Data Path Synthesis for BIST with Low Area Overhead *

Xiaowei Li and Paul Y.S.Cheung
Department of Electrical and Electronic Engineering
The University of Hong Kong
Pokfulam Road, Hong Kong

Abstract

This paper presents an attempt towards design quality improvement by incorporating of self-testability features during dada path (high-level) synthesis. This method is based on the use of test resource sharing possibilities to improve the self-testability of the circuit. This is achieved by incorporating testability constraints during register assignment. Experimental results are presented to demonstrate the effectiveness of the proposed data path synthesis for BIST approach.

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 [3]. 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 *pseudo-random pattern generator* (PRPG) for test vector generation and a *multiple-input shift register* (MISR) for response compression. The blocks which are required to perform a test (i.e., PRPG and MISR) are known as *test resource*. 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.

A major consideration in using BIST is the area overhead due to the modification of normal registers to be test registers. How to reduce the BIST area overhead without sacrificing the quality of the test is an important research problem. One of the difficulties in using BIST techniques is the *register self-adjacency* problem. A self-adjacent register cannot be configured as both a PRPG and a MISR simultaneously, unless it is implemented as a *concurrent BILBO* (CBILBO) [9] which can simultaneously perform both the PRPG and MISR operations because it has two sets of bistables. A CBILBO register is approximately 1.75 times the size of a BILBO register [1] and induces more delay during normal operation mode.

To deal with the problem of register self-adjacency, methods have been proposed either to avoid producing such self-adjacent registers or to minimize the number of self-adjacent registers during (high-level) synthesis process [1,4,6,8]. Papachristou *et* al. [6] first presented a combined register and ALU allocation method that generates self-testable designs that do not have any self-loops. The approach is based on constraining the allocation to generate a *testable functional block* (TFB). Its drawback is the inability to map operations whose variables life spans overlap to the same TFB, thus, the final design may use more TFBs than necessary. Avra [1] proposed a register allocation method that minimizes the number of self-adjacent registers in the design. The assumption in her work is that every self-adjacent register needs to be modified to be a CBILBO register, and thus the area overhead is high. Parulkar *et* al. [8] attempts to employ the concept of I-path to reduce the area overhead imposed by BILBO registers. Since the I-path does not alter the test data transferred along it, only the registers at the head and the tail of the I-path are considered to convert to BILBO registers.

In this paper, we explore test resource sharing and its relationship with register self-adjacency and BIST area overhead. Study shows that register self-adjacency does not necessarily imply poor testability. One key aspect of our research is to use two testability constraints to guide register assignment process which will result in a minimal area BIST solution. Another key aspect of our research is to use a graph model (as a static analysis tool) to solve area overhead minimization problem in BIST modification. This method was applied to several benchmarks resulting in high self-testability (with low area overhead) than the original design (without testability).

* This project is supported in part by the *Croucher* Foundation Grant #360/062/0994.

2 Test Resource Sharing and Register Self-adjacency

Consider the scheduled data flow graph (*DFG*) shown in figure 1. A minimum of 3 registers are required. There are 108 distinct assignments of the variables to 3 registers [8]. With respect to register and functional unit area, these 108 assignments are equivalent. Only a subset of these result in more self-testable data paths (with lower BIST area overhead) than the rest. One possible RTL implementation is shown in figure 2. As a BIST solution, it can be seen that R_1 and R_2 can be shared as PRPGs between adder (M_1) and multiplier (M_2), and R_3 can be shared as MISR for testing the adder and the multiplier in turn.

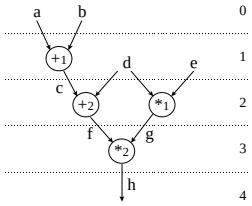

Figure 1. A Scheduled DFG.

Definition 1: A register R_i is said to be a *driver* of a module M if some outputs of R_i are inputs to M. A register R_j is said to be a *receiver* of M if some outputs of M are inputs to R_j. R_i is said to be *adjacent* to R_j if there exists a module M such that R_i is a driver of M and R_j is a receiver of M. If R_i is both a receiver and a driver of M, then it is *self-adjacent*.

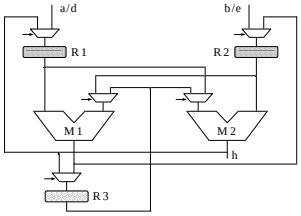

Figure 2. Data Path from the DFG in Figure 1.

A directed graph which named *register adjacency graph* (RAG) has been adopted to visualize register self-adjacency. Each node in the RAG represents a module or a register. A directed edge exists from register node R_i to a module node M_k if R_i is a driver of M_k, and a directed edge exists from M_k to node R_j if R_j is a receiver of M_k.

