling and hardware assignment, will lead to cost-effective BIST data paths. The basic idea is to introduce redundant paths that can be used to 1) transport test data, or 2) enable sharing of non-redundant paths in transporting test data. Corresponding to 1) and 2), we have two types of I-transformations, Type 1 and Type 2, respectively.

2.1 Type 1 I-transformation

The DFG shown in Fig. 1(a) illustrates the Type 1 I-transformation. The minimum latency achievable for this DFG is 3. An adder, a multiplier and a subtractor are required to implement the scheduled behavior. A data path synthesized with the objective of minimizing BIST area overhead using assignment techniques in [3] is shown in Fig. 3(a). It can be seen that the adder and multiplier require distinct registers for test pattern generation. Also two distinct registers are required to compress their test responses. The scheduling and assignment stages of synthesis cannot improve the sharing of BIST resources between the modules.

Fig. 1(b) shows the same DFG with an I-node inserted. A data path synthesized using this modified DFG is shown in Fig. 3(b) where the functionally redundant path is shown highlighted. The redundant path from 1) variable g to the I-node, and 2) from the I-node to variable g_1, make it possible for the adder and multiplier to share BIST resources. Variables f (output variable of the multiplication operation) and g_1 (output variable of the addition operation created

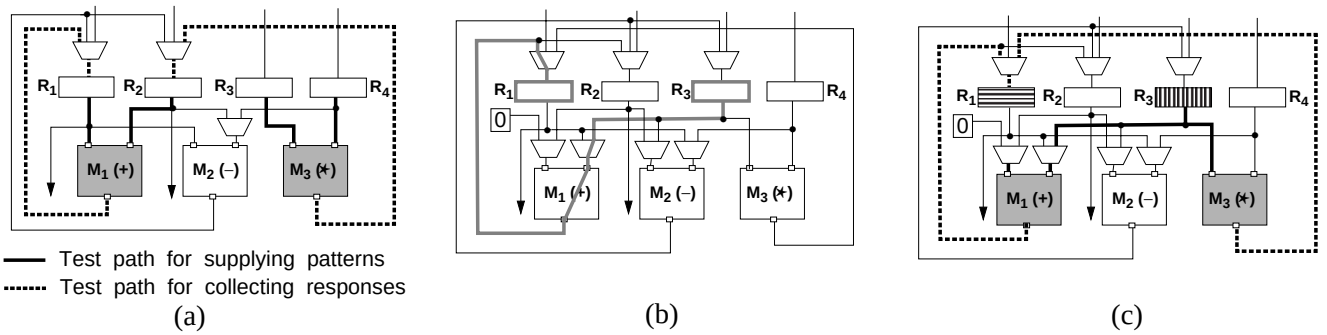

Figure 3: Data paths synthesized from DFG1 (Type 1 I-transformation)

by redundancy) can be assigned to the same register, R_1, which can compress responses for both the modules. Similarly, input variables of multiplication and addition, c and g, respectively, can be assigned to the same register, R_2, that can generate test patterns for both modules. As shown in Fig. 3(c), the redundant path through the adder can be shared with a functional path through the multiplier, resulting in common BIST resources.

2.2 Type 2 I-transformation

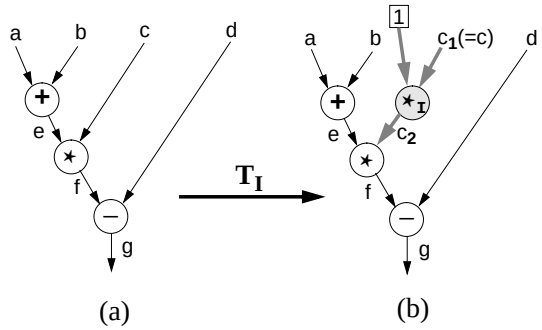

Figure 4: Type 2 I-transformation on DFG2

In contrast to Type 1 transformation, Type 2 transformation creates redundant paths that *enable* sharing of other non-redundant functional paths as test paths. In this case the redundant path does not transport any test data. This is illustrated using the DFG in Fig. 4(a). The minimum latency is 3 and the module requirement is one adder, one multiplier and one subtractor. Fig. 5(a) shows a data path synthesized from this DFG. It can be seen that it is possible to assign registers such that the test response compression of the adder and multiplier can be done by the same register, namely, R_1. However, a distinct register is required for test pattern generation for the right input port of the adder and multiplier, R_2 and R_3, respectively. Fig. 4(b) shows the DFG modified using a multiplication I-node. The corresponding synthesized data path is shown in Fig. 5(b) with the redundant path highlighted. The addition of redundancy creates a data transfer to the right input port of the multiplier. This enables the input variable of multiplication, c_2, and an input variable of addition, b, to be assigned to the same register, which can be shared as a test pattern generation resource between the adder and multiplier (Fig. 5(c)).

