IEEE the 9th Asian Test Symposium (ATS 2000), pp. 229-234, Dec. 2000.

Strong Self-Testability for Data Paths High-Level Synthesis *

Xiaowei Li
Dept. of Computer Science
Peking University
Beijing 100871, China
Email: lxw@cs.pku.edu.cn

Toshimitsu Masuzawa and Hideo Fujiwara
Graduate School of Information Science
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-0101, Japan
Email: fujiwara@is.aist-nara.ac.jp

Abstract

In this paper, we introduce strong self-testability for data paths at register transfer level (RTL). A high-level synthesis scheme is proposed for producing such strongly self-testable data paths. This is achieved by incorporating testability constraints during processes of register assignment and interconnection assignment. This method is based on the use of test resources reusability to improve the self-testability of data path. Experimental results are presented to demonstrate the effectiveness of the proposed approach.

Keywords: Data path, high-level synthesis, strong self-testability

1 Introduction

High-level synthesis can explore a larger design space than lower-level synthesis. An inherently testable architecture may already exist in the design space, which can be derived by high-level synthesis to produce a highly testable circuit at low or even no area and/or delay penalty. As *automatic test pattern generation* (ATPG)-oriented approaches, strong testability was introduced in [1], and weak testability was introduced in [2]. Many methods, whether BIST-oriented or ATPG-oriented, operate by modifying the allocation process so that the synthesized circuit does not have some undesirable structural property.

The BIST-oriented approaches usually assume the presence of a *test pattern generator* (TPG) for test vector generation and a *test response compressor* (TRC) for response compression. The blocks that are required to perform a test (i.e., TPG and TRC) are known as *test resources*. Since the BIST logic is combined with the system logic, opportunities exist for the synthesis technique to generate hardware that can be shared by both the system and test operation, resulting in improved performance and reduced cost.

Area and test time are major overheads encountered when using BIST techniques. Approaches have been proposed either towards the *minimal (extra) area overhead solution* (MAOS), or the *minimal test (application) time solution* (MTTS). Both approaches have merits and limitations. In general, the area overhead of MTTS may be too high and the test time of MAOS may be too long. Neither is acceptable when both attributes are important.

The MAOS-oriented approaches focus on minimizing area overhead. The basic approach is to maximize the sharing of test registers resulting in a fewer number of registers being modified for BIST [3-8]. The goal is to reduce BIST area overhead without sacrificing the quality of the test. The main limitation is the test time may be too long. The MTTS-oriented approaches focus on exploiting test concurrency. The basic approach is to maximize the test concurrency resulting in a fewer test session [9]. The goal is to reduce test time. The main limitation is the area overhead may be too high.

MAOS-oriented approaches dominate current research. A prime concern in BIST implementation is the area overhead due to the modification of normal registers to be test registers. One of the difficulties is the *register self-adjacency* problem. To deal with it, methods have been proposed either to avoid producing such self-adjacent registers or to minimize their number during synthesis process. Avra [4] proposed a register allocation method that minimizes the number of self-adjacent registers in the design. Papachristou *et* al. [5] presented a combined register and ALU allocation method that generates self-testable designs that do not have any self-loops. Parulkar *et* al. [6] attempted to employ the

*This research was supported in part by Japan Semiconductor Academic Research Center (STARC) under research project, and in part by the Natural Science Foundation of China (NSFC) grant #69976002.

concept of I-path [10] to reduce the area overhead imposed by BILBO registers.

In this paper, we first introduce strong self-testability for data paths at RTL. The key aspect is to propose a high-level synthesis scheme for producing such strongly self-testable data paths. This is achieved by incorporating testability constraints during processes of register assignment and interconnection assignment. This method is based on the use of test resources reusability to improve the self-testability of data path. Given a scheduled *data flow graph* (DFG) and a module assignment, we assign variables to registers and data transfer to interconnection paths, such that the area overhead required for a strongly self-testable data path is minimal.