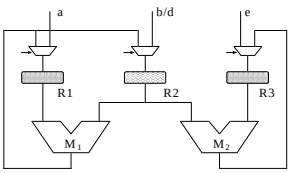

Figure 3. Data Path from the DFG in Figure 1.

Take scheduled DFG in figure 1 for example, one possible data path without regard for testability is shown in figure 3. It's corresponding RAG is shown in figure 4. A minimal area BIST solution for this data path is R_1 as a PRPG, and R_2 and R_3 as CBILBOs which is more costly than the minimal area BIST solution for the earlier design (in figure 2). It can be seen that R_1, R_2 and R_3 (in both figure 2 and figure 3) are self-adjacent registers, but their BIST solutions are totally different. In this paper, we broadly divide self-adjacent registers into two sorts:

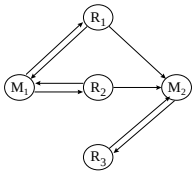

Figure 4. RAG of the Data Path in Figure 3.

Definition 2: A self-adjacent register is referred to as a *typical* self-adjacent, if and only if its topological structure in RAG is a exact self-loop.

Definition 3: A self-adjacent register is referred to as a *non-typical* self-adjacent, if its topological structure in RAG contains a self-loop.

Take RAG in figure 4 for example, R_3 is a typical self-adjacent register, while R_1 and R_2 are non-typical self-adjacent registers.

Based on the above definitions, a condition for a register to be a CBILBO is derived as follows:

A register needs to be modified as a CBILBO register if and only if it is a typical self-adjacent register. Any non-typical self-adjacent register can be modified as PRPG, MISR, BILBO, or CBILBO, depends on the results of test resource sharing (discuss in next session).

3 Data Path Behavioral Synthesis for BIST

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

The register assignment problem can be modeled as coloring the *register conflict graph* (RCG) [4]. Each node in the RCG represents an edge from the DFG that crosses a clock cycle boundary. A *conflict* edge between two nodes in the RCG 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. A coloring of RCG corresponds to a valid register assignment with each color corresponding to a register. Figure 5 shows the RCG for the scheduled DFG in figure 1.

In order to avoid creating any self-adjacent register, a testability conflict edge (*dashed line*) is added between two nodes when one node represents a variable that is an input to a function module and the other node represents a variable that is an output of the same function module. Testability conflict edges require that the inputs and the outputs of a function module be assigned to different registers, and this guarantees that no self-adjacent register will be synthesized.

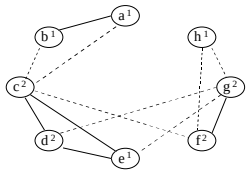

Figure 5. RCG of the Scheduled DFG in Figure 1.

Given the RCG, several algorithms exist that can almost always color the graph with a minimum number of colors in polynomial time [5], where the number of colors is the number of registers required in the data path design. However, a *k*-colorable graph, where *k* is the chromatic number of the graph, may have several different *k*-colorings, each coloring representing a different assignment of nodes to registers. These colorings may represent data path designs of different sizes due to different number of multiplexers and interconnects. Therefore, techniques must be employed to guide the graph coloring algorithm to the lowest-cost implementation.

Definition 4: The *sharing effectiveness* SE(*v*) of a variable *v* is the sum of the number of modules for which *v* is a driver and the number of modules for which *v* is a receiver.

We associate a sharing effectiveness SE(*v*) with each variable *v* (node) in the RCG, as shown in figure 5. The sharing effectiveness SE(*v*) reflects the number of modules for which the register *v* can act as PRPG and the number of modules for which it can act as MISR. Using this measure, the assignment process can be guided by choosing merges that result in large increases in the sharing effectiveness of registers.

Consider the scheduled DFG shown in figure 4 and the following module assignment. Operations $+_1$ and $+_2$ are assigned to module M_1 and operations $*_1$ and $*_2$ are assigned to module M_2. ({*a,d,h*}, {*b,e,f*}, {*c,g*}) is a possible register assignment.

Figure 2 shows the data path corresponding to this register assignment and the given module assignment. It can be seen that R_1 and R_2 can be shared as PRPGs between M_1 and M_2, and R_3 can be shared as MISR for testing M_1 and M_2 in turn. Since minimal area overhead is our objective, it is not necessary to test all the combinational modules at that same time, i.e., in one test session.

4 Data Path Structural Synthesis for BIST

The area overhead minimization problem (in the BIST context) is to decide which registers to modify and the modes (PRPG, MISR, BILBO or CBILBO) 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 PRPG or MISR and denote these roles by the letters *p* and *m*, 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_{im}. 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 *m*.

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

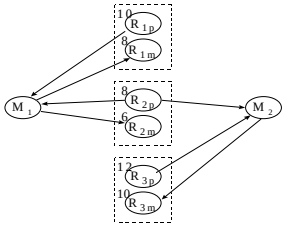

Figure 6. STG of the Data Path in Figure 3.

