Reducing Test Point Overhead with Don't-Cares

Kai-Hui Chang
Avery Design Systems, Inc.
Andover, MA, USA
changkh@avery-design.com

Chia-Wei Chang
National Central University
Jhongli, Taiwan
955401007@cc.ncu.edu.tw

Jie-Hong Roland Jiang
National Taiwan University
Taipei, Taiwan
jhjiang@cc.ee.ntu.edu.tw

Chien-Nan Jimmy Liu
National Central University
Jhongli, Taiwan
jimmy@ee.ncu.edu.tw

***Abstract*— Test points provide additional control to design logic and can improve circuit testability. Traditionally, test points are activated by a global test enable signal, and routing the signal to the test points can be costly. To address this problem, we propose a new test point structure that utilizes controllability don't-cares to generate local test point activation signals. To support the structure, we propose new methods for extracting don't-cares from assertions and finite state machines in the design. Our empirical evaluation shows that don't-cares exist in many designs and can be used for reducing test point overhead.**

I. Introduction

Logic Built-In-Self-Test (LBIST) is a commonly-used test method that relies on internally-generated pseudo-random vectors to uncover manufacturing defects in a circuit. Compared with tests with patterns generated by Automatic-Test-Pattern-Generation (ATPG) tools, LBIST requires less interaction with external testing equipment, thus shortening test time and reducing test cost. However, certain faults may be hard-to-detect by LBIST due to the difficulty to assign a specific value to a design node with random patterns. For example, the probability to generate a "1" on the output of a 32-input AND gate using random patterns is less than one in a billion. The term *random resistance* is used to describe this phenomenon. If the resistance of a node is high, then it is difficult to generate the non-prevalent value of the node. One way to address this problem is to insert test points that provide additional control for the high random-resistance nodes. The inserted test point provides a new logic path to change the value of the target node, thus reducing its random resistance.

Given that test points should not affect the logic of the circuit in normal operation, a signal called *test enable* is typically used: test points are activated only if test enable is asserted. Traditionally, this signal is provided as a primary input, which can occupy valuable routing resources due to the long wires to reach the test points. To address this problem, we extend the ideas described by Ren *et al.* [2] and propose a new test point structure based on *local activation signals*. The activation signals are generated from controllability don't-cares in design registers near the test points. This method can considerably reduce the use of routing resources, thus reducing test point overhead. Since the combination of register values can never be generated in functional mode, the circuit's logic remain unchanged during normal operation. During testing, however, such values can be generated because register values are from the scan chain instead of design logic.

The success of our local test point activation structure depends on the number of don't-cares. Our second contribution is two novel methods to extract don't-cares from the design. In the first method, we extract don't-cares from assertions provided with the design. In the second method, we analyze the Register Transfer Level (RTL) version of the design to identify Finite-State Machines (FSMs) — FSMs often have unused state encodings that are don't-cares. If no don't-cares can be found in the vicinity of the inserted test point, we generate a local don't-care by cloning a near-by register. Experimental results show that our don't-care extraction methods are effective in finding don't-cares in real designs.

II. Background

In this section we review existing test point insertion techniques and introduce the concept of don't-cares.

A. Test Point Insertion

To reduce the random resistance of a node, test points can be inserted [3], [4]. A test point example is shown in Figure 1. In the example, we assume the right-most node is 0-resistant. The inserted test point reduces 0-resistance of the node by ANDing the node with another signal, called a *random activation signal*, that has a high probability to be 0. To ensure that the function of the circuit remains unchanged after test point insertion, an active-low test enable signal is used. When the enable signal is 1, the output of the OR gate is always 1, which preserves the function of the logic.

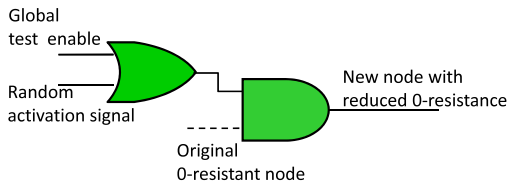

Fig. 1. Test point insertion example. The inserted test point reduces the 0-resistance of the node.