For each strongly self-testable data path, it has the following metrics

(1) 100% fault efficiency[1] for single stuck-at faults;
(2) At-speed testing;
(3) Short test time;
(4) Low extra area overhead;

The rest of this paper has been organized as follows. Section 2 introduces strong self-testability for data paths at RTL. In Section 3, we consider strong self-testability during high-level synthesis processes, including register assignment and interconnection assignment. Experimental results are presented in Section 4. Finally, conclusion and future work are discussed in Section 5.

2 Strongly Self-Testable Data Paths at RTL

2.1 Strong Self-Testability

A data path is the core of a processor, it is where all computations are performed. A typical data path consists of an interconnection of basic combinational functions, such as logic (AND, OR, XOR) or arithmetic operators (addition, multiplication, comparison, shift). RTL modules consist of functional modules and multiplexers. We only consider strong self-testability for functional modules.

Definition 1: A module is controllable if its output can be set to any desired value. A module is observable if any signal change at its inputs is reflected by a change at the output.

Definition 2: A *justification path* (J-path) of an n-bit width register r is a path that allows r to be assigned to the 2^n possible values from a *primary input* (PI) or a TPG. A *propagation path* (P-path) of r is a path that allows any change in r to be propagated from r to a *primary output* (PO) or a TRC.

An output port of a module M is observable if there exists an observable variable fed by this port such that its P-path do not contain M. A J-path is composed of a cascade of controllable modules. A P-path is composed of a cascade of observable modules. J-path and P-path are a generalization of the notion of I-path from [10].

Definition 3: A data path is *strongly self-testable,* if each module is random pattern testable, and there exists a test plan for each module, in which, for each input port of the module, there exists a J-path from a TPG to it, and for output port of the module, there exists a P-path from it to a TRC, such that those J-paths and P-path are mutually disjoint.

Random-pattern testability of the RTL modules present in the system is crucial for achieving complete fault coverage for the whole data path. Most RTL modules like adders, subtracters, multipliers, shifters, registers and register files are random-pattern testable. Meanwhile, modules like comparators, incrementers and decrementers are random-pattern resistant, but, they can be modified to be random-pattern testable by adequate test points insertion.

Here, for simplicity, we have following assumptions:

(1) In the Data Path G, each functional module M has two distinct input ports A and B, and has just one output port C;
(2) Each module is random-pattern testable.

2.2 Strongly Self-Testable Data Paths

The area overhead minimization problem for a strongly self-testable data path is to decide which registers to modify and the modes (TPG or TRC) to add, in such a way that every module in a given data path can be tested with minimal area overhead. We attempt to solve this problem by trying to modify registers in the data path which have the maximum sharing potential.

We define *role* of a register as its possible use as TPG or TRC and denote these roles by the letters p and r, respectively. A register instance is a particular register in a particular role. The two possible register instances of a register R_i are denoted R_{ip}, R_{ir}. The modification cost C_{ix} of a register R_i for a role x is defined as the amount of area overhead that would result from modifying R_i so that it has an additional modes represented by x, where x = p or r.

We associate a modification cost C_{ix} with each register. For a register of n bits, the cost of being implemented as a TPG (LFSR for instance) is roughly estimated as $2*(n\text{-}2)$ times the cost of a multiplexer. The cost of the register implemented as a TRC is roughly estimated as n-2 times the cost of a multiplexer.

Definition 4. A module is directly self-testable if

[1] Fault efficiency is the ratio of the number of faults identified as either detectable or redundant to the total number of faults.

there exists two J-paths directly from register R_i and R_j to its two distinct input ports, and there is a P-path directly from its output port to R_k. R_i and R_j are converted to R_{ip} and R_{jp}, and R_k is converted to R_{kr}. The R_{ip}, R_{jp} and R_{kr} are called a *test scenario* (TS) of the module.

Definition 5. A module is indirectly self-testable if there exists two J-paths from register R_i and R_j to its two distinct input ports through one and only one module, and there is a P-path from its output port to R_k through no more than one module. R_i and R_j are converted to R_{ip} and R_{jp}, and R_k is converted to R_{kr}.

