ESTIMATING TEST COST DURING DATA PATH AND CONTROLLER SYNTHESIS WITH LOW POWER OVERHEAD

Haidar M. Harmanani and Maya Kodeih

Department of Computer Science and Mathematics
Lebanese American University
Byblos, 1401 2010, Lebanon

ABSTRACT

This paper presents a method for concurrent BIST cost estimation during testable data path allocation. The method integrates testability in the design process and generates a distributed test controller that aims to minimize area and power. The system has been implemented and favorable results are reported.

1. INTRODUCTION

The testability of a circuit depends largely on its interconnect structure as well as on the functions of its components. It has been generally recognized that in order to have a good design quality, there must be a tight coordination between the design and test processes. Given a behavioral circuit description, there exist different implementations with various structures. The testability cost may increase or decrease accordingly. Based on this observation, this work incorporates test constraints *during* the design process by tightly integrating the design and test processes. The problem we address in this paper is described as follows:

> Given a behavioral level description of a circuit represented in the form of a scheduled DFG and a set of constraints, generate a *self-testable RTL data path structure* such that: 1) the datapath conforms to all the user constraints; 2) the overhead of test registers in the data path is minimized; 3) the power consumption is minimized during normal and test execution.

The method uses a hierarchical controller approach where each test kernel has its own test controller in addition to a small controller that synchronizes the distributed controllers. The resulting distributed controllers are small and mounted next to their corresponding kernels. The power dissipation will be distributed among the different sub-controllers. By distributing the control, useless power dissipation can be eliminated by selectively stopping the clock portions of the controllers that are inactive at the specific clock cycle.

1.1. Related Work

Several early approaches have been proposed for high-level synthesis with test [1, 2, 3]. Kim *et al.* [4] proposed a BIST synthesis approach based on ILP that performs testable synthesis in addition to test scheduling. Zwolinski *et al.* [5] proposed a BIST method with design and time exploration. Cheng *et al.* [6] proposed a register allocation algorithm that improves the testability for high-level circuit synthesis. Tosun *et al.* [7] proposed an integer linear programming method for reliability-oriented high-level synthesis that addresses the soft error problem. Safaria *et al.* [8] presented a register allocation method based on weighted graph coloring algorithm. Sun *et al.* [9] proposed a method for register allocation for testability based on a weighted compatibility clique partition algorithm.

Various techniques for power reduction in datapath synthesis for test, and controller optimization have been proposed. For example, Crews *et al.* [10] and Hertwig *et al.* [11] discuss techniques for high-performance controller synthesis. Benini *et al.* [12] describe transformation for incompletely specified state machines in order to automatically synthesize low power finite state machines with gated clocks while Favalli *et al.* [13] methodologies that guarantee testability for gated-clock FSM. In [14], Nourani *et al.* describe the design of an interacting datapaths and controller.

The remainder of this paper is organized as follows. In section 2 we describe the test methodology along with the allocation model, the test points selection method, and the test trade-offs scheme. Section 3 describes the controller synthesis approach in addition to the power minimization approach. Finally, results are presented and discussed in section 4.

2. TESTABLE DATA PATH SYNTHESIS

Given a scheduled data flow graph (DFG), a DFG node corresponds to 1) an operator that must be assigned to a functional unit during the control step in which it is scheduled; 2) a value that must be assigned to a register for the duration of its life time. Data transfers are assigned to some path of connections, buses and multiplexers.

The allocation model is based on the notion of *structural testability* introduced in [2]. The key element of the structural testability model is the *Testable Functional Block* (TFBs), shown in Fugure 1, which are test kernels that do not have any self-loops. In this model, behavioral operations are mapped onto the ALU of the TFB, while the behavioral variables are mapped to the register at the TFB output. For example, in to be able to test the circuit Figure 1(a), we allocate the registers at the input ports as TPGRs. Test patterns are collected and observed at the output port register, configured as an MISR.

The testable allocation algorithm is an iterative refinement procedure. The method constructs an initial datapath structure (IDP) that corresponds to an initial design point through the mapping of DFG nodes onto individual TFBs. The initial design cost is incrementally improved through the merging of TFBs guided by the cost difference of the current and the intended data path configuration.

