High-level Variable Selection for Partial-Scan Implementation

Frank F. Hsu Janak H. Patel
Center for Reliable & High-Performance Computing
University of Illinois, Urbana, IL

Abstract

In this paper, we propose a high-level variable selection for partial-scan approach to improve the testability of digital systems. The testability of a design is evaluated at the high level based on previously proposed controllability and observability measures. A testability grading technique is utilized to measure the relative testability improvement in a design as the result of making a subset of the variables fully controllable and observable. The variables that cause the greatest testability improvement are selected, and the selection process is performed incrementally until no further testability improvement can be achieved. Then the registers that correspond to the selected variables are placed in the scan-chain for partial-scan implementation. The experimental results shows that the variable selection approach produces partial-scan implementations that can achieve high fault coverage, while the logic overheads are fairly low.

1 Introduction

Conventional partial-scan selection techniques perform the flip-flop selection at the gate level after the logic structures have been realized. Although high fault coverages are often achieved, the information from the higher levels of abstraction, such as the register-transfer level or behavioral level, is not applied to further improve the quality of the flip-flops selected for partial scan. The lack of high-level information also causes the flip-flops to be selected from dispersed locations in a circuit. Although the partial-scan implementations may prove their effectiveness during testing, they have little value for designers during design debug and fault diagnosis.

Partial-scan approaches that use high-level information to improve the quality of scan element selection were proposed in [1, 2, 3, 4, 5, 6].

In [1], the authors presented a resource sharing technique to generate circuits with minimal partial-scan requirements. The algorithm first breaks all loops formed in the control-data flow graph with the minimal number of scan flip-flops; then scheduling and allocation are performed without introducing any loops in the datapath. The testability of hard-to-test registers in the circuit can be improved by sharing the same set of scan registers.

A behavioral synthesis-for-testability technique was presented in [2]. This technique generates easily testable nonscan or partial-scan implementations of a design by considering testability issues during resource allocation and register binding. Another behavioral synthesis approach was described in [3]. The authors defined a set of test environments to represent the testability of modules in a design. Given the scheduled data-flow graph of a design, the resource allocation algorithm is applied to generate an easily testable circuit.

A resource allocation algorithm was proposed in [4]. The algorithm first analyzes the design at the high level to identify loops such as functional loops, topological loops, single assignment loops, and multiple assignment loops. Then the allocation algorithm is applied to produce an acyclic structure with minimal area and partial-scan overhead.

The authors of [5] proposed a behavioral transformation technique to minimize partial-scan requirements in synthesized circuits. By analyzing the testability cost, scheduling difficulty, and transformation difficulty during high-level synthesis, designers can generate an easily testable circuit that requires less area and scan cost compared to the circuit synthesized from the original description. The same authors also presented a high-level partial-scan approach in [6]. In this approach, hardware sharing is utilized to break data-dependency loops, assignment loops, sequential false loops, and register file cliques. By sharing scan registers among several variables, the total partial-scan cost can be minimized.

The contribution of this study is to develop a partial-scan selection technique based on high-level information of a design. While most high-level techniques emphasize on the testability analysis of the datapath of a system, our technique focuses on both control flow and data flow. The testability analysis is per-

This research was supported in part by DARPA under Contract DABT63-95-C-0069, in part by the Semiconductor Research Corporation under Contract SRC 97-DJ-482, and by Hewlett-Packard under an equipment grant.

formed on the control-data flow graph of a design based on a set of controllability and observability metrics. A testability grade is also computed for the design. The variables defined at the high level are chosen for scan based on their ability to improve the testability of the system. The variable selection is performed incrementally until no further testability improvements can be made. A similar testability-based partial-scan selection approach was described in [7]. In that approach, testability measures are developed to guide the selection process based on the testability impact of each flip-flop. Instead of selecting individual flip-flops, our approach analyzes a design at the high level and selects the variables that contribute the most to improving the testability of a design.

After the desired variables are chosen, the registers that correspond to these variables are selected for partial-scan implementation. The output of our technique is a testability-and-cost graph that estimates the amount of partial-scan requirements needed to achieve high testability. The graph also allows designers to control the trade-offs between scan cost and testability improvement. The proposed approach is independent of the synthesis procedure and the test generation algorithm as long as there exists a mapping between the gate-level registers and variables defined in the high-level description.