The problem for finding strongly self-testable solutions is solved in three steps [11]. A synthesized strongly self-testable data path is shown in Fig.1.

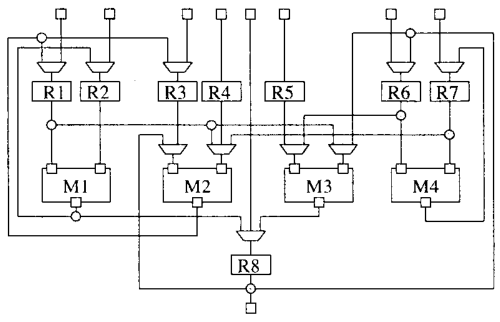

Fig.1 A Synthesized Data Path

Here is a possible strongly self-testable solution for the data path: for M1 self-testing, both R1 and R2 are direct TPGs and R8 is a direct TRC. For M2 self-testing, R7 is a direct TPG, and R6 is an indirect TPG (through M3 and R8) and R1 is a direct TRC. For M3 self-testing, both R6 and R1 are direct TPGs and R8 is a direct TRC. For M4 self-testing, both R6 and R7 are direct TPGs and R7 is also a direct TRC.

3 High-Level Synthesis for Strong Self-Testability

High-level synthesis starts with a behavioral description of a digital system and synthesizes a RTL data path. The synthesis model used in this work assumes single-cycle operations, individual register storage and point-to-point multiplexer (MUX) connectivity.

3.1 Testability Metrics on DFG

Variables in a behavior are potential test resources and contribute to BIST area overhead directly. Operations in a behavior are correlated to modules that form the functional module under test. Data transfers between variables and operations correspond to interconnection paths that are potential test paths. We refer to test registers as BIST resources, include those selected to be TPGs to modules and those selected to be TRCs from modules. Since minimal area overhead is our objective, it is not necessary to test all the modules in one test session.

For directing assignments to a strongly self-testable data path with low area overhead, a mechanism that relates variables to test functionality in a data path is required. For this purpose, we view each variable as a test variable in addition to a functional variable to be stored in a register. Basically, there are two kinds of test variables:

Definition 6: A variable is a *P-variable*, i.e., is a candidate for TPG, if there is a J-path from it to an input port of a module. A variable is an *R-variable* i.e., is a candidate for TRC, if there is a P-path from the output port of a module to it.

Usually, LFSR, BILBO and CBILBO can be a P-variable and MISR, BILBO, and CBILBO can be a R-variable, but have different area costs due to modification.

Definition 7: *Reusability* R(v) of a variable v is the sum of the number of modules for which v is a *P-variable* and the number of modules for which v is a *R-variable*, both directly and indirectly. The maximum clique size M(v) of a variable v is the maximum clique in the *variable conflict graph* (VCG) which v belongs.

The reusability R(v) reflects the number of modules for which the variable v can act as TPG and the number of modules for which it can act as TRC. For example, in Fig.2, the variable h is potentially to be TPG only for "module" (+1), R(h) is 1. The variable i is potentially to be TRC for "module" (*1) and to be TPG for "module" (*3) and (–1), R(i) is 3.

3.2 Problem Formulation

One way of making a data path strongly self-testable is to modify all functional registers and give them test capabilities. Such an approach is an overkill in terms of the area overhead. Usually, portions of a data path, such as multiplexers, interconnect and functional registers can be easily tested using functional patterns. The components that are hard to test using functional patterns are the modules such as ALUs, adders and multipliers. Hence, we modify a subset of the registers and give them the capability of generating pseudo-random test patterns for the modules in the data path and for compressing the responses from these modules into signatures.

This problem could be formulated as follows:

(1) Given a scheduled DFG, and a module assignment has been done, any module which is random-pattern-resistant has been replaced by random-pattern testable one.

(2) Exploit *test register reusability* in register assignment process, the optimization *objective* is to minimize the total number of registers needed to be modified as test registers, while keeping the synthesized data path strongly self-testable.
(3) Register assignment is followed by minimum interconnection assignment. The effect of interconnection assignment on BIST registers is exploited.