2.1. Concurrent BIST Points Selection

In order to be able to explore tradeoffs during the synthesis process, we use a methodology that takes BIST cost into consideration. Thus,

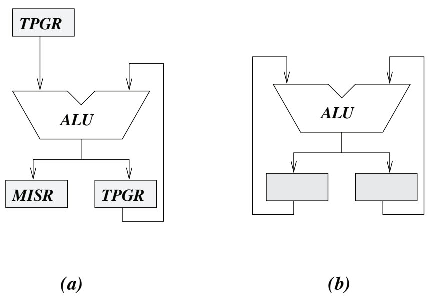

Fig. 1. (a) Testable ALU with self-adjacent register, (b) Non-Testable ALU with self-adjacent registers

the method cab determine the exact effect of the datapath merging process. We use the testability tradeoff model described in [15]. The model states that 1) whether the input patterns applied to this module are random enough and 2) whether the fault effects of this module can be sensitized through intermediate modules to an observable point. Only if these conditions cannot be satisfied, an observable or a controllable point needs to be inserted.

2.1.1. *Data Path Representation*

The data path consists of modules, registers as well as the connections among these entities. The usage or attribute of each test register can be *controllable, observable, pseudo-controllable,* or *pseudo-observable.* The final implementation of a register is the union of the underlying register usage. For example if a register is a controllable point for one module and an observable point for another module, then this register will have a final implementation attribute as a BILBO. This implementation can indicate the hardware overhead before and after merging of data flow nodes by computing the difference in cost over two test plans. During optimization, the pseudo merge of two TFBs is the combination of the given two TFBs into one and the linkage of their associated test plans into a preliminary *merged test plan.*

2.1.2. *Selection of Input Registers*

In order to test the datapath, we need to select a *pattern generator* for each module's input port resulting from merging the two given TFBs. Thus, for every CLB input port in the data path we select one register to provide random patterns during testing and remove the input port from all the *controllable* usage entry of the other input registers. The selection priority goes from *normal*, *observable*, *controllable*, to *controllable & observable.* It should be noted that if the current TFB is transparent such that the testing time is acceptable after the adjustment, then the output attribute is changed from *observable* to *pseudo-observable* for all input registers.

2.1.3. *Selection of Output Registers*

The selection of a signature analyzer avoids nodes merger when the resulting output register supplies test patterns to different input ports of the same module. The following cases apply:

1. *A BIST register combined with a BIST register:* If the merged test registers are controllable and observable pointa respectively, then the identifier list is updated from *controllable* to *pseudo-controllable,* and from *observable* to *pseudo-observable.* Assign implementation attribute to *normal* when the upper bound of test time is not exceeded. If, however, either register is attributed with both a controllable and an observable point, then we assign the implementation attribute *controllable & observable* to the output register.

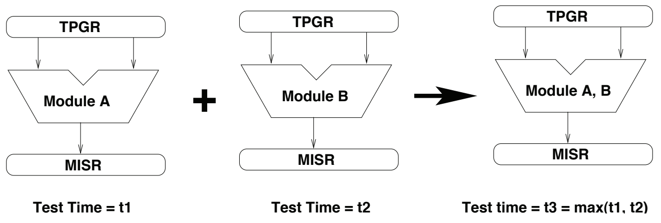

Fig. 2. Combining two TFBs

2. *A non-BIST register combined with a non-BIST register:* The resulting register in this case (Figure 2) is assigned the *normal* attribute except in the case where t_3 exceeds user constraints. In this case, a *controllable* attribute is assigned if the violation is due to pattern randomness. Otherwise, the attribute is assigned the *observable* attribute (transparency problem).
3. *Non-BIST registers combined with BIST registers:* If the BIST register has a *controllable* attribute then the attribute is removed if the test time is not exceeded. In this case, all attributes are updated from *controllable* to *pseudo-controllable.* Since now half of the time the patterns are random, it is sufficient to use twice the number of previous test pattern generation time in order to the modules that feed from this module. The same case applies when the register attribute is *observable.* If the time limit is exceeded then a BIST attribute is assigned to register after merging.