3 Properties of I-transformation

The I-transformation changes the structure and some properties of a DFG. To apply the transformation in an efficient and beneficial manner, we have characterized the effect of the I-transformation on a DFG. A *critical path* in a DFG is the longest path in the DFG and the length of the critical path is a lower bound on the achievable latency of a DFG.

Property 1 *If an I-node is introduced in a non-critical path of a DFG, then the minimum latency of the DFG remains unchanged.*

This property enables the introduction of I-nodes such that they do not effect the minimum latency of the DFG. For optimizing BIST area overhead we need to maximize the sharing of registers as BIST resources between different modules. This depends on the flexibility available in assigning variables to registers.

Property 2 *An I-transformation increases the number of storage variables in the DFG by 1, but the minimum number of registers required to store all the variables remains unchanged.*

Property 2 demonstrates that introduction of I-nodes is not harmful in terms of storage resource requirement of a data path. The minimum number of registers required to implement a data path before and after any I-node transformation is the same. Thus we see that I-nodes can be introduced such that the number of modules, number of registers and the latency remains the same. However the transformation can have an adverse effect on the interconnect complexity.

Property 3 *The number of multiplexer inputs at the input ports of a module to which n_I I-nodes have been assigned increases by at most $n_I + 1$.*

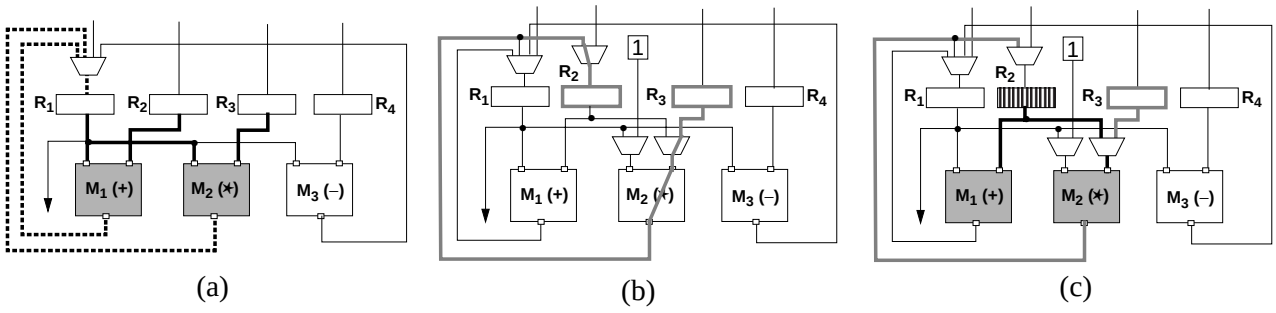

Figure 5: Data paths synthesized from DFG2 (Type 2 I-transformation)

The increase in the number of multiplexers is due to the fact that one additional operation (corresponding to the I-node) is assigned to a module and additional operands need to be supplied to the module. However, one of the operands of the I-node is a constant corresponding to the identity of the I-node operation. Taking advantage of this fact, multiplexer area can be optimized. This can be achieved in two ways.

1. Storing identity value in primary input register: The identity value of an I-node is required only during the clock cycle in which the redundant computation is performed. Hence a *primary input register*, such as R_1 in Fig. 6(a), that does not contain a valid value during that clock cycle can be used to supply the identity value during that clock cycle and the extra multiplexer input can be eliminated.

2. Logic optimization of multiplexers: Some times it is not possible to find a primary input register that does not hold a valid value during the clock cycle in which the redundant computation is performed. In such cases multiplexer overhead can be reduced by logic optimization as shown in Fig. 6(b).