We are now ready to model the information to be used for BIST area overhead minimization, in the form of a directed graph which we call the *self-testing graph* (STG). Each node in the STG represents a register instance, each node R_{ix} is labeled by the corresponding modification cost C_{ix}. Directed edges are from register instance R_{ip} to module and from the module to register instance R_{im}. A module can be self-testable if all of its driver R_i are converted to R_{ip} and all of its receiver R_j are converted to R_{jm}. R_{ip} and R_{jm} are called a *test solution* (TS) of the module. Figure 6 shows the testing graph for the data path in figure 3.

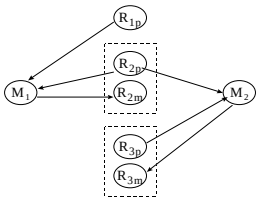

Figure 7. An Optimal Subgraph of STG in Figure 6.

The area overhead minimization problem is deciding which register to modify and the modes (PRPG, or MISR)

to add, in such a way that every module in the data path can be tested with minimal area overhead. In other words, the problem is to find a minimal cost proper subgraph covering the set of all module's TS. For example, in the testing graph of figure 6, we want to find a proper subgraph covering TSs of both adder (M_1) and multiplier (M_2) with the minimal modification cost.

Figure 7 shows such an optimal subgraph. R_1 is modified as PRPG, while both R_2 and R_3 needs to be modified as both PRPG and MISR (simultaneously), therefore, CBILBO is needed. The significance of an optimal subgraph is that the registers that must be modified for minimal area overhead are those whose instances belong to the optimal subgraph. This graph optimization problem can be formulated as an integer programming problem and solved by existing efficient algorithm [5].

5 Experimental Study

Figure 8 depicts a schematic view of the proposed (streamlined) data path BIST synthesis methodology.

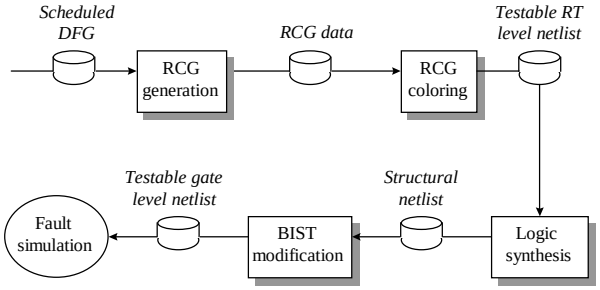

Figure 8. High-level BIST Synthesis Methodology.

The data path synthesis for BIST methodology, takes as input a scheduled DFG of the system logic, generates an area-optimized self-testable data path logic with few self-adjacent register. Two testability constraints are imposed on the RCG, which guarantee to generate a potential self-testable data path. Then, we apply a structural BIST modification on the synthesized data path to get a BIST solution with a minimal (extra) area overhead.

Table 1. Experimental Results on Academic Benchmarks.

DFG	DiffEq	AR_filter
Module Assignment	1+, 4*, 1-	4+, 8*
# Register	7	16
# Multiplexer	26	48
# CBILBO	1	0
# BILBO	2	7
# PRPG	2	8
# MISR	2	1

We present our results on two well-known high-level synthesis benchmarks (the 2nd order differential equation *DiffEq* [7] and the auto regression filter element *AR-Filter* [2]) in table 1. The above results are comparable with those published in literature.

6 Concluding Remarks

In this paper, we presented techniques to exploit test resource sharing to minimize the area overhead due to BIST modification in both structural and behavioral synthesis domains. Two testability constraints have been adopted to guide register assignment process which will result in a minimal area BIST solution. A graph model is proposed to solve area overhead minimization problem in (structural) BIST modification. By using modification cost associated with each register, an area-optimized self-testable design can be generated. Experimental results on academic benchmark examples demonstrate the ability of the proposed approach to generate self-testable data path with low area overhead.

As part of our ongoing work, the proposed data path synthesis for BIST methodology will be tested on more academic benchmarks as well as real industrial one.

References

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

[2] 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

[3] Mike T.C.Lee, High-Level Test Synthesis of Digital VLSI Crcuits, *Artech House Inc*., ©1997

[4] X.Li and P.Y.S.Cheung, "High-Level Synthesis for At-Speed Self-Test", *Proc. of Int'l Conf. on Computer-Aided Design and Computer Graphics*, 1997, pp.466-470

[5] M.W.Padberg and M.Rijal, Location, Scheduling, Design and Integer Programming, *Kluwer Academic Publishers*, ©1996

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

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

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

[9] L.T.Wang and E.J. McCluskey, "Concurrent Built-In Logic Block Observer (CBILBO)", *Proc. of IEEE Int'l Symp. on Circuits and Systems*, 1986, pp.1054-1057