We will describe the high-level testability analysis method and the testability grading technique in Section 2. The proposed variable selection algorithm will be discussed in Section 3, followed by experimental results in Section 4. Section 5 contains the conclusion of this study.

2 Testability Analysis

Testability metrics were proposed in [8] to evaluate the testability of a design at the high-level. The control-data flow graph (CDFG) is first extracted from the high-level description of a design. Then the graph is analyzed based on the controllability and observability metrics. The goal of controllability and observability analysis is to estimate the testability of branch conditions, functional units, and variables defined in the design. Then the result of the testability analysis can be summarized to form a testability grade for the system. In this work, a node within a CDFG is the **locus of execution** for the system if it is currently being executed by the system. The testability definitions are summarized as follows.

DEFINITION C-1: *An assignment node in the CDFG has controllability cost* **C** *if all of its operands can be controlled by the primary inputs* **C** *clock cycles before the* **locus** *reaches the node, where* **C** *is a specific integer.*

DEFINITION C-2: *An assignment node in the CDFG is* **noncontrollable** *if one or more of its operands cannot be controlled by the primary inputs within any predetermined number of clock cycles prior to the* **locus** *reaching the node. This node has controllability cost* **C** *equal to infinity.*

DEFINITION C-3: *A branch of a decision node has controllability cost* **C** *if the outcome of the branch condition can be controlled to take that branch by the primary inputs* **C** *clock cycles before the* **locus** *reaches the node, where* **C** *is a specific integer.*

DEFINITION C-4: *A branch of a decision node is* **noncontrollable** *if the outcome of the branch condition cannot be controlled to take that branch by the primary inputs within any predetermined number of clock cycles prior to the* **locus** *reaching the node. This branch has controllability cost* **C** *equal to infinity.*

DEFINITION O-1: *An assignment node in the CDFG has observability cost* **O** *if the result of the operation can be observed at the primary outputs after* **O** *clock cycles, where* **O** *is a specific integer.*

DEFINITION O-2: *An assignment node in the CDFG is* **nonobservable** *if the result of the operation cannot be observed at the primary outputs within any predetermined number of clock cycles. This node has observability cost* **O** *equal to infinity.*

In our experiments, the *infinity* value is typically set to a large integer. It should be noted that the testability measures defined above are used for guidance, and therefore the accuracy of **C** and **O** is not extremely important. When high-level synthesis is used, the clock cycle boundaries may not be known exactly before synthesis. Therefore, one needs to approximate the above measures. One approximation is to count the number of register transfer (RT) statements in place of *clock cycles*. Basically, the controllability cost estimates the level of difficulty in controlling a node from the primary inputs, and the observability cost approximates the level of difficulty in propagating the value of a node to the primary outputs.

2.1 Controllability analysis

Before starting the computation of controllability measures, all noninput variables have controllability costs equal to *infinity*, and all primary inputs and constant values have controllability cost equal to zero. For an assignment node in the CDFG, the destination variable can be controlled within one cycle after the operands are controlled. The controllability cost represents the number of RT statements required to control a node. Therefore, the destination variable and the functional unit denoted by the RT statement have controllability cost equal to the maximum controllability cost of the operands plus one. The formula to compute controllability cost for the destination variable is

$$C(destination) = MAX\ [\ C(operands)\ ] + 1$$

During the controllability evaluation of the CDFG, the controllability cost of a variable is recorded every time it is computed. The number of different controllability costs for each variable is equal to the number of RT statements in which it is the designated destination variable. The combined controllability value for a variable is calculated by taking the average of all the controllability costs recorded for the variable. This average controllability value represents the ability of the primary inputs to control the variable.

2.2 Observability analysis

Initially, all nonoutput variables have observability costs equal to *infinity*, and all primary outputs have observability costs equal to zero. The algorithm starts with the last node in the CDFG. All functional units that assign their results to a primary output have observability costs equal to zero, because these results are immediately observable. The computation of observability cost of an arbitrary functional unit (RT statement) is more complicated, because the propagation of a signal at the input of a functional unit depends on the controllability of other inputs to the same unit, i.e., a hard-to-control input may cause difficulties in the propagation of values at the other inputs. The formula for computing the observability cost for each input operand of a functional unit becomes

$$O(op_i) = MAX \begin{cases} O(destination) + 1 \\ MAX[C(op_{j \neq i})] + 1 \end{cases}$$

During the observability evaluation of the CDFG, the observability cost of a variable is recorded every time it is computed. The number of different observability costs for each variable is equal to the number of RT statements in which the variable occurs as an operand variable. The combined observability value for a variable is calculated by taking the average of all the observability costs recorded for the variable. This average observability value represents the difficulty in propagating the content of the variable to the primary outputs.