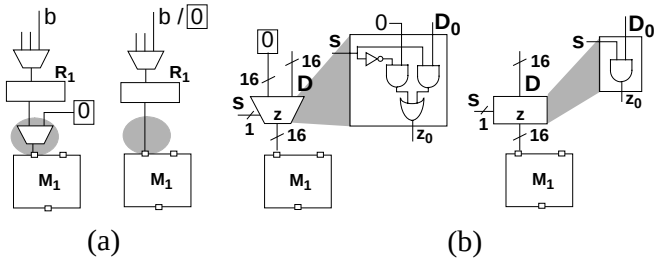

Figure 6: Optimization of multiplexers

4 Strategy for Introduction of I-nodes

The strategy for introduction of I-nodes involves two tasks: 1) identify pairs of operations that will have a potential problem with sharing of BIST resources in subsequent stages of synthesis; and 2) resolve the problem identified in Task 1 by introducing an I-node.

4.1 Identifying BIST Resource Sharing Problems (Task 1)

A BIST resource sharing problem between a pair of modules arises out of the inability to share registers by the input and output variables of pairs of operations assigned to those modules. Hence, the task of identifying BIST resource sharing problems in a DFG involves identification of pairs of operations that have 1) a high probability of being assigned to different modules, and 2) a high probability of having disjoint sets of BIST resources that cannot be shared.

First, we identify pairs of operations that get assigned to different modules with *certainty*. If the operations are of different types then they will definitely get assigned to different modules. If they are of the same type, then they will get assigned to different modules with certainty only if they are concurrent in all possible schedules. We target only such pairs of operations for the following two reasons. Firstly, since many parameters of the data path are not fixed at this point, it is efficient to use information that has a *high degree of certainty*. Secondly, from our experiments we find that introduction of I-nodes follows the *law of diminishing returns*. As more and more I-nodes are added, the resulting savings in BIST area overhead become marginal. Hence we target those pairs of operations that have the problem to the *highest degree* and with the *highest certainty*.

In [11], a procedure for estimating the number of *registers* for an unscheduled DFG is presented. Based on that we have derived a probabilistic estimate of the number of *BIST resources* required for a pair of operations [12]. A higher estimate of BIST resources corresponds to a higher degree of difficulty in sharing these resources between a pair of operations. From the pairs of operations that are assigned to different modules with certainty, a pair with the highest degree of difficulty of sharing BIST resources is selected as the bottleneck to be resolved in Task 2.

4.2 Choosing an I-node (Task 2)

The second task of choosing an I-node to resolve the problem identified in Task 1 has two parameters: 1) the operation type of the I-node , e.g. addition or multiplication, and 2) the position of the I-node in the DFG. As shown in section 3, the introduction of an I-node anywhere in the DFG preserves its functionality but changes some of the properties of the original DFG and affects the synthesized data path in different ways. To ensure that I-nodes are introduced in a manner that is beneficial for BIST but not harmful in terms of functional constraints, the introduction of an I-node should 1) not violate timing constraints, 2) not violate resource constraints, 3) not limit scheduling mobility, 4) maximize re-usage of interconnect to keep interconnect complexity low, and 5) optimize BIST area overhead. Criteria 1 and 2 are strict in the sense that I-nodes that violate them are not allowed. Criteria 3 and 4 are desirable and I-nodes with least impact on scheduling mobility and interconnect complexity are preferred. From the I-nodes that qualify for the first four criteria, an I-node with the highest impact on the BIST resource sharing problem is selected. The impact of an I-node on BIST resources is quantified using the same probabilistic BIST resource estimate that was used in Task 1 to identify a pair of operations that had difficulty in sharing BIST resources. For every candidate I-node the BIST resources are estimated for the pair of operations under consideration and an I-node that decreases the degree of difficulty of sharing BIST resources the most is selected for insertion into the DFG.

Introduction of I-nodes is an iterative process. The scheduling mobility of a DFG is used to determine whether more I-nodes should be added. I-nodes are introduced only while the scheduling mobility remains over a certain predetermined threshold. If the scheduling mobility drops below the threshold, no more transformations are performed.