3.3 Register Assignment

Given a scheduled DFG. We assume that module assignment is done without any testability consideration. Because registers can be viewed as potential test resources only after the module assignment is fixed. Here, we consider BIST area minimization in register assignment process for the proposed strong self-testability.

The register assignment problem can be modeled as coloring the VCG. Each node in the VCG represents an edge from the DFG that crosses a clock cycle boundary. A *conflict* edge between two nodes in the VCG indicates that two variables associated with those nodes cannot be stored in the same register. All nodes with the same color can be mapped to the same register in the final implementation. The number of colors is the number of registers required in the data path design. A coloring of VCG corresponds to a valid register assignment with each color corresponding to a register.

For the given the VCG, several algorithms exist that can almost always color the graph with a minimum number of colors in polynomial time. However, a k-colorable graph, where k is the chromatic number of the graph, may have several different k-colorings, each coloring scheme represents a different assignment of nodes to registers. Because of using different number of multiplexers and interconnects, these colorings may represent data path designs of different sizes. Furthermore, a coloring scheme that uses more than k colors may represent a smaller implementation. Therefore, techniques must be employed to guide the nodes coloring algorithm to reach the lowest-cost implementation.

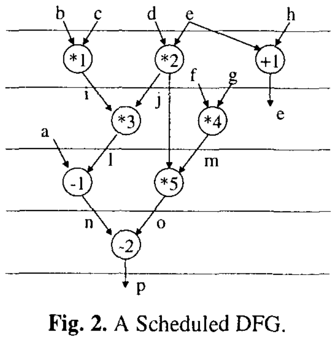

Fig. 2. A Scheduled DFG.

Consider the scheduled DFG shown in Fig.2, with module assignment: {M1:(*1,*3), M2:(*2,*4,*5), M3:(-1,-2), M4:(+1)}. We associate a reusability R(v) with each variable v (node in the VCG), as shown in the Fig.3. Using this measure the assignment process can be guided by choosing merges that result in large increases in reusability of registers. Usually, a VCG is an interval graph. The minimum coloring algorithm on interval graph is a greedy algorithm [12]. A node n on a graph G is simplicial, if its adjacency set induces a clique in G. The adjacency set is the set of all nodes that are connected to n. An ordering of the nodes such that each node is a simplicial node of the remaining graph is called *a perfect node elimination scheme* (PNES) [13]. An interval graph has many such PNESs. The optimal coloring algorithm constructs one such scheme arbitrarily and colors the nodes greedily in the reverse order.

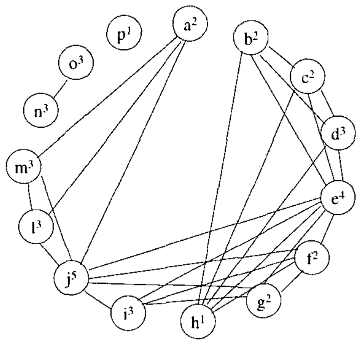

Fig. 3. The corresponding VCG.

The nodes are ordered such that if x is before y, then $R(x) \leq R(y)$ and if $R(x)=R(y)$ then $M(x) \leq M(y)$. At every step of constructing a PNES, there is a choice of simplical nodes. The PNES is determined such that at each step a simplicial node that is earliest in this order is selected. Since nodes are colored in the reverse PNES order, this results in nodes with higher reusability to be considered earlier when there is maximum flexibility in the assignment of colors. The node (i+1) is assigned in the following way. If (i+1) conflicts with all registers, then a new register is created. Otherwise, pick a register such that the $\Delta R(x^i) = R(x^{i+1})-R(x^i)$ is maximum. Such a x^i corresponds to a register that can best utilize (i+1) to improve its reusing as a test resources. This process is near optimal since it relies on a PNES of a VCG.