2.3 Testability Grading

After a high-level testability analysis, the computed measures include the controllability of branch/loop conditions, the controllability and observability of assignment nodes, and the controllability and observability of variables and signals, as shown at the top of Figure 1. The testability values of various portions of a design can be combined according to their type and weight, as shown in Table 1. The testability values from various components can be combined by computing the *weighted-average* of their testability values.

First, the controllability value of every branch from a decision node can be combined; each branch has a weight equal to the number of nodes following each branch. The combination of the testability of all decision nodes provides the testability estimate for the control flow of the system.

Table 1: Testability and weight of system components.

Type	T.	Wt.
Branch condition	C	# of nodes in branch
Assignment node	C/O	# of operations
Variable (V)	C/O	Log_2 (# of bits)
Signal (S)	C/O	Log_2 (# of bits)
Control logic	C	# of decision nodes
Datapath	C/O	# of assignment nodes
Func. network (F)	C/O	# nodes in CDFG
Data network (D)	C/O	wt(V) + wt(S)
Entire system	C/O	wt(F) + wt(D)

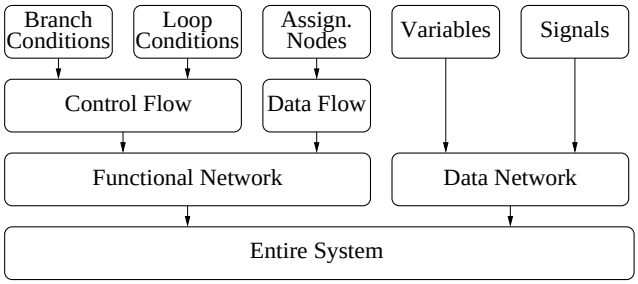

Figure 1: Testability grading hierarchy.

The combined testability value of an assignment node is computed by taking the average of its controllability and observability values. Then the combined testability of all assignment nodes is computed by taking the *weighted-average* of the testability value from each assignment node. The resultant testability represents the testability of the data flow of the system. Together with the testability of the control flow, we have an estimate of the testability of the entire CDFG, which is also referred to as the functional network in Figure 1.

The testability of each variable/signal is calculated by combining the testability values computed during testability analysis. The weight of each variable equals Log_2 of the number of bits defined for the variable, with 1 as the minimum weight. This is done to avoid placing too much weight on large variables such as memory arrays. The controllability and observability values are first combined for each variable/signal; then, the testability values of all variables/signals are combined to form a single value representing the testability of the underlying data and interconnect framework. This combined testability value provides a testability estimate of the data network.

Finally, the testability of the functional network and the data network can be combined to generate a quan-

titative measure of system testability. Notice that this single measure can range from zero (easily testable) to *infinity* (not testable), assuming *infinity* is a large integer. The following formula can be utilized to transform the testability value into a testability-grade (TG) number between zero and 100, which is a notation commonly used. Similar to the *fault coverage* value representing the level of testability for a circuit, the testability grade represents the system testability at the high level.

$$TG_{(0\leftrightarrow 100)} = \frac{|inf| - Testability_{(0\leftrightarrow inf)}}{|inf|} * 100$$

3 Variable Selection Procedure

Conventional partial-scan selection approaches perform the flip-flop selection at the gate-level, after the structure of the circuit is known. Our approach utilizes high-level information to guide the scan selection process in order to generate the optimal partial-scan implementation. Gate-level techniques tend to select individual flip-flops that meets certain testability criteria, while our approach requires all flip-flops that correspond to the selected variables to be scanned.