2.1.4. *Self-Loop Removal*

The selection of input and output test points may create self-loops due to the functional removal of test points. A self-loop problem arises when there exists a path $F \rightarrow C$ and/or $C \rightarrow F$ where F and C are test registers (Figure 3). There are two cases: 1) Test plan T_1 connects to test plan T_2, and the path F $\rightarrow$ C pass through the boundary and 2) Path F $\rightarrow$ C is in the same test plan. In what follows, we describe the loop breaking algorithm with reference to Figure 3.

Both TFBs are in different test plans

Consider the case F $\rightarrow$ C; if F and C are in different test plans and at least one of register does not have a *normal* attribute, there is no self-loop between F and C. Otherwise:

1. Find all the children of F and store in a queue I.
2. $\forall A \in I$:
   (a) If $A = C$, then there is a cycle; add to C an implementation attribute of either *controllable* or *observable.* Update the attribute table. Stop.
   (b) If A has already been visited then go to 2.
   (c) If A has an implementation attribute of either *controllable* or *observable* but is not an interface node then mark A visited and go to 3.
   (d) Add all A's unvisited children to queue I, and mark A *visited.*

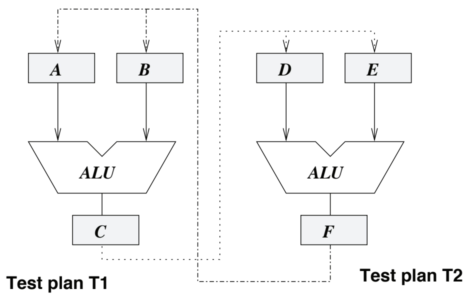

Fig. 3. Paths that may cause functional self-loops

Both TFBs are in the same test plan

Consider the case F $\rightarrow$ C. If F and C are in the same test plan then if the union of both implementation attributes is:

1. *Normal:* Find a controllable and an observable point.
2. *Controllable* (*observable*): Find an *observable* (*controllable*) point.
3. *Controllable & observable*: Similar to case 2.

Next, the loop breaking process proceeds as follows:

1. Start from F, and store its children in queue I. Assign each node in I a label that is the same as its implementation attribute.
2. Dequeue node A, and mark it as *visited.* Repeat until I is empty.
   (a) If the label set union of the node A and F is 1) *controllable, observable* or 2) *controllable & observable*, *observable*, then there is no self-loop in this path. Go to 2.
   (b) If any child of A is C, then there is a self-loop. Record the current node label and identifier that caused the self-loop.
   (c) Otherwise, assign the label set union of F and of the current node to the *current label.* Assign all unvisited children of the current node the *current label set,* and enqueue in I.
3. Break-Loop as follows:
   (a) Let Start be the union of the attributes of registers F and C.
   (b) If Start is *normal* and the union of all the labels recorded is (i) *normal,* then assign all the recorded nodes *controllable* and the node F *observable*; (ii) *controllable,* then assign all the recorded nodes that have attribute *normal* with attribute *controllable* and the node F *observable*; (iii) *observable,* then assign all the recorded nodes that have attribute normal with attribute *observable* and the node F *controllable*; (iv) *controllable & observable,* then assign all the recorded nodes that have attribute *normal* with attribute *observable,* and the node F *controllable and observable.* Once the implementation attribute has been added, modify the attribute table accordingly.
   (c) If Start is *controllable* then add to the recorded registers the attribute *observable;* modify the implementation attribute and usage attribute accordingly.
   (d) if Starts is *observable* then add to the recorded registers the attribute *controllable,* and modify the implementation attribute and usage attribute accordingly.
   (e) If Start is *controllable & observable* then either F or C is *controllable & observable* while the other one is *normal.* In this case, add to all the recorded registers *observable* attribute and modify the implementation attribute and usage attribute accordingly.

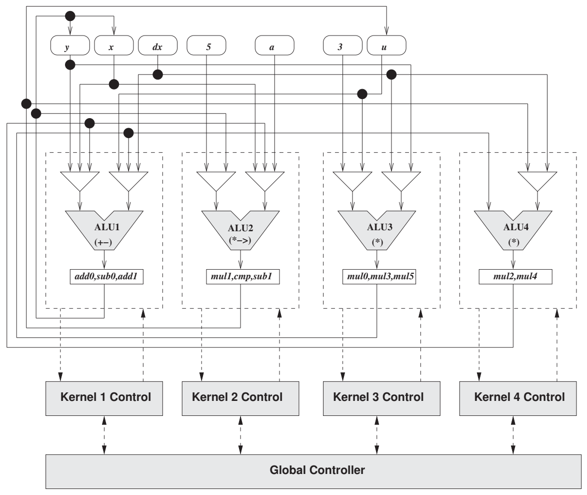

Fig. 4. Datapath with Distributed Control

3. DISTRIBUTED CONTROLLER SYNTHESIS

Distributed controllers tend to consume more power than centralized controllers. However, the key is to ensure that distributed controllers are not concurrently active as total parallelism is not possible to achieve.