Selecting the location for test point insertion have been studied extensively, such as the work by Savaria *et al.*, Touba *et al.*, and Ren *et al.* [2], [3], [4]. In this work we assume that the location for test point insertion has been determined, and our goal is to reduce the test point overhead. One research to reduce such overhead is by Yang *et al.* Instead of using dedicated registers for random activation, they reuse functional flip-flops to reduce area overhead. Touba *et al.* ANDed signals from primary inputs to drive control points and reduce area overhead. However, routing global signals to the test points can still be costly. The methods proposed by Ren *et al.* [2] implicitly implied that test enable signals can be derived

from design logic and unreachable states can be used to generate the random activation signals. Their experimental results showed that test points using internally-generated control signals provide lower area and timing overhead than traditional methods. However, they did not describe how they identified the unreachable states, which is addressed in this work.

B. Don't-Cares in Logic Synthesis

Don't-cares can be classified into two major types: controllability and observability. Given a set of nodes, controllability don't-cares are created by the fan-in logic of the nodes that prevents a set of values from being created, and observability don't-cares are caused by the fan-out cone of the nodes that prevents different sets of values from creating different values at the outputs. Controllability don't-cares are used in our work to generate local test point activation signals.

III. Test Point Insertion

In this section we first provide an overview of our test point insertion methodology. We then propose a new test point structure based on controllability don't-cares. If no don't-cares exist in the vicinity of the target test point, we provide a register cloning method.

A. Our Methodology

An overview of our methodology is shown in Figure 2. After the user identifies the locations for test point insertion, we perform don't-care analysis to find registers that can generate local random activation signals. If such resisters are found, we perform test point insertion using those registers. Otherwise, we use register cloning to generate local don't-cares.

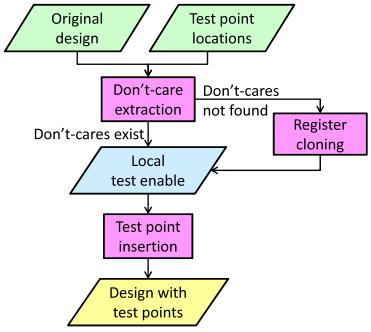

Fig. 2. Our test point insertion flow.

B. Test Point with Local Random Activation

Our test point structure is shown in Figure 3. The test enable signal is generated based on register values that can never appear in the circuit's functional mode, and it also serves as the random activation signal (called a *local random activation* signal in this paper). This signal can be activated in test mode because specific values to enable the signal can be shifted in. In functional mode, however, the signal can not be activated because design logic cannot generate such values.

For example, suppose that a 3-bit variable stores values using one-hot encoding in the functional mode, then ANDing at least any of its two bits will always produce a zero in functional mode. However, if all three bits are scannable, we

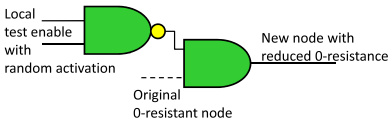

Fig. 3. Our test point structure. Our random activation signal (output of the NAND gate) is generated locally and can never be 1 in functional mode because the signal sources are controllability don't-cares. It can only be 1 in test mode when register values are scanned-in.

can scan-in two ones into the registers, which can produce a one on the output of the AND gate in test mode. We use the output of the AND gate as our local random activation signal to enable the test point.