5 Experimental Results

The I-node transformation was performed on some example DFGs. *ex* is the example from Fig. 1 and *diffeq*, *Tseng* and *AR_filter* are standard high-level synthesis benchmarks. All DFGs were analyzed for BIST resource sharing problems and the DFGs were transformed by introducing I-nodes. TOPS [13], a synthesis-for-test tool was used to synthesize data paths from the original and transformed DFGs which were then made self-testable using minimal intrusion BIST. Components designed using a macro-cell library supplied by LSI Logic Corp. were used for synthesis and the area numbers in the tables are given in cell units [14].

Table 1 shows the components of all synthesized data paths and the BIST resources required to make them self-testable. The name extension of '0' for a DFG refers to the original untransformed version while the rest are after performing I-transformations. The number and type of redundant operations are shown in the column labeled *I-nodes*. Table 2 shows the actual areas of the data paths before and after they were made self-testable The second column in Table 2 indicates whether multiplexer optimization was performed on the data path. It can be seen that introduction of one addition I-node in *ex*-1 results in a significant reduction in BIST area overhead and also total area. However, adding one more redundant addition in *ex*-2 does not give any significant improvement over *ex*-1. In *ex*-3, more redundant operations are added which result in a very large reduction in BIST area overhead. The multiplexer complexity goes up and the savings in total area are not as significant. The experiments on *ex* indicate that introduction of I-nodes follows the *law of diminishing returns*.

Version *diffeq*-1 has one redundant addition and one multiplication as compared to *diffeq*-0. The savings of about 6% in area are achieved at the expense of about 22% reduction in the scheduling mobility of the DFG. For the *AR_filter*, after adding 6 redundant computations, the BIST area overhead is reduced significantly (from 16% to 8.5%) sacrificing 60% of the scheduling mobility. However about only 2% saving in total area is achieved after multiplexer optimization. The *AR_filter* has a high degree of freedom in the scheduling and assignment stages and hence there is not much scope for BIST resource optimization by transforming the behavior. The last four columns in Table 1 show the number of different *types* of BIST resources in the minimal intrusion BIST solutions of all the synthesized data paths. As stated in section 1.1, depending on the test function, four different types of BIST resources with varying costs are possible. A significant reduction in BIST area overhead is achieved even if the number of cheaper BIST resources (such as TPG and SA) increases because the number of more expensive BIST resources (such as CBILBO) decreases.

6 Conclusions

We have demonstrated how the spare capacity of modules available in data paths can be exploited at the behavioral level for reducing BIST resources. A transformation that utilizes the spare resource capacity in a behavior and introduces redundant computations has been described. A technique based on identifying potential testability problems and resolving them by adding I-nodes has been presented. Experimental results demonstrate that very few well-placed I-nodes can significantly reduce the BIST resource requirements of a data path. The transformation is especially useful in cases where there is not enough freedom in the subsequent scheduling and assignment stages to optimize for BIST resources.

Table 1: Characteristics of synthesized data paths

DFG Version	Lat L	Reg	Modules					Muxes					I-nodes	BIST Resources			
			$*$	$+$	-	AND	OR	2:1	3:1	4:1	5:1	6:1		#C	#B	#T	#S
ex-0	3	4	1	1	1	-	-	6	0	0	-	-	-	1	1	2	0
ex-1	3	4	1	1	1	-	-	5	2	0	-	-	1+	0	1	2	1
ex-2	3	4	1	1	1	-	-	5	3	0	-	-	2+	0	0	2	2
ex-3	4	4	1	1	1	-	-	4	4	1	-	-	2+, 2$*$	0	0	2	1
diffeq-0	4	6	3	1	1	-	-	5	2	2	1	-	-	1	1	2	1
diffeq-1	4	6	3	1	1	-	-	10	2	1	-	-	1+,1$*$	0	1	3	2
Tseng-0	4	4	1	2	1	1	1	1	2	1	-	-	-	1	3	0	0
Tseng-1	4	4	1	2	1	1	1	2	2	1	-	-	1+	0	4	0	0
AR_filter-0	8	16	8	4	-	-	-	11	-	4	-	2	-	0	7	8	1
AR_filter-1	8	16	8	4	-	-	-	25	4	2	-	2	2+,4$*$	0	3	5	2