The distributed controller scheme is based on a hierarchical finite state machine (FSM) model where every test kernel has its own *local* controller. Each state corresponds to a clock cycle and includes all necessary signals needed for the proper execution of operations in the kernel *only*, including the selection of the multiplexers inputs and register modes. The next states are determined based on the next activation clock cycle in the kernel, as implied by the schedule. This may include loops and branches. Depending on the test schedule, the sub-controller may not necessarily have actions to execute in every clock cycle. A relatively larger controller will coordinate the execution of the sub-controllers (Figure 4).

During distributed control execution, there are registers that maybe shared among one or more test kernel. These registers could be either *controllable* or *observable*. In order to better quantify the effect of power consumption in both models, we have implemented both mechanism as follows:

1. *Option 1:* Kernels include muxes, ALU's and output registers with all I/O belonging to the main controller.
2. *Option 2:* Registers are assigned to a kernel based on their test function.

Circuit	*Type*	**Normal**	**TPGR**	**MISR**	*BILBO*	**Mux In**	**Area**	**Ovhd%**
Elliptic 17	Normal	3120	0	0	0	1664	4784	
	BILBO	1248	768	0	2328	1664	6008	20.37
	Concurrent	1456	768	608	776	1664	5480	12.70
Elliptic 21	Normal	2288	0	0	0	1612	3900	
	BILBO	832	768	0	1552	1612	4584	14.92
	Concurrent	1248	512	608	388	1612	4368	10.71
DCT 4	Normal	2704	0	0	0	1148	3852	
	BILBO	0	2560	0	1940	1088	5588	31.07
	Concurrent	1664	1536	304	0	1248	4752	18.94
Wavelet6	Normal	2912	0	0	0	1312	4224	
	BILBO	0	2816	0	1940	1328	6084	30.57
	Concurrent	2496	1024	304	0	1112	4936	14.42
IIR3	Normal	3120	0	0	0	1020	4140	
	BILBO	0	2816	0	1552	864	5232	20.87
	Concurrent	1664	1280	608	0	992	4544	8.89
FIR6	Normal	2912	0	0	0	768	3680	
	BILBO	0	2816	0	1552	752	5120	28.13
	Concurrent	1040	2048	608	0	752	4448	17.27

Table 1. Results Summary from the High-Level Synthesis Benchmarks

The first option delegates the registers control to the global controller, while the second option, although results in a larger kernel size, reduces the size of the global controller, since fewer registers are controlled directly. The system generates both types of kernels for comparison reasons.

The global controller controls the distributed sub-controllers via control signals. The sub-controllers act as decoders that are triggered by the main controller at the corresponding clock cycle. For each sub-controller, we construct a state or decoding table. An optimization step is done to remove duplicate control signals that target the same registers form and to different sub_controllers.

During test mode, the controller requires an additional external signal to distinguish between test and system mode. Each test session consists of three phases: 1) initialization step, where the test pattern generators are initialized with a value different from zero and the compressors are set to a defined state; 2) test execution, and 3) shifting out the corresponding signatures. A counter is initialized with the corresponding test length before each test session. The counter's output is connected with the global controller. The controller also specifies various constants such as the number of test sessions, the scanpath length, the number of registers between a test pattern generator and a signature analyzer. In the scan path all registers are included, which act as a compressor in a test session. After each test session the signatures are shifted out.

3.1. Power Reduction

BIST circuits result in considerably higher circuit activity rate compared to normal mode operation. This is due mostly to the higher switching activity of a digital circuit and is, in general, the major source of power consumption [16]. Thus, the difference between BIST and normal modes is in the activity rate, which is, for instance, assumed 0.5 in the case of pseudo-random based BIST schemes, and often less for normal mode operation [13]. In order to obtain a realistic power dissipation for a random logic block, a predetermined percentage (ex. 10%) needs to be added to the block size N, corresponding to the average area overhead for the selected BIST scheme.