The selection algorithm starts by obtaining the register mapping information from the gate-level registers to the variables defined at the high level. During high-level synthesis, registers may be allocated for some variables, while the other variables are treated as temporary signals, and no registers are allocated. Before variable selection is performed, all registered variables are recognized and placed on the *candidates list* for selection. After the register mapping is obtained, variables that are costly to scan and variables that will not contribute to testability improvement are eliminated from the selection process; large arrays and output buffers are removed from the *candidates list.*

3.1 Selection Algorithm

The selection procedure requires three major components: the candidate list, the solution list, and the testability grading engine. The candidate list contains all registered variables with the exception of large arrays and output buffers. The solution list is the record of partial-scan solutions developed during the iterative selection process. The testability grading engine is the tool developed to perform testability grading using the proposed controllability and observability measures. In this work, a *solution* is defined as a combination of variables selected for partial-scan implementation, and each solution has its testability grade and associated scan cost (bit-width). The solutions are ranked first by increasing testability and second by decreasing scan cost, because a good partial-scan solution should achieve high testability while requiring a minimal number of scan flip-flops. A *seed* is the highest ranking solution in the solution list.

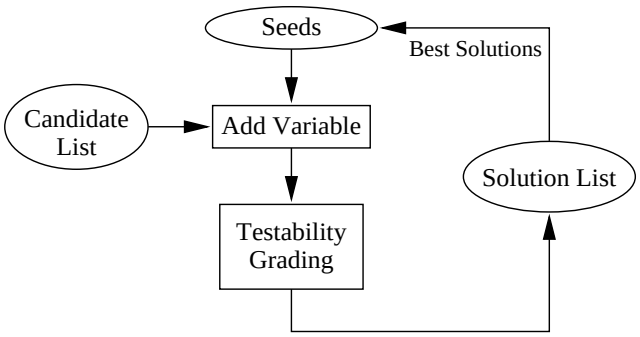

Figure 2: Variable selection algorithm.

The first step of the selection process is to evaluate the testability impact of each candidate variable. Testability grading is performed individually assuming each variable is fully controllable and observable; this is to simulate the effects of scanning the corresponding registers. The result is a list of single-variable solutions, each with a testability grade and scan cost. The best solutions are stored in the solution list for the next iteration. In our experiments, only the five best solutions are stored on the solution list.

The next step is an iterative process starting from the existing single-variable solutions. Instead of the exhaustive approach that is of $O(2^N)$ complexity, where N is the number of variables, we have developed an $O(N^2)$ greedy algorithm to select the best combinations of variables for partial-scan. The algorithm starts by using the five best solutions from the solution list as seeds. For each seed, variables not already present in the seed are added individually to the seed to form new solutions. For each of these newly formed solutions, testability grading is performed on the design, treating the selected variables as fully controllable and observable elements. Then the testability grade and the scan cost are stored with the solution, and the solution is inserted into the solution list according to its testability grade and cost.

At the end of an iteration, a brief examination will reveal whether there are testability improvements among the best solutions in the solution list. If the maximum testability grade have not increased in the current iteration, it means the optimal testability level has been achieved, and the procedure terminates. Otherwise, more iterations are required until no further testability improvement can be achieved. The variable selection algorithm is also presented pictorially in Figure 2.

3.2 Barcode Example

The output of the variable selection process is a testability-and-cost graph, similar to the one shown in Figure 3. The original order listed in the solution list is displayed at the bottom of the figure, with version 1 being the best solution suggested by the selec-

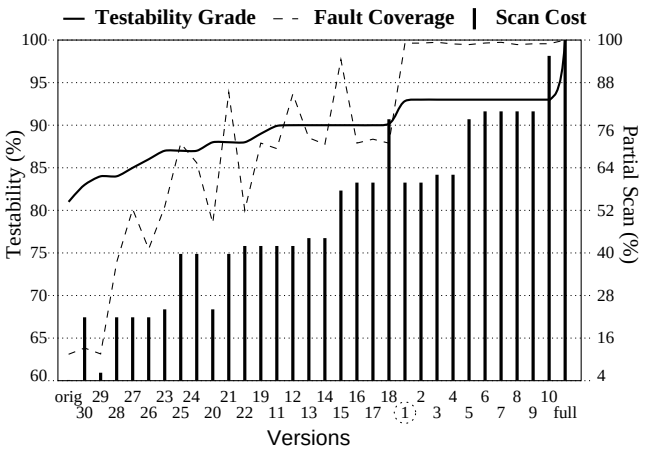

Figure 3: Testability and cost graphs for Barcode.

tion procedure. The second best solution is labeled 2, etc. In the figure, the data points are first ordered by their testability grade, then by their scan cost. Actual ATPG was performed for every partial-scan implementation of *Barcode* to construct the fault coverage curve in the figure.