The advantage of this structure is that a global test enable signal is no longer necessary, which can save considerable routing resources. The reduction in loads at the global test enable signal can also alleviate timing issues when at-speed testing also needs to be performed. In addition, our local random activation signal combines both test enable and random activation signals in the traditional structure, which eliminates the need to find additional registers to serve as the random activation signal. The disadvantage is that controllability don't-cares may not exist in the vicinity of the targeted test point location. Therefore, such structure may not always be applicable. Another limitation is that to generate a don't-care, more than one register typically needs to be involved, thus reducing the chance for the test point to assume the non-prevalent value. For example, if a 4-bit state variable only has 15 states, the probability for the unused state to activate the random activation signal under LBIST is only 1/16, which is smaller than the traditional structure (typically close to 1/2). On the other hand, as we will show in our experimental results, controllability don't-cares involving fewer than 5 registers can be easily found in many designs. Given that thousands of patterns are typically simulated in BIST, the local random activation signal can be turned on hundreds of times, which can already reduce random resistance considerably.

C. Register Cloning

If don't-cares cannot be found in the design or the registers that can generate local random activation signals are distant from the test point, we apply register cloning by duplicating a register near the test point and then make both registers scannable. Local test activation can then be produced by XORing the two registers.

IV. Don't-Care Extraction

In this section we first describe how don't-cares can be identified from assertions provided with the design that are proven to hold. Next, we propose a new algorithm for recognizing FSMs in a design and then extract don't-cares based on unused state encoding.

A. Design Assertion Analysis

To utilize assertions for extracting don't-cares for local random activation generation, the assertions must be combinational. The algorithm shown in Figure 4 can then be used to extract the don't-care values for generating local random

activation signals. The inputs to the algorithm are the assertion and the registers whose don't-cares need to be identified, and the output is the registers values that are don't-cares.

```
function cdc_enum(assertion, regs)
1   inst ← build_cnf_instance(assertion);
2   while (sol ← sat_solve(inst, regs) exists)
3     cdc_values ← cdc_values ∪ sol;
4     inst ← inst ∧ ¬sol;
5   return cdc_values;
```

Fig. 4. Algorithm for enumerating don't-care values from an assertion.

In the algorithm, we first build an instance in the Conjunctive Normal Form (CNF) from the assertions. We then use a SAT solver to solve for values for the given registers that violate the assertion in line 2. If such values exist, they are added to *cdc_values* as controllability don't-cares — the values violate the assertion and thus can never appear in the design. The negation of the values are then added to the CNF instance, *inst*, so that the same solution will not be returned again. This process repeats until no further solutions can be found. At this point, *cdc_values* contains the register values that can be used to generate local random activation signals.

To increase the probability for activating the test point, it is desirable to reduce the number of registers involved when generating the local random activation signal. To achieve this goal, one can build a truth table based on the returned values and then perform two-level logic optimizations. Prime implicants that contain the smallest number of registers can then be used to construct the local random activation signal.

If assertions are not provided with a design, assertion synthesis techniques can be applied to extract design assertions automatically, such as the work by Chung *et al.* [1].

B. Finite State Machine Analysis

In this subsection we propose a technique to quickly find controllability don't-cares from the RTL code of a design. We focus on extracting don't-cares from FSMs because (1) FSMs have regular coding styles, making them easy to to recognize; and (2) usually not all state encodings are used by an FSM, creating don't-cares from the unused encodings. To extract don't-cares from FSMs, we propose a new algorithm shown in Figure 5. The input to the algorithm is the design, and the outputs are the recognized FSMs as well as their states. In this algorithm we use symbolic simulation. However, other Boolean reasoning techniques can also be used.

```
function fsm_recog(design)
1   inject a symbol to each design variable;
2   perform symbolic simulation for one cycle;
3   foreach var ∈ design.variables
4     (ctrl, data)← split_strace_input(var);
5     if (check_const(data) && check_self(var, ctrl))
6       fsms ← fsms ∪ var;
7       var.state ← data;
8   return fsms;
```

Fig. 5. Algorithm for recognizing FSMs and their states.

In the algorithm, we first inject a symbol into each design variable that are storage devices and then perform symbolic simulation for one cycle. The purpose of this step is to create Boolean functions from the logic among design variables. In line 3, for each design variable, we analyze its symbolic trace and split the inputs of the trace based on whether they are on the control or data paths. Variable *ctrl* saves the inputs on the control paths, and *data* saves those on the data paths.