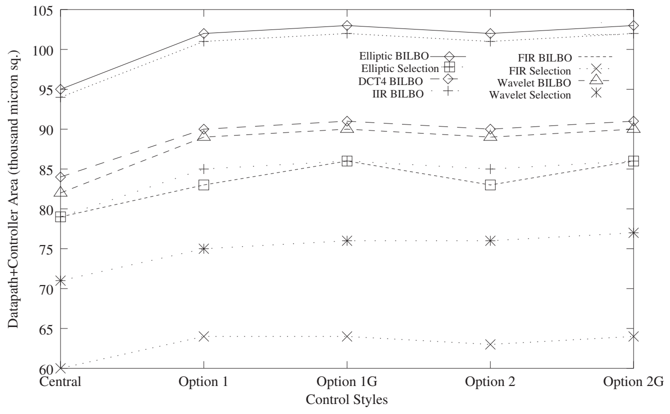

Fig. 5. Datapaths Area with Distributed Control

To solve the above problem, we use *RTL clock gating* at the distributed controllers level. Thus, according to the test schedule, we can take advantage of the fact that not all test kernels are active during the same clock. This means that the non-active test kernels are shutdown using gating. Thus, by grounding their clock during test mode where the signal has the information indicating when the kernel under test is active, we minimize switching activity and hence reduce power consumption. RTL clock gating has many advantages. First, they are easy to implement and maintain, technology independent, and requires no RTL code change. Furthermore, they deliver significant power savings due to a reduced internal power consumption in gated register, reduced capacitance (switching power) on clock network, and minimizes area.

4. RESULTS

We implemented the described method and measured the performance of our BIST system for various data flow graphs including the

Table 2. The dynamic power measured using Synopsys simulation for a 4-bit bus

Benchmark Example	*Mode*	*# Central*	*# Option1*	*# Option1G*	*# Option2*	*Option2G*
Elliptic	Norm	22.4693 uW	16.5983 uW	12.7921 uW	16.4339uW	8.2887 uW
BILBO	Test	8.7646 uW	25.9482 uW	7.9905 uW	80.3593 uW	7.1345uW
Elliptic	Norm	21.6776 uW	15.5717 uW	11.0283 uW	15.4314 uW	7.3950 uW
Selection	Test	8.5584 uW	24.5886 uW	7.6763 uW	79.7385 uW	6.8329uW
FIR	Norm	71.9638 uW	74.6128 uW	66.8155 uW	74.6128 uW	66.8155 uW
BILBO	Test	64.1252 uW	123.3939 uW	62.1604 uW	123.3939 uW	62.1604 uW
FIR	Norm	41.4004 uW	35.1743 uW	30.0457 uW	35.1743 uW	30.0456 uW
Selection	Test	28.9279 uW	63.7957 uW	26.7423 uW	63.7957 uW	27.7330 uW
Wavelet	Norm	72.1938 uW	39.4201 uW	16.6070 uW	39.3697 uW	16.1519 uW
BILBO	Test	21.2511 uW	107.0060 uW	11.0410 uW	106.9427 uW	10.6488 uW
Wavelet	Norm	42.3162 uW	38.9226 uW	12.4490 uW	39.1628 uW	16.1519 uW
Selection	Test	8.7174 uW	35.1173 uW	8.7757uW	106.9427uW	10.6488 uW
DCT4	Norm	28.5887 uW	28.7692 uW	19.6521 uW	27.9099 uW	14.6900 uW
BILBO	Test	18.4989 uW	89.1389 uW	9.3732 uW	88.9121 uW	8.8147uW
IIR	Norm	93.4799 uW	83.5396 uW	18.2383 uW	83.3994 uW	16.6751 uW
BILBO	Test	19.1457 uW	85.0118 uW	10.7404 uW	81.2435 uW	8.0898 uW

fifth order elliptical wave filter, *the* 6^{th} *order finite impulse response filter (FIR)*, *the* 3^{rd} *order infinite impulse response filter (IIR)*, *the 4-point discrete cosine transform circuit (DCT)*, and *the 6-tap wavelet filter*. The implementation details are shown in Table 1. For every example, we show the circuit area based on the transistor count based on [17] and we show the overhead of the generated designs. The datapath logic of a circuit is not considered in the transistor count. The number of transistors in test registers and multiplexers is based on the circuits in [17, 18]. As shown in Table 1, our system successfully synthesized two different style data paths; a design style that is based on BILBO and another one that is based on concurrent selection of test points. The area overhead ranges from 14.92% to 20.37% in the case of BILBO style and from 5.24% to 12.70% in the case of concurrent test point selection.