Each flat segment on the testability grade curve represents a group of solutions having similar testability while each solution scans different subsets of registers. Since the proposed variable selection approach is an estimation technique, it does not predict the exact fault coverage of the circuits. However, the increasing trend displayed by the fault coverage curve demonstrates good correlation between the testability grades and the actual fault coverages.

Notice the plateau region formed by both the testability grade and the fault coverage curve at the right-hand side of the figure. This occurs when the selection algorithm cannot develop better solutions with higher testability grades. One reason is that no further testability improvements can be achieved even if more variables are selected for scan. The second reason is that further testability improvement is very costly. While the partial-scan implementation for several solutions can achieve very high fault coverage, the least costly solution is preferred. For the Barcode design, *Solution* 1 requires only 58% of flip-flops to be scanned to achieve maximum testability.

4 Experimental Results

Experiments were conducted to demonstrate the effectiveness of the proposed partial-scan selection technique. The gate-level netlists were obtained using Design Compiler [10]. Each synthesized circuit contains the entire system, including the control logic and the datapath. Automatic test generation was conducted using the tool from Sunrise Test Systems [11].

To compare the performance of different partial-scan selection techniques, three partial-scan circuits were implemented for each design. The first version scans the flip-flops suggested by *OPUS* [9] to break all cycles in the circuit structure. The second version scans the flip-flops selected by *Autoloop* [11] targeting 95% fault efficiency. The third version scans the registers suggested by the proposed variable selection technique. The time spent for the variable selection process ranges from a few seconds for the smallest design to more than 15 hours for the largest design. *OPUS* and *Autoloop* are gate-level tools while ours is a high-level tool. The full-scan fault coverage is also listed for reference. The results are shown in in Table 2.

For *DHRC*, the variable selection technique selected 26% of the flip-flops and achieved 98.2% fault coverage. Although *OPUS* selected only two flip-flops to achieve 97.5% fault coverage, it is much more difficult and costly to gain additional testability improvement.

Our approach is able to obtain very high fault coverage for *GCD* while the number of flip-flops selected is much less compared to the results of *Autoloop*. Although our approach selects more flip-flops compared to *OPUS*, the difference between the results is two bits in a variable; *OPUS* selects six bits out of eight bits while our approach selects the eight flip-flops corresponding to the variable. Similarly, our technique scans every flip-flops corresponding to the selected variables in *Barcode* and achieves higher fault coverage compared to the circuits generated by the other partial-scan techniques.

Scanning only 40% of flip-flops in *LRU* can improve the fault coverage to 93.4%, higher than the 85.4% resulting from cutting all cycles in the circuit. The fault coverage of circuit implementations suggested by our technique is comparable to the fault coverages produced by *Autoloop*, while fewer flip-flops are selected for partial scan.

In *Kalman*, the variable selection technique is able to achieve 97.3% fault coverage by scanning just 11% of flip-flops, only a fraction of the cost compared to the 94% scan implementation recommended by *Autoloop*. Our approach produces a much more testable partial-scan implementation compared to the design generated using *OPUS*.

In *Prawn*, higher fault coverage can be achieved using fewer scan flip-flops compared to the circuits produced by *Autoloop*. However, our approach selects more flip-flops in *AM2910* to achieve similar testability compared to the other two approaches, the difference being that our approach selects every flip-flops in a variable, while others select only a portion of the registers.

The results for *GL85* and *i8251* are very similar to each other. Our approach selects more flip-flops compared to *OPUS*, but less compared to *Autoloop*, while the fault coverage achieved by variable selection is also between *OPUS* and *Autoloop*.

In general, the circuits produced by the variable

Table 2: Results for various partial-scan implementations.