In the above example, the procedure results in a minimum assignment of colors. Eight registers are needed. {R_1:(j,o,c), R_2:(i,b), R_3:(d,m), R_4:(p), R_5:(a), R_6:(h,n), R_7:(e), R_8:(p,l,f)}. Fig.1 shows the resulting data path corresponding to this register assignment and the given module assignment. According to the definition 3, one possible strongly self-testable solution for the synthesized data path is as shown in Table 1:

Table 1. Strongly Self-testable Solution for the Synthesized Data Path.

Module	TPG for left input port	TPG for right input port	TRC
M1	R1	R2	R8
M2	R8 through M3	R7	R1
M3	R6	R1	R8
M4	R6	R7	R7

3.4 Interconnection Assignment

For a given module assignment, different register assignments have different effects on interconnection area. The minimum interconnection assignment can be modeled as a double clique partitioning of the input *register compatibility graph* (RCG) [14]. In RCG, each input register is represented as a vertex, an edge between two vertices if the two input registers can be connected to the same input port (of a module). Weights are used in the clique partitioning algorithm to direct the assignment towards finding cliques, such that registers with high reusabilities are connected to both input ports (of the same module).

4 Experimental Results

In order to examine the efficiency of the proposed methodology, we generated minimal area BIST solution for several high-level synthesis benchmarks. Here, we present results on high-level synthesis, and RTL modification for strongly self-testability.

We choose three popular academic high-level synthesis benchmarks data paths:

(1) the 2nd order differential equation *DiffEq* [15];
(2) the auto regression filter element *AR_Filter* [16];
(3) the 5th order elliptic wave filter *FIR_filter* [17];

Table 2. Our Experimental Results (with strong BIST)

	Modules			Registers				Multiplexers					BIST overhead
	+	-	*	CBILBO	BILBO	TPG	TRC	2:1	3:1	4:1	5:1	6:1	
DiffEq	1	1	3	0	1	3	2	9	3	0	0	0	18.67%
FIR_filter	4	0	4	1	2	4	1	7	2	0	2	0	27.38%
AR_filter	4	0	8	0	3	5	2	14	5	3	3	0	7.11%

Table 3. Traditional High-Level Synthesis Results (without BIST consideration)

	Modules			Registers				Multiplexers					BIST overhead
	+	-	*	CBILBO	BILBO	TPG	TRC	2:1	3:1	4:1	5:1	6:1	
DiffEq	1	1	3	1	1	2	1	6	4	0	0	0	29.25%
FIR_filter	4	0	4	2	4	1	0	6	1	0	1	0	46.06%
AR_filter	4	0	8	0	7	8	1	11	2	2	0	3	17.66%

In Table 2, we report the results we obtained for strongly self-testable data paths. For each benchmark data path (given in scheduled DFG and a module assignment), we report the number of registers, including the number of CBILBO, the number of BILBO, the number of TPG, the number of TRC. The number of multiplexers, and the extra area overhead due to the configuration for the synthesized strongly self-testable data path. The BIST area overhead is expressed as a percentage increase in the gate count as a result of the modification for the BIST scheme. For the purpose of comparison, we list in Table 3 traditional high-level synthesis results [18] on the same benchmarks without any BIST consideration.

5 Conclusions and Ongoing Work

In this paper, we first introduced strong self-testability for data paths at RTL. The key aspect is to propose a high-level synthesis scheme for producing such strongly self-testable data paths. This is achieved by incorporating testability constraints during register assignment. This method is based on the use of test resources reusability to improve the self-testability of a data path. Given a scheduled data flow graph and a module assignment, we assign variables to registers, such that the area overhead required for a strongly self-testable data path is minimal. Experimental results are presented to demonstrate the effectiveness of the proposed approach.

The advantages of the proposed method are high fault coverage for single stuck-at faults, low extra hardware overhead and capability of at-speed testing. As part of our ongoing work, the proposed method will be tested on more academic benchmarks as well as real industrial ones.

Acknowledgements

Authors would like to thank Mr. Hiroki Wada at Nara Institute of Science and Technology for his constructive discussion.