Table 2: Area comparisons of synthesized data paths

DFG Version	Mux opt.?	Area before BIST A	BIST overhead B	% BIST overhead $(B/A \cdot 100)$	Total area $C = A + B$	% decrease in total area $(C^0 - C)/C^0)^\dagger \cdot 100$
ex-0	-	6812	2272	25.01	9084	-
ex-1	No	7260	1088	13.03	8348	8.10
ex-2	No	7580	768	9.20	8348	8.10
ex-3	No	8188	576	6.57	8764	3.53
ex-1	Yes	**7228**	1088	**13.08**	**8316**	**8.46**
ex-2	Yes	**7324**	768	**9.49**	**8092**	**10.92**
ex-3	Yes	**7900**	576	**6.80**	**8476**	**6.70**
diffeq-0	-	6033	1072	15.09	7105	-
diffeq-1	No	5916	736	11.06	6652	6.38
diffeq-1	Yes	**5900**	736	**11.09**	**6636**	**6.61**
Tseng-0	-	4078	1296	24.12	5374	-
Tseng-1	No	4094	1024	20.01	5118	4.76
Tseng-1	Yes	**3998**	1024	**25.62**	**5022**	**6.56**
AR_filter-0	-	14000	2656	15.95	16656	-
AR_filter-1	No	15504	1440	8.50	16944	-1.73
AR_filter-1	Yes	**14928**	1440	**9.65**	**16368**	**1.73**

$^\dagger C^0$ is the total area of Version 0 of each DFG.

Acknowledgments

The authors would like to acknowledge Prof. Edward J. McCluskey and Dr. LaNae J. Avra of the Center for Reliable Computing at Stanford University for providing the TOPS synthesis system.

References

[1] H. Harmanani and C. Papachristou. An Improved Method for RTL Synthesis with Testability Tradeoffs. In *Proc. Intn'l Conf. on Computer-Aided Design*, Nov. 1993.

[2] L. Avra. Allocation and Assignment in High-level Synthesis for Self-testable Data Paths. In *Intn'l. Symp. on Circuits and Systems*, Aug. 1991.

[3] I. Parulkar, S.K. Gupta, and M.A. Breuer. Data Path Allocation for Synthesizing RTL Designs with Low BIST Area Overhead. In *Proc. 32nd Design Automation Conf.*, pages 395–401, June 1995.

[4] I. Parulkar, S.K. Gupta, and M.A. Breuer. Scheduling and Module Assignment for Reducing BIST Resources. In *Proc. Design Automation and Test in Europe*, Feb. 1998.

[5] A. Karasniewski. Can Redundancy Enhance Testability? In *Proc. Intn'l Test Conf.*, Oct. 1991.

[6] A.P. Chandrakasan, M. Potkonjak, J. Rabaey R. Mehra, and R. Brodersen. Optimizing Power Using Transformations. *IEEE Trans. on Computer-Aided Design*, Jan. 1995.

[7] L. Guerra, M. Potkonjak, and J. Rabaey. High-level Synthesis for Reconfigurable Datapath Structures. In *Proc. Intn'l Conf. on Computer-Aided Design*, pages 26–29, Nov. 1993.

[8] R. Karri and A. Oraiglu. Transformation-based High-level Synthesis of Fault-tolerant ASICs. In *Proc. 29th Design Automation Conf.*, June 1992.

[9] S. Dey and M. Potkonjak. Transforming Behavioral Secifications to Facilitate Synthesis of Testable Designs. In *Proc. Intn'l Test Conf.*, Oct. 1994.

[10] M.S. Abadir and M.A. Breuer. A Knowledge-Based System for Designing Testable VLSI Chips. *IEEE Design & Test of Computers*, Aug. 1985.

[11] R. Moreno, R. Hermida, and M. Fernandez. Short Note: Register Estimation in Unscheduled Dataflow Graphs. *ACM Trans. on Design Automation of Electronic Systems*, July 1996.

[12] I. Parulkar, S.K. Gupta, and M.A. Breuer. *Using Redundany to Minimize BIST Resources in Data Paths.* CEng Tech. Report 97-15, Univ. of Southern California, Oct. 1997.

[13] L.J. Avra, L. Gerbaux, J-C. Giomi, F. Martinolle, and E.J. McCluskey. *A Synthesis-for-Test Design System.* Tech. Report CSL TR 94-622, Stanford University, May 1994.

[14] Data Book. *G10-p Cell-Based ASIC Products.* LSI Logic Corp., May 1996.