	Original				OPUS		Autoloop		Var Select			Full
Circuit	**Prim.**	**Var.**	**DFFs**	**FC (%)**	**Scan (%)**	**FC (%)**	**Scan (%)**	**FC (%)**	**V.S. (%)**	**Scan (%)**	**FC (%)**	**FC (%)**
DHRC	4503	10	123	83.4	2	97.5	90	99.1	50	26	98.2	99.9
GCD	943	3	51	92.5	65	97.2	100	99.7	67	69	99.5	99.7
Barcode	835	8	46	66.1	39	83.8	53	97.5	50	61	99.6	99.8
LRU	1271	7	93	23.7	55	85.4	62	96.9	71	40	93.4	99.9
Kalman	7326	13	540	4.5	5	43.7	94	98.9	31	11	97.3	99.3
Prawn	2214	17	84	77.3	18	95.2	41	98.0	53	38	98.5	99.7
AM2910	1543	9	115	47.5	22	95.7	13	95.6	33	24	95.8	100
GL85	6045	78	280	13.7	30	72.2	82	98.2	35	51	89.5	98.4
i8251	6926	57	289	7.48	20	47.3	100	96.8	77	63	91.0	96.8

Prim.: number of logic primitives Var.: number of registered variables
Scan %: percentage of DFFs scanned FC: fault coverage
V.S. %: percentage of registered variables selected for partial scan

selection technique are able to achieve higher fault coverage compared to circuits produced by *OPUS*, and slightly lower fault coverage compared to circuits generated by *Autoloop*. The proposed partial-scan technique produces circuits with smaller logic overhead compared to circuits generated by *Autoloop*, and slightly higher logic overhead compared to circuits produced by *OPUS*.

5 Conclusions

A variable selection for partial-scan approach is described in this paper. Variables defined at the high level are selected for scan based on their ability to improve the testability of the system. The selection procedure is an iterative process in which variables are selected incrementally until no further testability improvements can be achieved.

The result of the selection process is an estimation of the amount of scan registers required to achieve high fault coverage. The designer may allocate circuit area for scan circuitry using the estimated scan cost early in the design flow. The testability-and-cost graph also allows designers to control the trade-offs between testability and scan cost. Furthermore, by identifying partial-scan solutions at the high level, the synthesis tool can perform logic optimization which can reduce the scan cost in the gate-level implementation.

References

[1] S. Dey, M. Potkonjak, and R. K. Roy, "Exploiting hardware sharing in high-level synthesis for partial scan optimization," in *Proceedings of the IEEE International Conference on Computer-Aided Design,* 1993, pp. 20–25.

[2] T. Lee, N. K. Jha, and W. H. Wolf "Behavioral synthesis of highly testable data paths under the non-scan and partial scan environments," in *Proceedings of the IEEE Design Automation Conference,* 1993, pp. 292–297.

[3] S. Bhatia and N. K. Jha, "Genesis: A behavioral synthesis system for hierarchical testability," in *Proceedings of the IEEE European Design and Test Conference,* 1994, pp. 272–276.

[4] V. Fernandez and P. Sanchez, "Partial scan high-level synthesis," in *Proceedings of the IEEE European Design and Test Conference*, 1996, pp. 481-485.

[5] M. Potkonjak, S. Dey, and R. K. Roy, "Considering testability at behavioral level: use of transformations for partial scan cost minimization under timing and area constraints," *IEEE Transactions on Computer-Aided Design,* vol. 14, no. 5, pp. 531–546, May 1995.

[6] M. Potkonjak, S. Dey, and R. K. Roy, "Behavioral synthesis of area-efficient testable designs using interaction between hardware sharing and partial scan," *IEEE Transactions on Computer-Aided Design,* vol. 14, no. 9, pp. 1141–1154, September 1995.

[7] P. S. Parikh and M. Abramovici, "Testability-based partial scan analysis," *Journal of Electronic Testing: Theory and Applications*, vol. 7, pp. 61–70, August 1995.

[8] F. F. Hsu, E. M. Rudnick, and J. H. Patel, "Enhancing high-level control-flow for improved testability," in *Proceedings of the IEEE International Conference on Computer-Aided Design,* 1996, pp. 322–328.

[9] V. Chickermane and J. H. Patel, "An optimization based approach to the partial scan design problem," in *Proceedings of the International Test Conference,* 1990, pp. 377–386.

[10] *Synopsys Reference Manual: Design Compiler,* Version 1997.01, November 1996.

[11] *Sunrise Tests Systems Reference Manual: Testgen and Autoloop,* Version 2.3b, February 1997.