References

[1] H.Wada, T.Masuzawa, K.K.Saluja and H.Fujiwara, "Design for Strong Testability of RTL Data Paths to Provide Complete Fault Efficiency", *Proc. of IEEE 2000 Int'l Conf. on VLSI Design (VLSI Design'2000)*, pp.300-305

[2] M.Inoue, K.Noda, T.Higashimura, T.Masuzawa and H.Fujiwara, "High-Level Synthesis for Weakly Testable Data Paths", *IEICE Transactions on Information and Systems*, Vol.E81-D.No.7, 1998, pp.645-653

[3] S.Ravi, N.K.Jha and G.Lakshminarayana, "TAO-BIST: A Framework for Testability Analysis and Optimization of RTL Circuits for BIST", *Proc. of IEEE 1999 VLSI Test Symposium (VTS'99)*, pp.398-406

[4] L.J.Avra, "Allocation and Assignment in High-Level Synthesis for Self-Testable Data Paths", *Proc. of IEEE 1991 Int'l Test Conf. (ITC'91)*, pp.463-472

[5] C.Papachristou, S.Chiu and H.Harmanani, "SYNTEST: a method for high-level SYNthesis with self-TESTability", *Proc. of IEEE 1991 Int'l Conf. on Computer Design (ICCD'91)*, pp.458-462

[6] I.Parulkar, S.Gupta and M.A.Breuer, "Data Path Allocation for Synthesizing RTL Designs with Low BIST Area Overhead", *Proc. of ACM/IEEE 1995 Design Automation Conf. (DAC'95)*, pp.395-401

[7] X.Li and P.Y.S.Cheung, "Data Path Synthesis for BIST with Low Area Overhead", *Proc. of IEEE 1999 Asian and South Pacific Design Automation Conference (ASP-DAC'99)*, pp.275-278

[8] N.Mukherjee, M.Kassab, J.Rajski and J.Tyszer, "Arithmetic Built-In Self-Test for High-Level Synthesis", *Proc. of IEEE 1995 VLSI Test Symposium (VTS'95)*, pp.132-139

[9] I.G.Harris and A.Orailoglu, "SYNCBIST: SYNthesis for Concurrent Built-In Self-Testability", *Proc. of IEEE 1994 Int'l Conf. on Computer Design (ICCD'94)*, pp.101-104

[10] M.A.Abadir and M.A.Breuer, "A Knowledge-Based System for Designing Testable VLSI Chips", *IEEE Design and Test of Computers*, 1985, pp.56-68

[11] X.Li, T.Masuzawa and H.Fujiwara, "An Approach to Designing Strongly Self-Testable Data Paths at RTL", *Technical Report*, 2000

[12] D.L.Springer and D.E.Thomas, "Exploiting the Special Structure of Conflict and Compatibility Graphs in High-Level Synthesis", *Proc. of IEEE 1990 Int'l Conf. on Computer-Aided Design (ICCAD'90)*, pp.254-257

[13] M.C.Golumbic, Algorithmic Graph Theory and Perfect Graphs, *Academic Press*, 1980.

[14] B.M.Pangrle, "On the Complexity of Connectivity Binding", *IEEE Trans. on CAD*, Vol.10, 1991, pp.1460-1465

[15] P.Paulin and J. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASICs", *IEEE Trans. on CAD.*, Vol.8, No.6, 1989, pp.661-679

[16] N.Park and A.C.Parker, "SEHWA: A program for Synthesis of Pipelines", *Proc. of ACM/IEEE 1986 Design Automation Conf. (DAC'86)*, pp.454-460

[17] R.Jain, A.C.Parker and N.Park, "Predicting System-Level Area and Delay for Pipelined and Non-pipelined Designs", *IEEE Trans. on CAD*, Vol.11, No.8, 1992, pp.955-965

[18] I.Parulkar, S.Gupta and M.Breuer, "Scheduling Data Flow Graphics for Minimizing BIST Resources in Data Paths", *Proc. of IEEE 1998 Design, Automation and Test in Europe (DATE-98).*