In our algorithm, a data path is the path that assigns a value to the variable, and a control path is the path that selects which value to assign. Take the following code for example:

```
assign a = b ? c : 1;
```

Both "c" and "1" are inputs on the data path because their values can propagate to "a". On the other hand, "b" is on the control path because its value never propagates to "a". It selects whether "c" or "1" should be propagated, though.

In line 5 of the algorithm we check whether all inputs on the data path are constant and the inputs on the control path involve the variable (*var*) itself. If they do, then we flag the variable as an FSM and add it to *fsms* in line 6. The reason is that for an FSM, the transitions of states should involve the current state in at least some states, and the next state should always be a deterministic state that is a constant value. We then assign the constant values in *data* to *var.state* as its states in line 7. Line 8 then return the FSMs in *fsms*.

After obtaining the FSMs and their states, we compare the number of bits (n) that an FSM uses with its number of states (m). If $2^n > m$, then some encodings of those n bits are not used and are don't-cares.

To generate a local random activation signal, we first build a truth table that contains all the unused encodings. We then perform two-level Boolean minimization and select a prime implicant with the smallest number of registers. The signal can then be generated based on the prime implicant.

V. Experimental Results

The work by Ren *et al.* [2] provided comprehensive evaluations on how the use of don't-cares can reduce test point insertion overhead. Due to the lack of access to physical design and testing tools, we could not provide similar evaluations. On the other hand, given that our methods are enhancements upon Ren's work, we focus on evaluating how well our methods can identify unreachable states in a design instead. The benchmarks used in our experiments are shown in Table I. Design DLX and Alpha are from the Bug UnderGround project at Michigan [6], designs AES to WB_conmax are from OpenCores [8], and the rest of the designs are from our industrial partners. To show the approximate sizes of the designs, we synthesized the public-domain designs and reported their cell counts. We were not able to synthesize the industrial designs, though, due to their use of proprietary memory models and blocks. Therefore, we report the number of lines of their RTL source code instead.

To evaluate the efficiency and effectiveness of our FSM analysis method discussed in Section IV-B, we applied the

TABLE I

CHARACTERISTICS OF BENCHMARKS.

Design	#Cells	Description
DLX	6075	MIPS-lite 5-stage CPU
Alpha	14353	Alpha-lite CPU
AES_core	10657	5AES encoder
CPU8080	1420	8080 compatible CPU
TV80	1010	Z-80 compatible CPU
USB_funct	7688	USB function IP core
WB_conmax	17781	Wishbone interconnect matrix
Design	#Lines	Description
Industry1	38826	Block of a PCI-Express chip
Industry2	176205	Block of a PCI-Express chip
Industry3	37265	Cache controller
Industry4	108721	Communication chip

algorithm to the benchmarks shown in Figure I, and the results are shown in Table II. In the table, "#FSM" is the number of FSMs identified by our algorithm, "#FSM (with red.)" is the number of FSMs with redundant (unused) state encodings, and "FSM size hist." is the histogram of FSM sizes (number of bits in the FSM) with unused state encoding.

TABLE II

FSM ANALYSIS RESULTS.

Design	Runtime (sec)	#FSM	#FSM (with red.)	FSM size hist. ($\leq 5, \leq 10, > 10$)
DLX	1	No FSM found		
Alpha	1	No FSM found		
AES_core	1	No FSM found		
CPU8080	4	1	1	0,1,0
TV80	2	1	0	0,0,0
USB_funct	1	4	4	2,1,1
WB_conmax	101	80	0	0,0,0
Industry1	11	16	16	7,9,0
Industry2	269	237	161	161,0,0
Industry3	930	8	5	5,0,0
Industry4	153	34	31	31,0,0