We compare next our implementation using the *Synopsys* synthesis tools. The implementation details by our synthesis system are shown in Figure 5 and in Table 2. As shown, *Option 2* in the distributed control mechanism results in the best improvement in power dissipation with this option providing the most parallel solution. The improvement in power consumption is on the average 54.22% in normal mode while, as expected, the average power reduction decreases to 28.56% in the test mode due to the high switching activity. In specific, the improvement ranges from 7.15% in normal mode in the case of FIR BILBO to 82.16% in the case of the IIR FILBO. During testing mode, the improvement ranges from 3.5% in the case FIR BILBO to FIR BILBO to 57.74%. The benchmarks design area, based on Synopsys, are shown in Figure 5.

5. REFERENCES

[1] L. Avra, "Allocation and Assignment in High-Level Synthesis for Self-Testable Data Paths," *Proc. ITC*, 1991.

[2] H. Harmanani, C. Papachristou, "An Improved Method for RTL Synthesis with Testability Tradeoffs," *ICCAD*, 1993.

[3] K. Olcoz, F. Tirado, H. Mecha, "Unified Data Path Allocation and BIST Intrusion," *Integration, the VLSI Journal*, 1999.

[4] H. Kim, D. Ha, T. Takahashi, T. Yamaguchi, "A New Approach to Built-In Self-Testable Datapath Synthesis Based on ILP," *IEEE Trans. on VLSI*, 2000.

[5] M. Zwolinski, M. S. Gaur, "Integrating Testability with Design Space Exploration," *Microelectronics Reliability*, 2003.

[6] B. Cheng, H. Wang, S. Yang, D. Niua, Y. Jin, "Register Allocation Algorithm for High-Level Circuit Synthesis for Improved Testability," *Tsinghua Science & Technology*, 2008.

[7] S. Tosun, O. Ozturk, N. Mansouri, E. Arvas, M. Kandemir, Y. Xie, W.-L. Hung, "An ILP Formulation for Reliability Oriented High-Level Synthesis," *ISQED*, 2005.

[8] S. Safaria, A. H. Jahangira, H. Esmaeilzadehb, "A Parameterized Graph-Based Framework for High-Level Test Synthesis," *Integration, the VLSI Journal*, 2006.

[9] Q. Sun, T. Zhou, H. Lia, "A Novel Register Allocation Algorithm for Testability," *Tsinghua Science & Technology*, 2007.

[10] A. Crews, F. Brewer, "Controller Optimization for Protocol Intensive Applications," in *Proc. EURO-DAC*, 1996.

[11] A. Hertwig, H. Wunderlich, "Fast Controllers for Data Dominated Applications," *Proc. ED&TC*, 1997.

[12] L. Benini, G. De Micheli, "Automatic Synthesis of Low-Power Gated-Clock FSM," *IEEE Trans. CAD*, 1996.

[13] M. Favalli, L. Benini, G. De Micheli, "Design for Testability of Gated-Clock FSMs," *Proc. ED&TC*, 1996.

[14] M. Nourani, J. Carletta, C. Papachristou, "Synthesis for Testability of Controller-Datapath Pairs that Use Gated Clocks," *Proc. DAC*, 2000.

[15] S. Chiu, C. A. Papachristou, "A Design for Testability Scheme with Applications to Data Path Synthesis," *Proc. DAC*, 1991.

[16] Y. Zorian, "A Distributed BIST Control Scheme for Complex VLSI Devices," *Proc. VTS*, 1993.

[17] B. Konemann, J. Mucha, G. Zwiehoff, "Built-In Logic Block Observation Techniques," *Proc. ITC*, 1979.

[18] L.T. Wang, E.J. McCluskey, "Concurrent Built-In Logic Block Observer," *Proc. ISCAS*, 1986.