From the results, we observe that our runtime was short for most designs. Even for large industrial blocks, we could still finish the analysis in 1000 seconds. The results show that no FSMs were identified in DLX and Alpha. This is because these two designs were straight-forward pipelined processors and no FSMs were used. AES_core also did not have any FSM because it is an arithmetic core. The rest of the designs all have FSMs in them, and most of the FSMs have unused state encoding that can be used for generating local random activation signals, especially those in industrial designs. This result shows that our test point structure can be useful for real designs. One exception is WB_conmax, where all its FSMs use full state encodings. We inspected those FSMs and found that they are not real FSMs but are storage elements for the arbitration algorithm that happen to match our template. Since the arbiter must accept data from all inputs, all "states" are used in those elements and there are no unused encodings.

The histogram on the sizes of the FSMs with unused encodings shows that most FSMs consist of five or fewer bits, suggesting that local random activation signals generated using those FSMs can be activated at least 1/32 of the time, sufficient to reduce random resistance of the test point target.

To utilize assertion-based don't-care extraction techniques discussed in Section IV-A, assertions must be provided. Given that our benchmarks do not include assertions that we can utilize, we implemented the assertion-extraction techniques described in [1] to mine properties in DLX and Alpha. To serve our purpose, we only generate combinational properties between two registers that match the "implication" template. For DLX, runtime was 10s and no assertions were found. For Alpha, runtime was 214s and 11 assertions were found. We inspected the assertions and found that they were control registers that record similar information for different purposes. For example, one assertion says "*mem_wb_take_branch*" implies "*id_ex_cond_branch*" (if write-back stage needs to take branch then the execution condition also needs to take branch), and another one says "$\overline{id\_ex\_illegal}$" implies "*if_valid_inst_reg*" (if the instruction at decoder is not illegal then it is valid at the output of the fetch stage). The local random activation signal for an implication assertion, "*a* implies *b*", can then be generated by "$a\&\overline{b}$".

VI. CONCLUSION

Test point insertion is a commonly-used method for improving design testability. Since test points should not affect the circuit's functionality, they are typically controlled by a global test enable signal and are activated only when the signal is asserted. Routing the signal from a primary input to the test points, however, can consume valuable routing resources. In this paper we proposed a new test point structure that utilizes controllability don't-cares to generate its control signals. By generating the signals locally and using them for random activation as well, test point insertion overhead can be reduced. To support the new structure, we proposed two new algorithms for extracting controllability don't-cares from the design: one identifies don't-cares from design assertions and the other one utilizes unused state encodings in design FSMs. Our experimental results show that our techniques can efficiently and effectively extract don't-cares from many designs, which allows the generation of local random activation signals.

REFERENCES

[1] C.-N. Chung, C.-W. Chang, K.-H. Chang, S.-Y. Kuo, "Applying Verification Intention for Design Customization via Property Mining under Constrained Testbenches", *ICCD'11*, pp. 84-89

[2] H. Ren, M. P. Kusko, V. N. Kravets and R. Yaari, "Low Cost Test Point Insertion without Using Extra Registers for High Performance Design", *ITC'09*, Paper 12.2

[3] Y. Savaria, M. Youssef, B. Kaminska, M. Koudil, "Automatic Test Point Insertion for Pseudo-random Testing", *ISCAS'91*, pp. 1960-1963.

[4] N. A. Touba, E. J. McCluskey, "Test Point Insertion Based on Path Tracing", *VTS'96*, pp. 2-8

[5] J.-S. Yang, B. Nadeau-Dostie, N. A. Touba, "Test Point Insertion Using Functional Flip-Flops to Drive Control Points", *ITC'09*, Paper 17.3

[6] Bug UnderGround, http://bug.eecs.umich.edu

[7] Berkeley Logic Synthesis and Verification Group, ABC: A System for Sequential Synthesis and Verification, http://www.eecs.berkeley.edu/∼alanmi/abc/abc.htm

[8] OpenCores, http://www.opencores.org